UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Mathematics

Doctoral School in
Mathematics

Complexity in Infinite Games on Graphs and Temporal Constraint Networks

Doctoral Dissertation

Author
Carlo Comin

Advisor
Prof. Romeo Rizzi

Co-Advisor
D.d.R. Stéphane Vialette

LABORATOIRE D'INFORMATIQUE
GASPARD-MONGE

Sous la co-tutelle de :
CNRS
ÉCOLE DES PONTS PARISTECH
ESIEE PARIS
UPEM • UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

January 2017

Complexity in Infinite Games on Graphs and Temporal Constraint Networks

A dissertation
presented to the Doctoral School in Mathematics
of the University of Trento
in partial fulfilment of the requirements for the
Ph.D. Degree

Sous la co-tutelle de
LIGM, Université Paris-Est Marne-la-Vallée

by
Carlo Comin
January 2017

Advisor
Prof. Romeo Rizzi
University of Verona, Italy

Co-Advisor
D.d.R. Stéphane Vialette
Universitè Paris-Est Marne-la-Vallée, France

Referee
Prof. Luke Hunsberger
Vassar College, New York (U.S.)

Referee
Prof. Angelo Montanari
University of Udine, Italy

Abstract

This dissertation deals with a number of algorithmic problems motivated by automated temporal planning and formal verification of reactive and finite state systems. Particularly, we shall focus on game theoretical methods in order to obtain improved complexity bounds and faster algorithms.

In the *first paper* we introduce *Hyper Temporal Networks (HyTNs)*, a strict generalization of Simple Temporal Networks (STNs), to overcome the limitation of considering only conjunctions of constraints but maintaining a practical efficiency in the consistency check of the instances. STNs provide a powerful and general tool for representing conjunctions of maximum delay constraints over ordered pairs of temporal variables. HyTNs are meant as a light generalization of STNs offering an interesting compromise. On one side, there exist practical pseudo-polynomial time algorithms for checking consistency and computing feasible schedules for HyTNs; the computational equivalence between checking consistency in HyTNs and determining winning regions in Mean Payoff Games (MPGs) is also pointed out. On the other side, HyTNs offer a more powerful model accommodating natural constraints that cannot be expressed by STNs like *"Trigger off exactly δ min before (after) the occurrence of the first (last) event in a set."*

Then, we turn our attention to the Conditional Simple Temporal Network (CSTN) model, a constraint-based graph-formalism for conditional temporal planning. Three notions of consistency arise for CSTNs: weak, strong, and dynamic. Dynamic consistency is the most interesting notion, but it is also the most challenging and it was conjectured to be hard to assess. CSTNs are an extension of Simple Temporal Networks (STNs) [44]. In the *second paper* we introduce and study the *Conditional Hyper Temporal Network (CHyTN)* model, a natural extension and generalization of both the CSTN and the HyTN model, obtained by blending them together. We show that deciding whether a CSTN is dynamically-consistent is coNP-hard; and that deciding whether a CHyTN is dynamically-consistent is PSPACE-hard, when the input instances are allowed to include both multi-head and multi-tail hyperarcs. Then, we offer the first deterministic (pseudo) singly-exponential time algorithm for the problem of checking the dynamic consistency of such CHyTNs, also producing a dynamic execution strategy whenever the input CHyTN is dynamically-consistent. The presentation of such connection is mediated by the HyTN model. In order to analyze the time complexity of the algorithm, we introduce a refined notion of dynamic consistency, named ϵ-dynamic consistency, and present a sharp lower bounding analysis on the critical value of the reaction time $\hat{\varepsilon}$ where a CHyTN

transits from being, to not being, dynamically-consistent.

The ε-DC notion turns out to be interesting per se, and the proposed ε-DC-Checking algorithm rests on the assumption that reaction-time satisfies $\varepsilon > 0$; leaving unsolved the question of what happens when $\varepsilon = 0$. In the *third paper*, we introduce and study π-DC, a sound notion of DC with an *instantaneous reaction-time* (i.e., one in which the planner can react to any observation at the same instant of time in which the observation is made). Firstly, we demonstrate by a counter-example that π-DC is not equivalent to 0-DC, and that 0-DC is actually inadequate for modeling DC with an instantaneous reaction-time. This shows that our previous results do not apply directly, as they were formulated, to the case of $\varepsilon = 0$. Then, our previous tools are extended in order to handle π-DC, and the notion of *ps-tree* is introduced, also pointing out a relationship between π-DC and HyTN-Consistency. Thirdly, a simple *reduction* from π-DC to DC is identified. This allows us to design and to analyze the first sound-and-complete π-DC-Checking procedure, whose time complexity remains (pseudo) singly-exponential in the number of propositional letters.

Next, an arena is a finite directed graph whose vertices are divided into two classes, i.e., $V = V_\Box \cup V_\bigcirc$; this forms the basic playground for a plethora of 2-player infinite pebble games. In the *fourth paper*, we introduce and study a refined notion of reachability for arenas, named *trap-reachability*, where Player $\Box$ attempts to reach vertices without leaving a prescribed subset $U \subseteq V$, while Player $\bigcirc$ works against. It is shown that every arena decomposes into *strongly-trap-connected components (STCCs)*. Our main result is a linear time algorithm for computing this unique decomposition. This theory has direct applications in solving Update Games (UGs) faster. Dinneen and Khoussainov showed in 1999 that deciding who's the winner in a given UG costs $O(mn)$ time, where n is the number of vertices and m is that of the arcs. We solve that problem in $\Theta(m+n)$ linear time. Finally, the polynomial-time complexity for deciding Explicit McNaughton-Müller Games is also improved, from cubic to quadratic.

Then, in the *fifth paper* we offer a $\Theta(|V|^2|E|\,W)$ pseudo-polynomial time and $\Theta(|V|)$ space deterministic algorithm for solving the Value Problem and Optimal Strategy Synthesis in Mean Payoff Games. This improves by a factor $\log(|V|\,W)$ the best previously known pseudo-polynomial time upper bound of Brim, *et al.* In the *sixth paper* we further strengthen the links between Mean Payoff Games (MPGs) and Energy Games (EGs). We offer a faster $O(|V|^2|E|\,W)$ pseudo-polynomial time and $\Theta(|V| + |E|)$ space deterministic algorithm for solving the Value Problem and Optimal Strategy Synthesis in MPGs. This improves significatly over our previous $\Theta(|V|^2|E|\,W)$ time algorithm, also in practice. Moreover, we study structural aspects concerning Optimal Positional Strategies (OPSs) in MPGs. We observe a *unique complete decomposition* of the space of all OPSs, $\texttt{opt}_\Gamma\Sigma^M_0$, in terms of so-called *extremal*-SEPMs in reweighted Energy Games; this points out what we called the "Energy-Lattice $\mathcal{X}^*_\Gamma$ of $\texttt{opt}_\Gamma\Sigma^M_0$". Lastly, it is offered a *pseudo-polynomial total-time* recursive procedure for *enumerating* (w/o repetitions) all the elements of $\mathcal{X}^*_\Gamma$ and $\texttt{opt}_\Gamma\Sigma^M_0$.

Acknowledgements

First and foremost I wish to thank my thesis advisor, Prof. Romeo Rizzi*, for his guidance and for the time and efforts he dedicated to my training during this journey. He has been a constant source of friendly support and inspiration without whom this thesis would not exist. Romeo has been the greatest mentor from all points of view.*
I was lucky and honored to have been one of his students.
I wish to thank my co-advisor, D.d.R. Stéphane Vialette*, for supporting me at UPEM and for having accompanied me throughout my stay in France. I came into contact with an outstanding research environment and with the best research seminars I had ever experienced. I also thank him for his advices which supported my path.*
I wish to thank the external referees of this dissertation, Prof. Luke Hunsberger *and* Prof. Angelo Montanari*, as well as* Dr. Pietro Sala *and* Dr. Nicola Gigante*, for their careful review and for their valuable comments which helped improving both the content and quality of the manuscript.*
I thank Dr. Anthony Labarre *for kindly sharing with me his office at UPEM for a period of time and for having worked together during my stay and later on.*
Thanks to the AlgoB Group *at UPEM,* Dr. Philippe Gambette, Dr. Karel Břinda, Dr. Emerite Both Neou*, as well as* Dr. Massimo Cairo *and all the Doctoral Fellows of the XXIX cycle in Maths at University Trento, for making this unique experience and for being great and friendly colleagues.*
Lastly, I would like to thank my family for all their love and encouragement.
A special thank goes to my mother, words cannot express all the sacrifices she has made on my behalf. I hope one day to be able to make a sense of all these.

Carlo Comin,
January 2017

Contents

1 Introduction and Context **5**

1.1 Contributions and Organization 6

1.2 Constraint Satisfaction Problems 8

1.2.1 Temporal Constraint Networks 10

1.2.2 Simple Temporal Networks 11

1.3 Algorithmic Game Theory 13

1.3.1 Topological Banach-Mazur Games 14

1.3.2 Gale-Stewart Games 15

1.3.3 Borel Determinacy Theorem 16

1.4 Modal μ-calculus, ω-Regular and Mean-Payoff Games 17

1.4.1 Syntax and Semantics of the Modal μ-calculus 17

1.4.2 ω-Regular Games 21

1.4.3 From model-checking of L_μ to parity games 25

1.4.4 Mean-payoff games 27

1.4.5 From parity games to mean-payoff games 28

I Temporal Constraint Networks **29**

2 Hyper Temporal Networks **30**

2.1 Introduction 31

2.1.1 Contribution 32

2.1.2 Organization 32

2.2 Motivating Examples 32

2.3 Background and Notation 38

2.4 HyTN and Consistency Property 39

2.5 Mean Payoff Games 47

2.6 The Reductions 51

2.7 Computational Experiments 60

2.8 Related Works 70

2.9 Conclusion 72

3 Checking Dynamic Consistency of Conditional Hyper Temporal Networks via Mean Payoff Games 74

3.1 Introduction and Motivation 76

3.1.1 Contribution 77

3.1.2 Organization 79

3.2 Background and Notation 79

3.2.1 Simple Temporal Networks 79

3.2.2 Hyper Temporal Networks 79

3.3 Conditional Simple / Hyper Temporal Networks 80

3.4 Algorithmics of Dynamic Consistency 87

3.4.1 ϵ-Dynamic Consistency 95

3.4.2 A (pseudo) Singly-Exponential Time Algorithm for DC and CHyTN-DC 99

3.5 Bounding Analysis on the Reaction Time $\hat{\epsilon}$ 105

3.6 Related Works 112

4 Instantaneous Reaction-Time in Dynamic Consistency Checking of Conditional Simple Temporal Networks 114

4.1 Introduction and Motivation 115

4.1.1 Contribution 115

4.2 Background and Notation 116

4.2.1 ε-Dynamic-Consistency 116

4.3 DC with Instantaneous Reaction-Time 117

4.3.1 The ps-tree: "skeleton" structure for π-dynamic π-ESs . . 121

4.3.2 Verifying a c-ps-tree. 123

4.3.3 A Singly-Exponential Time π-DC-Checking Algorithm . . 127

4.4 Related Works 132

4.5 Conclusion 133

II Infinite Games on Graphs 134

5 Linear Time Algorithm for Update Games via Strongly-Trap-Connected Components 135

5.1 Introduction 136

5.1.1 Background and Notation 137

5.2 Trap-Reachability 137

5.3 Strongly-Trap-Connectedness 140

5.4 TR-Depth-First-Search 142

5.5 Linear Time Algorithm for STCCs 154

5.5.1 Proof of Correctness of Algo. 13 161

5.6 Application to Update Games 163

5.7 Application to Explicit McNaughton-Müller Games 165

5.8 Conclusion 165

6 Improved Pseudo-Polynomial Upper Bound for the Value Problem and Optimal Strategy Synthesis in Mean Payoff Games 166
6.1 Introduction 167
6.1.1 Contribution 168
6.1.2 Organization 169
6.2 Background and Notation 169
6.3 Values and Optimal Positional Strategies from Reweightings . . . 175
6.3.1 On Optimal Values 175
6.3.2 On Optimal Positional Strategies 178
6.4 A $\Theta(|V|^2|E|W)$ time Algorithm for solving the Value Problem and Optimal Strategy Synthesis in MPGs 183
6.4.1 Description of the Algorithm 184
6.4.2 Proof of Correctness 188
6.4.3 Complexity Analysis 192
6.5 Conclusions 194

7 Faster $O(|V|^2|E|W)$-Time Energy Algorithm for Optimal Strategy Synthesis in Mean Payoff Games 195
7.1 Introduction 196
7.1.1 Contribution 196
7.1.2 Organization 197
7.2 Background and Notation 198
7.3 A Faster $O(|V|^2|E|W)$-Time Algorithm for MPGs by Jumping through Reweighted EGs 198
7.3.1 Description of Algorithm 15 199
7.3.2 Correctness of Algorithm 15 218
7.3.3 Complexity of Algorithm 15 238
7.3.4 An Experimental Evaluation of Algorithm 15 239
7.4 Conclusion 242

8 The Energy Structure of Optimal Positional Strategies in MPGs 243
8.1 Introduction 245
8.1.1 Contribution 245
8.1.2 Organization 246
8.2 An Energy-Lattice Decomposition of $\texttt{opt}_\Gamma\Sigma_0^M$ 246
8.3 A Recursive Enumeration of $\mathcal{X}_\Gamma^*$ and $\texttt{opt}_\Gamma\Sigma_0^M$ 251

1 Introduction and Context

This dissertation provides further evidence that game theoretic arguments help to study algorithmic problems in the area of automated temporal planning and formal verification of finite state non-terminating systems.

Automated temporal planning [44, 59, 88] is a branch of Artificial Intelligence (AI) that concerns the realization of temporal strategies or temporal action sequences, typically for execution by intelligent agents, autonomous robots and unmanned vehicles. Unlike classical control and classification problems, the solutions are complex and must be discovered and optimized in multidimensional space. Planning is also related to decision theory. In known environments with available models, planning can be done offline; solutions can be found and evaluated prior to execution. In dynamically unknown environments, the strategy often needs to be revised online; solutions usually resort to iterative trial and error processes commonly seen in AI.

On the other hand, *non-terminating* computing systems involving multiple, distributed, and interacting *agents* [106, 119] abound today and can be found in environments as varied as household appliances, medical equipment, industrial control systems, flight control systems in airplanes, etc. In these contexts, failures caused by design faults may be very costly and they should be avoided as much as possible. Behaviour of such systems is typically very complex which makes their design and validation a challenge. *Formal methods* try to address this challenge by developing formal models of such systems, and methods to specify and reason about their properties. A formal method is of particular interest if it offers not only a rigorous and unambiguous way to describe systems and their intended behaviour, but also provides efficient algorithms allowing to automate (parts of) the design and validation tasks.

We began our research by studying algorithmic problems in automated temporal planning, particularly, our original motivation was the study of certain *conditional* temporal planning problems. At some point, we have identified interesting connections between the algorithmics of these problems and that of some fundamental tasks related to a specific family of infinite 2-player pebble games that are played on finite graphs, i.e., the Mean Payoff Games (MPGs). These games, in addition to being of an independent interest, are intimately related to the semantics of a well-known model of calculus for formal verification, e.g., the modal μ-calculus. In summary, it turned out that a game theoretic formulation helps to abstract away from syntactic and semantic peculiarities of modelling formalisms and makes the conditinal temporal constraints problems in question more easily amendable to algorithmic and complexity analysis.

This led us, on one side, to adopt a game theoretic viewpoint for studying those conditional temporal planning problems, ultimately obtaining faster algorithms and improved complexity bounds; and then, on the other side, these first encouraging results have led us to deepen the study of algorithmic and complexity issues in infinite games on finite graphs per se. In the end, we obtained faster algorithms and improved complexity bounds for some of these games, i.e., Update Games, Explicit McNaughton-Müller Games, and finally, for Mean Payoff Games. Our contributions have thus an algorithmic nature, focused on improving state-of-the-art computational complexity upper bounds.

1.1 Contributions and Organization

This dissertation comprises an introductory chapter and then two major parts. Chapter 1 provides context and background notions on the covered subjects, plus an outline of the main contributions (see below).

Part I presents our contributions in automated temporal planning, it contains a revised version of the following articles:

- Carlo Comin, Roberto Posenato, Romeo Rizzi. A Tractable Generalization of Simple Temporal Networks and its relation to Mean Payoff Games. Accepted in *21st International Symposium on Temporal Representation and Reasoning (TIME 2014). University of Verona, Verona, Italy. September 2014.* [32]

- Carlo Comin, Roberto Posenato, Romeo Rizzi. Hyper Temporal Networks. Accepted in *Constraints, an International Journal, Springer-US, pp 1-39. March 2016.* [33]

- Carlo Comin, Romeo Rizzi. Dynamic Consistency of Conditional Simple Temporal Networks via Mean Payoff Games: a Singly-Exponential Time DC-Checking. Accepted in *22nd International Symposium on Temporal Representation and Reasoning (TIME 2015), University of Kassel, Kassel, Germany. September 2015.* [34]

- Carlo Comin, Romeo Rizzi. Checking Dynamic Consistency of Conditional Hyper Temporal Networks via Mean Payoff Games (Hardness and pseudo-Singly-Exponential Time Algorithm). Accepted in *Information and Computation, Elsevier. (It will appear during 2017)* [40]

- Massimo Cairo, Carlo Comin, Romeo Rizzi. Instantaneous Reaction-Time in Dynamic-Consistency Checking of Conditional Simple Temporal Networks. Accepted in *23rd International Symposium on Temporal Representation and Reasoning (TIME 2016), Technical Univeristy of Denmark (DTU), Copenhagen, Denmark, October 2016.* [17]

Part II presents our contributions concerning algorithmic and complexity issues in infinite 2-player pebble games on graphs, it contains a revised version of the following articles:

- Carlo Comin, Romeo Rizzi. Linear Time Algorithm for Update Games via Strongly-Trap-Connected Components (A 2-Player Infinite Pebble Game Generalization of Strongly-Connected Components). Submitted. [39]

- Carlo Comin, Romeo Rizzi. Energy Structure and Improved Complexity Upper Bound for Optimal Positional Strategies in Mean Payoff Games. Accepted in *3rd International Workshop on Strategic Reasoning (SR 2015), University of Oxford, Oxford, U.K., September 2015.* [35]

- Carlo Comin, Romeo Rizzi. Improved Pseudo-Polynomial Bound for the Value Problem and Optimal Strategy Synthesis in Mean Payoff Games. Accepted in *Algorithmica, Springer-US, pp 1-27, February 2016.* [38]

- Carlo Comin, Romeo Rizzi. Faster $O(|V|^2|E|W)$-Time Energy Algorithm for Optimal Strategy Synthesis in Mean Payoff Games. Submitted. [37]

During the doctoral course, the author of this dissertation also co-authored the following articles; which however do not belong to this dissertation, being motivated by topics in computational biology.

- Carlo Comin, Anthony Labarre, Romeo Rizzi, Stéphane Vialette. Sorting with Forbidden Intermediates. Accepted in *3rd International Conference on Algorithms for Computational Biology (AlCoB 2016), Trujillo, Spain. June 2016 [31].* An extended version of this paper has been submitted to the *IEEE/ACM Transactions on Computational Biology and Bioinformatics.*

- Carlo Comin, Romeo Rizzi. An Improved Upper Bound on Maximal Clique Listing via Rectangular Fast Matrix Multiplication. *Accepted in Algorithmica, Springer-US. (It will appear during 2017)* [36]

For completeness, we mention another published contribution related to the theory of automata; but the result contained in the article below has not been obtained during the doctoral course, it belongs to author's MSc thesis.

- Carlo Comin. Algebraic Characterization of the Class of Languages Recognized by Measure Only Quantum Automata. Accepted in *Fundamenta Informaticae, Annales Societatis Mathematicae Polonae, IOS Press, 335–353, vol 134, 2014.* [29]

In summary, algorithmic game theory (especially, infinite two-player pebble-games played on finite graphs) is the red thread that makes it possible to look at the various contributions of this dissertation in a sufficiently coherent way. In the following sections we provide some further information to better outline the context in which our results can be placed.

1.2 Constraint Satisfaction Problems

The notion of constraint is central to a number of human activities and disparate processes. A *constraint* limits (or restricts) the field of possibilities in a certain universe. For example, a school timetable that coordinates students, teachers, lessons, rooms and time slots, must satisfy many constraints, i.e., not all combinations are possible since the involved constraints may be numerous and various. Besides school timetabling, constraint satisfaction problems arise in many enterprise and industrial tasks, ranging from scheduling to configuration, circuit design and molecular biology [78]. Also in Artificial Intelligence (AI) and Operations Research (OR), *constraint satisfaction* is the process of finding a solution to a set of constraints imposing conditions that the variables must satisfy. A solution is therefore a set of values for the variables that satisfies all the constraints – that is, a point in the feasible region. Formally, we shall consider the following model:

Definition 1.1 ([101]). *A* constraint satisfaction problem (CSP) *is a triplet,*

$$\Phi \triangleq (\mathbf{X}, \mathbf{D}, \mathbf{C}),$$

where:

- $\mathbf{X} \triangleq \{X_1, X_2, \ldots, X_n\}$ *is a set of* variables;
- $\mathbf{D} \triangleq \{D_1, D_2, \ldots, D_n\}$ *is a set of nonempty domains;*
- $\mathbf{C} \triangleq \{C_1, C_2, \ldots, C_m\}$ *is a set of* constraints.

Each variable X_i will take on its value in the nonempty domain D_i, i.e., D_i is the domain of possibles values of X_i. Each constraint C_j involves some subset of the variables and specifies the allowable combination of values for that subset, i.e., C_j is in turn a pair (T_j, R_j) where $T_j \subseteq X$ is a subset of k variables and R_j is a k-ary relation on the corresponding subset of domains. A state Ψ *of the CSP Φ is defined by an assignment of values to some or all of the variables, i.e.,*

$$\Psi \triangleq \{X_i = v_i, X_j = v_j, \ldots\}, \textit{ for some } v_i \in D_i, v_j \in D_j, \ldots$$

An assignment that doesn't violate any constraint is consistent *(or* feasible*), where a constraint (T_j, R_j) is* satisfied *if the values assigned to the variables T_j satisfy the relation R_j. A* complete *assignment is one in which every variable is mentioned, and a* solution *to a CSP is a complete assignment that satisfies all the constraints.*

Notable examples of problems that can be modeled as a CSP include the *eight queens puzzle.*

Example 1.1. *This is the problem of placing eight chess queens on an* 8×8 *chessboard so that no two queens threaten each other, where a solution requires that no two queens share the same row, column, or diagonal. The problem can be recast as one in which exactly one queen is assigned to each of the chessboard's columns, and the solver needs* *to find out only which row each of these queens is going to be placed in. Thus we have* 8 *variables* $\mathbf{X} \triangleq \{Q_1, Q_2, \ldots, Q_8\}$*, each of which can assume the value within the domain* $D_i \triangleq \{1, \ldots, 8\}$*. Given that a feasible solution is a configuration in which no two queens can attack each other, we have the following constraints: horizontally, no two queens should be in the same row, so:*

$$Q_i \neq Q_j \textit{ whenever } i \neq j;$$

along any diagonal, they should not be the same number of columns apart as they are rows apart, so:

$$|Q_i - Q_j| \neq |i - j| \textit{ whenever } i \neq j.$$

Thus there are 56 *constraints. Of course the puzzle can be generalized, with n queens on a chessboard of* $n \times n$ *squares (the reader is referred to e.g. [3] for a survey of known results and research open problems concerning the* n-queens puzzle*).*

The identification of CSPs as a general class is due to *Ugo Montanari* [88], who also pointed out the notion of *constraint network* and *propagation by path consistency*. Recall that a n-ary relation over the variables $X_1, X_2, \ldots, X_n$ is any subset of the corresponding domains $\mathsf{D} \triangleq D_1 \times D_2 \times \ldots \times D_n$. Let us denote by R_{ij} any binary relation between X_i and X_j, where we allow $R_{ij} \neq R_{ji}$ generally. Given $i \in [n]$, the identity relation I_{ii} exists, defined only between a variable X_i and itself, as follows: $I_{ii} \triangleq \{(d, d) \mid d \in D_i\}$. Also, it is worth mentioning the unit relation $U_{ii} \triangleq D_i \times D_i$. The formal definition of *constraint network* (i.e., a network of binary constraints) goes as follows.

Definition 1.2 ([88])**.** *A* constraint network $\mathcal{N} \triangleq (\mathsf{D}, \{R_{i,j}\}_{i,j})$ *is made of a family of sets,*

$$\mathsf{D} \triangleq \{D_1, D_2, \ldots, D_n\},$$

plus a relation R_{ij} *from every set* D_i *to every set* D_j*, for* $i, j \in \{1, 2, \ldots, n\}$*.*

Furthermore, $R_{ii} \subseteq I_{ii}$ *for every i.*

A constraint network can be thought of as representing a n-ary relation ρ, where the n-tuple $\mathbf{a} \in \rho$ iff its projections on all the two-dimensional subspaces of D satisfy the binary constraints of $\mathcal{N}$. A useful way of visualizing a network is by a directed graph in which vertices $v_1, v_2, \ldots, v_n$ correspond to sets $D_1, D_2, \ldots, D_n$, and an arc (v_i, v_j) is present iff $R_{ij} \neq U_{ij}$ (when $i \neq j$) or $R_{ii} \neq I_{ii}$ [88]. For instance,

Example 1.2. *The following n-ary* ($n = 3$) *relation* ρ *is represented by the network in Fig. 1.1; let* $D_i = \{x_{i1}, x_{i2}, x_{i3}\}$ *for* $i = 1, 2, 3$*, then let:*

$$\rho \triangleq \{(x_{11}, x_{21}, x_{31}), (x_{11}, x_{21}, x_{32}), (x_{12}, x_{23}, x_{31})\}.$$

As observed in [88], the class of n-ary relations representable by constraint networks is much narrower than the class of all n-ary relations. The reader is referred to [88] in order to get more details and fundamental properties concerning constraint networks.

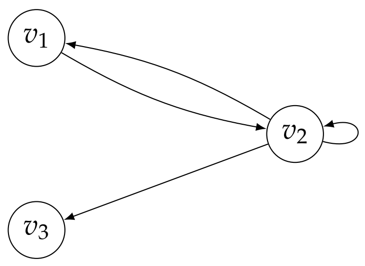


Figure 1.1: The constraint network ρ of Example 1.2.

1.2.1 Temporal Constraint Networks

In [44], network-based methods of constraint satisfaction are extended to include continuous variables, thus providing a framework for processing temporal constraints.

Definition 1.3 ([44]). *A* temporal constraint network (TN) Γ *is a constraint network involving a set of variables* $\mathbf{X} \triangleq \{X_1, X_2, \ldots, X_n\}$ *having continuous (real-valued) domains* $\mathbf{D} \triangleq \{D_1, D_2, \ldots, D_n\}$*; each variable represents a time point. Furthermore, a set* $\mathbf{C}$ *of unary and binary constraints is involved. Each (unary or binary) constraint* $C_j \in \mathbf{C}$ *is represented by a set of real intervals:*

$$C_j \triangleq \{I_1, I_2, \ldots, I_k\} = \big\{[a_1, b_1], [a_2, b_2], \ldots, [a_k, b_k]\big\}.$$

Closed, open and semi-open intervals are generally allowed. The intended interpretation going as follows.

- *A* unary *constraint* T_{ii} *restricts the domain of variable* X_i *to the given set of intervals; namely, it represents the disjunction:*

$$\bigvee_{q=1}^{k} \Big(a_q \leq X_i \leq b_q\Big).$$

- *A* binary *constraint* T_{ij} *constrains the permissible values for the distance* $X_j - X_i$*; namely, it represents the disjunction:*

$$\bigvee_{q=1}^{k} \Big(a_q \leq X_j - X_i \leq b_q\Big).$$

Constraints are given in a canonical form *where all intervals are pairwise disjoint.*

A TN can be represented by a directed constraint graph, in much the same way as we did above for constraint networks, where vertices represent variables and an arc (X_i, X_j) indicates that a proper constraint T_{ij} is specified; but for TN, in addition, the arc is labeled by the interval set. An example is shown in Fig. 1.2.

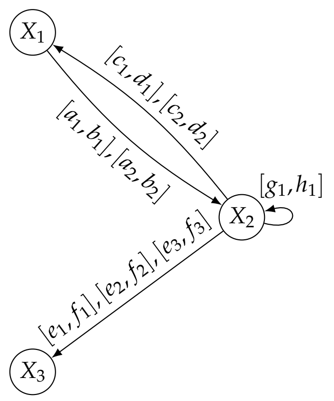


Figure 1.2: A Temporal Constraint Network.

Usually, a special time point Z (or X_0) is employed to represent the beginning of the world, i.e., for simplicity Z is always scheduled at time $t_Z = 0$. All other times are relative to Z; thus we may treat each unary constraint T_{ii} as a binary constraint T_{0i} (having the same interval representation).

1.2.2 Simple Temporal Networks

Definition 1.4. *A* simple temporal problem (STP) *(or* simple temporal network (STN)*) is a TN in which all constraints specify a single interval; namely, for each constraint C_j, there is an interval I_j such that $C_j = \{I_j\}$.*

In STNs, each arc (X_i, X_j) is labeled by an interval $[a_{ij}, b_{ij}]$, representing the constraint:

$$a_{ij} \leq X_j - X_i \leq b_{ij}.$$

Alternatively, the same constraint can be expressed as a pair of inequalities:

$$X_j - X_i \leq b_{ij} \text{ and } X_i - X_j \leq -a_{ij}$$

Notice that solving a STP/STN amounts to solving a set of linear inequalities on the X_i, where each inequality involves exactly two variables; thus a shortest-path algorithm on graphs, such as Bellman-Ford [4,42,55], can be applied.

Particularly, we shall consider graphs that are directed and weighted on the arcs. Thus,

Definition 1.5. *If $G = (V, A)$ is a graph, then every arc $a \in A$ is a triple (t_a, h_a, w_a) where $t_a \in V$ is the tail of a, $h_a \in V$ is the head of a, and $w_a \in \mathbb{R}$ is the weight of a.*

Moreover, since we use graphs to represent distance constraints, they do not need to have either loops (unary constraints are meaningless) or parallel arcs (two parallel constraints represent two different distance constraints between the same pair of node: only the most restrictive is meaningful). We also use the notations $h(a)$ for h_a, $t(a)$ for t_a, and $w(a)$ or $w(t_a, h_a)$ for w_a, when it helps.

The order and size of a graph $G = (V, A)$ are denoted by

$$n \triangleq |V| \text{ and } m \triangleq |A|,$$

respectively. The size is a good measure for the encoding length of G.

Definition 1.6. *A* cycle *of G is a set of arcs $C \subseteq A$ cyclically sequenced as $a_0, a_1, \dots, a_{\ell-1}$ so that,*

$$h(a_i) = t(a_j) \iff j = (i+1) \mod \ell;$$

it is called a negative cycle *if $w(C) \le 0$, where $w(C)$ stands for $\sum_{e \in C} w_e$.*

Definition 1.7. *A graph is called conservative when it contains no negative cycle.*

A potential is a function $p : V \mapsto \mathbb{R}$. The reduced weight of an arc $a = (u, v, w_a)$ with respect to a potential p is defined as

$$w_a^p \triangleq w_a - p_v + p_u.$$

A potential p of $G = (V, A)$ is called feasible if $w_a^p \ge 0$ for every $a \in A$. Notice that, for any cycle C, $w^p(C) = w(C)$. Therefore, the existence of a feasible potential implies that the graph is conservative as $w(C) = w^p(C) \ge 0$ for every cycle C.

So, the Bellman-Ford algorithm [4,42,55] can be used to produce in $O(nm)$ time:

- either a proof that G is conservative in the form of a feasible potential;
- or a proof that G is not conservative in the form of a negative cycle C in G.

When the graph is conservative, the smallest weight of a walk between two nodes is well defined, and, fixed a root node r in G, the potentials returned by the Bellman-Ford algorithm are, for each node v, the smallest weight of a walk from r to v. Moreover, if all the arc weights are integral, then these potentials are integral as well. Therefore, the Bellman-Ford algorithm provides a proof to the following theorem.

Theorem 1.1 ([4, 42, 55])**.** *A graph admits a feasible potential if and only if it is conservative. Moreover, when all arc weights are integral, the feasible potential is an integral function.*

Thus an STN can be viewed as a weighted graph whose nodes are time-points that must be placed on the real line and whose arcs express mutual constraints on the allocations of their end points.

Definition 1.8. *An STN $G = (V, A)$ is called consistent if it admits a* feasible scheduling, *i.e., a scheduling $s : V \mapsto \mathrm{R}$ such that*

$$s(v) \le s(u) + w(u,v), \quad \forall \text{ arc } (u,v) \text{ of } G.$$

Then we have the following characterization result for STN's consistency in terms of conservative graphs.

Corollary 1.1 ([4,42,44])**.** *An STN G is consistent if and only if G is conservative.*

Proof. A feasible scheduling is just a feasible potential. Therefore, this corollary is just a restatement of Theorem 1.1. □

The reader is referred to [44] for further details and fundamental properties concerning the theory of TNs and STNs.

1.3 Algorithmic Game Theory

As argued in [120], the birth of *Algorithmic Game Theory (AGT)* is often equated with the following three seminal papers, cited in [56] for laying the foundation of growth in AGT:

- *Koutsoupias, Papadimitriou, Worst-case Equilibria. STACS 1999: 404-413* [74], introduced the notion of *"price of anarchy"*, a measure of the extent to which competition approximates cooperation, quantifying how much utility is lost due to selfish behaviors on the Internet, which operates without a system designer or monitor striving to achieve the social optimum.

- *Roughgarden, Tardos: How Bad is Selfish Routing? FOCS 2000: 93-102* [100], studied the power and depth of the *"price of anarchy"* as it applies to routing traffic in large-scale communications networks to optimize the performance of a congested network.

- *Nisan, Ronen: Algorithmic Mechanism Design. STOC 1999: 129-140* [91], studied classical mechanism design from algorithmic and complexity-theoretic perspectives.

However the study of algorithmic questions in combinatorial games goes back a long time ago. Perphaps the first combinatorial game described in a mathematical form (in Europe) dates back to the beginning of the XVII century, to the time when *Bachet de Méziriac* proposed in his *"Problémes Plaisants"* the following game: two players alternately choose numbers between 1 and 10; the player, on whose move the sum attains exactly 100, is the winner. This kind of game, whose natural generalization nowadays is known as *Nim*, was also studied by *Bouton (1901-2) [11]* and it has an extensive literature [6]. In Nim, under the normal play convention, the game is between two players and it is played with n heaps (i.e., a pile of n counters) of any number of objects. The two players alternate by taking (removing), at each turn, any number of objects from any single one of the heaps at their choice. The winning condition is to be the last to take an object. So, Nim can be played as a *normal play game*, i.e., one in which the player who makes the last move wins. Also, Nim is *impartial* since the allowable moves depend only on the position and not on which of the two players is currently moving, and the payoffs are symmetric; otherwise stated, the only difference between Player 0 and Player 1 is that Player 0 goes first. "Normal play Nim" is fundamental to the *Sprague–Grundy Theorem* [63], which asserts that every impartial game under the normal play convention is equivalent to a *nimber* (i.e., the value of a Nim heap of a certain size) [41].

The *nimbers* are the ordinal numbers,

$$0, 1, 2, \ldots, n, \ldots, \omega, \omega+1, \omega+2, \ldots, \omega\cdot 2, \omega\cdot 2+1, \omega\cdot 2+2, \ldots, \omega^2, \ldots, \omega^3, \ldots, \omega^\omega, \ldots, \omega^{\omega^\omega}, \ldots$$

endowed with a new *nimber addition* and *nimber multiplication*, which are distinct from ordinal addition and ordinal multiplication; see [41] for more details. Some other connections between positional games and the infinite soon emerged in the literature.

1.3.1 Topological Banach-Mazur Games

The infinite positional games of perfect information were discovered and initially studied in Poland around the '30s. In 1935 *Stefan Banach* started a notebook, called the *Scottish Book*, where the mathematicians residing in or visiting Lwów proposed various mathematical problems, or conjectures, and also indicated their partial or complete solutions. In the same year *Stanisław Mazur* proposed an infinite combinatorial game. The game is described in Problem 43 of the Scottish Book; its solution, given by Banach, is dated August 4, 1935. Hence, the game became known as the *Banach-Mazur Game*. Mazur discovered the game in 1928; however, later on, *Ulam* [114] gives the year 1935 for its complete solution, referring to a conversation in the Scottisch Café, where "Mazur proposed the first examples of infinite mathematical games"). Later on Ulam prepared an English translation of the Scottish Book, see [115].

Definition 1.9 (Banach-Mazur Games on $\mathbb{R}$). *In a* Banach-Mazur Game, *played on the real line, the winning condition is given by a set $\mathtt{Win} \subseteq \mathbb{R}$ of real numbers; in the first move, Player 0 selects an interval d_0 on the real line, then Player 1 chooses an interval $d_1 \subsetneq d_0$, then Player 0 chooses a further refinement $d_2 \subsetneq d_1$ and so on. Thus a play forms an infinite (proper) chain sequence:*

$$d_0 \supsetneq d_1 \supsetneq d_2 \supsetneq \cdots .$$

Player 0 wins iff the intersection of all intervals d_i contains a point of $\mathtt{Win}$; namely,

$$\text{Player 0 wins} \iff \bigcap_{n\in\mathbb{N}} d_n \cap \mathtt{Win} \neq \emptyset.$$

A similar game can be played on a generic topological space $\mathbf{X}$. Let $\mathbf{V}$ be a family of subsets of $\mathbf{X}$ such that: each $V \in \mathbf{V}$ contains a nonempty open subset of $\mathbf{X}$; and each nonempty open subset of $\mathbf{X}$ contains an element $V \in \mathbf{V}$. In the Banach-Mazur game defined on $(\mathbf{X}, \mathbf{V})$ with winning condition $W \subseteq \mathbf{X}$, the two players take turns to choose sets $V_0 \supsetneq V_1 \supsetneq V_2 \supsetneq \cdots$ in $\mathbf{V}$, where Player 0 wins iff $\bigcap_{n\in\mathbb{N}} V_n \cap \mathtt{Win} \neq \emptyset$.

Definition 1.10 (Determinacy). *A game is said to be* determined *if one or the other of the players has a winning strategy, i.e., a selection of moves granting him the victory, no matter how the opponent plays.*

Concerning the determinacy of these games, the original proof of Banach never appeared. The first published proof is that of *Oxtoby Theorem* [94], which characterizes determinacy of *topological Banach-Mazur games* in terms of topological properties of the winning condition. The term *"topological game"* was introduced by Berge [5].

Observe that Banach-Mazur's infinite game can also be played on graphs:

Definition 1.11 (Banach-Mazur Games on Graphs). *It is given (G, v_s), where $G = (V, A)$ is a directed graph and $v_s \in V$ is a distinguished initial vertex, it is assumed that G has no sink vertices. A winning condition $\texttt{Win}$ is any subset of infinitely long paths in G, each starting at v, i.e., any $\texttt{Win} \subseteq Paths(G, v_s)$. The game $(G, v_s, \texttt{Win})$ starts at vertex v_s with a move of Player 0 and the players strictly alternate. In a move, after a sequence of moves $p_0, p_1, \ldots, p_{m-1}$ forming a finite path, $p_0 p_1 \cdots p_{m-1}$, that has already been played, the corresponding Player $(i \mod 2)$ prolongs the path by choosing another finite path p_i whose initial vertex is the end vertex of p_{m-1}. Thus, a play results into an infinite path:*

$$\pi \triangleq p_0 p_1 \cdots p_m \cdots \in Paths(G, v),$$

the winning condition being that:

$$\text{Player 0 wins} \iff \pi \cap \texttt{Win} \neq \emptyset;$$

otherwise, Player 1 wins.

This reformulation shows that Banach-Mazur games are an interesting starting point for the exploration of properties of infinite games on graphs. Many infinite games on graphs and their determinacy properties find a natural place in a hierarchy known as the *Borel Hierarchy*, whereas the fundamental theorem concerning determinacy in infinite games is the *Borel Determinacy Theorem*; the latter can be formulated in terms of *Gale-Stewart Games* which are recalled in the next subsection.

1.3.2 Gale-Stewart Games

In 1953 *David Gale* and *Frank Stewart* introduced the following positional infinite game of perfect informationin [57]. This allowed them to observe fruitful connections between set theory and infinite games, particularly, this led to important applications of the notion of determinacy in the foundations of set theory [83,90].

Definition 1.12 (Gale-Stewart Games). *Let Σ be an* alphabet*, i.e., a finite nonempty set of symbols. A Gale-Stewart game on Σ is a pair $\Gamma \triangleq (\Sigma, \texttt{Win})$, where $\texttt{Win} \subseteq \Sigma^\omega$ is called the* winning condition*. The two players alternate turns, and each player is aware of all moves before making the next one. On each turn, Player i (for $i = 0, 1$) chooses a single element of Σ (i.e., a* position*) to play. But the same element may be chosen more than once without restriction. The play continues ad infinitum, so that a* *single* play *of the game determines an infinite sequence:*

$$\pi \triangleq (a_0, a_1, \ldots, a_n, \ldots) \in \Sigma^\omega.$$

At this point, Player 0 wins if $\pi \in$ `Win`; otherwise, the winner is Player 1.

Definition 1.13 (Open Gale-Stewart Games)**.** *Given a word $p \in \Sigma^*$, the following subset $[p] \subseteq \Sigma^\omega$ is named a* cone*:*

$$[p] \triangleq \{\pi \in \Sigma^\omega \mid p \text{ is a prefix of } \pi\}.$$

We say that,

$$U \subseteq \Sigma^\omega \text{ is open} \iff \text{every } \pi \in U \text{ has a prefix } p \text{ such that } [p] \subseteq U.$$

If $\mathcal{O}$ is the family of all open subsets of Σ^ω, then $(\Sigma^\omega, \mathcal{O})$ is called the Cantor topology on Σ^ω. *A Gale-Stewart game $\Gamma \triangleq (\Sigma,$ `Win`$)$ is called* open (closed) *whenever* `Win` *is so in the Cantor topology on Σ^ω.*

The main result of Gale and Stewart [57] is dated 1953, going as follows.

Theorem 1.2 (Gale-Stewart Determinacy Theorem [57])**.** *Every open or closed Gale-Stewart game is determined.*

In the next twenty years this result was extended to slightly higher levels of the Borel hierarchy. At some point this led to the question of whether or not the Gale-Stewart game is determined whenever the payoff set is *Borel*, as recalled next.

1.3.3 Borel Determinacy Theorem

The question of whether Borel Gale-Stewart games are determined was answered by *Donald A. Martin* [81] in 1975. Let us firstly recall some basic notions from topology.

Definition 1.14. *Let Γ be a topological space. A* Borel set *is any set in Γ that can be formed from open sets through the operations of countable union, countable intersection, and relative complement (set difference).*

Notice that the collection of all Borel sets of Γ forms a σ-algebra; indeed, the Borel sets are the smallest σ-algebra of subsets of Σ^ω containing all open sets in $\mathcal{O}$.

At this point, Borel sets are classified in the *"Borel Hierarchy"* according to how many times the operations of complement and countable union are required to produce them from open sets.

Definition 1.15. *In the* Borel Hierarchy *there are three classes for every countable ordinal $\alpha > 0$:*

$$\Sigma^0_\alpha, \Pi^0_\alpha, \text{ and } \Delta^0_\alpha.$$

- *A set S is in Σ^0_1 iff S is open in $(\Sigma^\omega, \mathcal{O})$.*

- *A set S is in Π^0_α iff the complement of S is in Σ^0_α.*
- *A set S is in Σ^0_α for $\alpha > 1$ iff there exists a sequence of sets $A_1, A_2, \ldots$ such that each A_i is in $\Pi^0_{\alpha_i}$ for some $\alpha_i < \alpha$ and $A = \cup_{i \geq 1} A_i$.*
- *A set is in Δ^0_α iff it is both in Σ^0_α and Π^0_α.*

Definition 1.16. *When* `Win` *is* Borel, *the Gale-Stewart game $(\Sigma, \texttt{Win})$ is called* Borel.

The *Borel Determinacy Theorem* states that any Borel Gale-Stewart game is determined, meaning that one of the two players will have a winning strategy for the game.

Theorem 1.3 (Borel Determinacy Theorem [81, 82])**.** *Every Borel Gale-Stewart game is determined.*

As already mentioned, this was proved by Martin [81] in 1975. The original proof was quite complicated, the same author published a shorter purely inductive proof in 1982 [82].

The Borel Determinacy Theorem provides a framework for the determinacy question of a number of distinct but interrelated infinite game models.

Furthermore, the problem of verifying correctness and temporal aspects of non-terminating computing systems involving multiple, distributed, and interacting agents [106] soon led to the study of interesting model-checking questions [22]. Some of these problems turned out to be equivalent to the determinacy question for certain infinite games lying at a low level (3rd level) of the Borel Hierarchy, e.g., the *parity games* [61,122]. Some of these developments are mentioned in the next subsections.

1.4 Modal μ-calculus, ω-Regular and Mean-Payoff Games

1.4.1 Syntax and Semantics of the Modal μ-calculus

The modal μ-calculus is an extension of propositional modal logic, particularly, it is a logic that combines simple modal operators with fixed point operators to provide a form of recursion. The μ-calculus originates with *Dana Scott* [105] and *Jaco de Bakker* [43], later on was further developed by *Dexter Kozen* [75]. It can be viewed as a logic describing properties of *transition systems*, i.e., potentially infinite graphs with labeled arcs (*transitions*) and vertices (*states*). Transitions are labeled with actions drawn from, $\mathcal{A} \triangleq \{a, b, c, \ldots\}$, and states are labeled with sets of *propositions*, drawn from $\mathcal{P} \triangleq \{p_1, p_2, \ldots\}$.

Thus a *transition system* can be viewed as a tuple:

$$\mathcal{M} \triangleq \Big(S, \{R_a\}_{a \in \mathcal{A}}, \{P_i\}_{i \in \mathbb{N}}\Big),$$

where S is a set of states, $R_a \subseteq S \times S$ is a binary relation defining transitions for every action $a \in \mathcal{A}$, and there's a set $P_i \subseteq S$ for each proposition $p_i \in \mathcal{P}$. A pair $(s, s') \in R_a$ is called *a-transition*, an *a-path* is a sequence of pairs

$(s_0,s_1),(s_1,s_2),\ldots$ all lying in R_a. Actually, there is a great interest in efficient solutions of the model-checking and the satisfiability problems of the modal μ-calculus, see [13] for a survey. The syntax of modal μ-calculus is formally recalled next.

Definition 1.17 (Syntax of the modal μ-calculus [13,75,105])**.** *Let* `Var` *be a countable set of variables (whose meaning will be sets of states). A* formula *α of the modal μ-calculus is defined by the following grammar L_μ:*

$$L_\mu \triangleq X \mid p \mid \neg p \mid \alpha \wedge \beta \mid \langle a\rangle\alpha \mid [a]\alpha \mid \mu X.\alpha \mid \nu X.\alpha;$$

where $X \in$ `Var`*, $p \in \mathcal{P}$, $a \in \mathcal{A}$, and α,β range over formulas of L_μ. The formulas $\mu X.\alpha$ and $\nu X.\alpha$ are named* fixpoint *formulas; and the symbols μ and ν are the* least *and* greatest *fixpoint operators, respectively. Disambiguating parentheses are added when necessary, where $[a]$ and $\langle a\rangle$ bind more tightly than boolean operators, μ and ν bind loosely.*

Example 1.3. *The formula (sentence) $\nu Y.\big(\mu X.((p \wedge \langle a\rangle Y) \vee \langle a\rangle X)\big)$ can be written as:*

$$\nu Y.\mu X.(p \wedge \langle a\rangle Y) \vee \langle a\rangle X.$$

One may adopt a series of syntactic tricks to simplify the notation: let us write $\sigma X.\alpha$ for $\mu X.\alpha$ or $\nu X.\alpha$; also, to underline the dependency of the value of a formula α on a variable X, $\sigma X.\alpha(X)$ stands for $\sigma X.\alpha$; in any formula like $\sigma X.\alpha(X)$, we say that X is a *bound* variable; a variable X is *free* in a formula $\alpha(X)$ if it is not bound; a *sentence* is a formula with no free variables occuring in it; finally, by $\alpha[\beta/X]$ we denote the result of substituting β for every free occurrence of X in α. Note one can always make sure that no variable has at the same time a free and a bound occurrence in a formula, as clearly $\sigma X.\alpha$ is equivalent to $\sigma Y.\alpha[Y/X]$; so, w.l.o.g., bound and free variables are always different. Also, we can require that every variable is bound at most once in a formula. A formula is *well-named* when both of the latter two conditions hold. Moreover, we can even ensure that in every formula $\mu X.\alpha(X)$ the variable X appears only once in $\alpha(X)$, as $\sigma X.\sigma Y.\alpha(X,Y)$ is equivalent to $\sigma X.\sigma X.\alpha(X,X)$.

Let us proceed by recalling the notion of *unfolding* of a formula, which will be useful in the reminder of this section: fix $\sigma \in \{\mu,\nu\}$, any fixpoint formula $\sigma X.\alpha(X)$ is equivalent to its unfolding, namely, $\alpha(\sigma X.\alpha(X))$; indeed, this equivalence follows directly from the definition of the fixpoint operators μ and ν. Note that there is no negation operation in the syntax, just negations of propositions; however, the operation of the negation of a sentence will turn out to be definable.

A meaning of a formula in a transition system is a set of states satisfying the formula, the meaning of a variable will be also a set of states of the transition system.

Definition 1.18 (Denotational Semantics of the modal μ-calculus [13,75,105])**.**

Given a transition system,

$$\mathcal{M} \triangleq \Big(S, \{R_a\}_{a\in\mathcal{A}}, \{P_i\}_{i\in\mathbb{N}}\Big),$$

and a valuation,

$$\mathcal{V} : \texttt{Var} \to 2^S,$$

the meaning $[\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}}$ *of a formula* α *of* L_μ *is defined by induction on its structure:*

- *The meaning of any variable* $X \in \texttt{Var}$ *is* $[\![X]\!]^{\mathcal{M}}_{\mathcal{V}} \triangleq \mathcal{V}(X)$;
- *The meaning of a propositional constant* $p_i \in \mathcal{P}$ *and its negation* $\neg p_i \in \mathcal{P}$ *is defined (respectively) by:*

$$[\![p_i]\!]^{\mathcal{M}}_{\mathcal{V}} \triangleq P_i \text{ and } [\![\neg p_i]\!]^{\mathcal{M}}_{\mathcal{V}} \triangleq S \setminus P_i.$$

- *Conjunction is interpreted as set intersection:*

$$[\![\alpha \wedge \beta]\!]^{\mathcal{M}}_{\mathcal{V}} \triangleq [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} \cap [\![\beta]\!]^{\mathcal{M}}_{\mathcal{V}}.$$

- *Disjunction is interpreted as set union:*

$$[\![\alpha \vee \beta]\!]^{\mathcal{M}}_{\mathcal{V}} \triangleq [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} \cup [\![\beta]\!]^{\mathcal{M}}_{\mathcal{V}}.$$

- *The meaning of* $\langle a\rangle\alpha$ *is:*

$$[\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} \triangleq \big\{s \in S \mid \exists_{s'\in S} \big(R_a(s,s') \wedge s' \in [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}}\big)\big\};$$

- *The meaning of* $[a]\alpha$ *is:*

$$[\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} \triangleq \big\{s \in S \mid \forall_{s'\in S} \big(R_a(s,s') \Rightarrow s' \in [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}}\big)\big\};$$

- *The* μ *and* ν *constructs are interpreted as fixpoints of operators on sets of formulas. A formula* $\alpha(X)$ *with free variable* X *can be seen as an operator on sets of states, mapping a set* S' *to the semantics of* α *when* X *is interpreted as* S'*, namely,*

$$S' \mapsto [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]}.$$

  By definition this operator is monotonic, thus it has well defined least and greatest fixpoints by Knaster-Tarski's fixed point theorem [111].

  Then, the meaning of μ *and* ν *is defined (respectively) by:*

$$[\![\mu X.\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} \triangleq \bigcap \Big\{S' \subseteq S \mid [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]} \subseteq S'\Big\},$$
$$[\![\nu X.\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} \triangleq \bigcup \Big\{S' \subseteq S \mid S' \subseteq [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}[S'/X]}\Big\}.$$

Example 1.4. *The formula $\nu Y.\mu X.(p \wedge \langle a\rangle Y) \vee \langle a\rangle X$ means:*

"p holds infinitely often on some a-path".

Concerning the negation operator, as mentioned, it can be expressed within L_μ in a meaningful way, by induction on the structure of the negated formula [13]:

Definition 1.19 (Negation operator for the modal μ-calculus [13])**.**

$$\begin{aligned} \neg(\neg p) &\triangleq p & \neg(\neg X) &\triangleq X \\ \neg(\alpha \vee \beta) &\triangleq \neg\alpha \wedge \neg\beta & \neg(\alpha \wedge \beta) &\triangleq \neg\alpha \vee \neg\beta \\ \neg\langle a\rangle\alpha &\triangleq [a]\neg\alpha & \neg[a]\alpha &\triangleq \langle a\rangle\neg\alpha \\ \neg\mu X.\alpha(X) &\triangleq \nu X.\neg\alpha(\neg X) & \neg\nu X.\alpha(X) &\triangleq \mu X.\neg\alpha(\neg X) \end{aligned}$$

When applying this translation to a formula without free variables (i.e., to a *sentence*), the final result has all variables occurring un-negated, because of the two negations introduced when negating fixpoint expressions.

Thus, the following holds.

Theorem 1.4 ([13])**.** *For every sentence α of L_μ, every transition system $\mathcal{M}$ over the set of states S, and every valuation $\mathcal{V}$:*

$$[\![\neg\alpha]\!]^{\mathcal{M}}_{\mathcal{V}} = S \setminus [\![\alpha]\!]^{\mathcal{M}}_{\mathcal{V}}.$$

We should mention that in 1986 *Niwiński* [92] introduced a hierarchy of fixpoint terms based on the number of alternations between least and greatest fixpoint operators. Also, *Emerson* and *Lei* [54] defined a similar notion of *alternation depth* of a formula. While most useful properties can be expressed with few fixpoints, it is the nesting of the two types of fixpoints that is the source of both expressive power and algorithmic difficulties; indeed, the modal μ-calculus can encode most of the other logics used in verification and still the algorithmics is not harder than the others [13].

Let α be a well-named formula, thus for every bound variable Y we have a unique subformula $\sigma Y.\beta_Y$ in α; let's say that Y is a μ-variable or a ν-variable depending on the binder σ. Then, alternation depth can be defined as follows.

Definition 1.20 (Alternation Depth for modal μ-calculus [13, 54, 92])**.** *The* dependency order *on bound variables of α is the smallest partial order such that $X \leq_\alpha Y$ if X occurs free in $\sigma Y.\beta_Y$. The* alternation depth *of a μ-variable X in a formula α is the maximal length of a chain $X_1 \leq_\alpha \cdots \leq_\alpha X_n$, $X = X_1$, where variables $X_1, X_3, \ldots, X_{2i+1}, \ldots$ are μ-variables and variables $X_2, X_4, \ldots, X_{2i}, \ldots$ are ν-variables, for $i \geq 1$; and alternation depth of ν-variables is defined by interchanging the role of ν and μ. The* alternation depth *of a formula α, denoted `adepth`(α), is the maximum of alternation depths of variables bound in α, or* zero *if there are no fixpoints.*

Proposition 1.1 ([13,54,92])**.** *Fix some $\sigma \in \{\mu, \nu\}$. A formula $\sigma X.\beta(X)$ has the same alternation depth as its unfolding $\beta(\sigma X.\beta(X))$.*

The alternation depth will allow us to illustrate the game semantics of the modal μ-calculus in terms of parity game conditions. These are introduced in the remainder of this section.

Let us mention the complexity of the satisfiability problem for the modal μ-calculus. Suppose that we want to decide if two given formulas of the modal μ-calculus are equivalent, i.e., whether the two formulas are satisfied in the same set of models. Formula equivalence is nothing else than the satisfiability problem: deciding if there exists a model and a state where the formula is true. The following complexity result holds.

Theorem 1.5 ([51,109])**.** *The satisfiability problem for the modal μ-calculus is* ExpTime-*complete.*

Instead, the model-checking problem of the modal μ-calculus turns out to lie within $\mathsf{NP} \cap \text{co-}\mathsf{NP}$, being it equivalent to determining the winner in *parity games*; which in turn belong to the family of ω-regular games. In the following we shall recall these games and some of their basic properties.

1.4.2 ω-Regular Games

An ω-regular game $(\Gamma, \texttt{Win})$ is made of an arena Γ plus a winning condition $\texttt{Win}$. An *arena* is a tuple $\Gamma = (V, E, \langle V_0, V_1 \rangle)$ where $G^\Gamma \triangleq (V, E)$ is a finite directed graph and (V_0, V_1) is a partition of V into the set V_0 of vertices owned by Player 0 (a.k.a. Player $\Box$), and the set V_1 of vertices owned by Player 1 (a.k.a. Player $\bigcirc$). A *weighted arena* is a tuple $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ where: $(V, E, \langle V_0, V_1 \rangle)$ is an arena, and $G^\Gamma \triangleq (V, E, w)$ is a finite weighted directed graph. W.l.o.g. it can be assumed that G^Γ has no sink, i.e., $N^{\text{out}}(v) = \texttt{post}(v) \neq \emptyset$ for every $v \in V$. Still, we remark that G^Γ is not required to be a bipartite graph on colour classes V_0 and V_1. Fig. 1.3 depicts a simple example of an arena Γ.

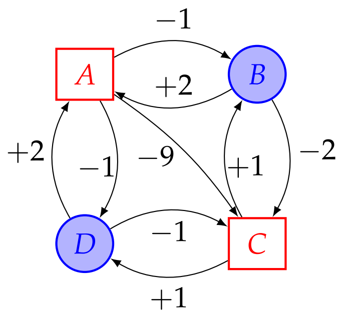


Figure 1.3: An arena Γ.

In this context a game on Γ is played for infinitely many rounds by two players moving a pebble along the arcs of G^Γ. At the beginning of the game we find the pebble on some vertex $v_s \in V$, which is called the *starting position* of the game. At each turn, assuming the pebble is currently on a vertex $v \in V_i$ (for $i = 0,1$), Player i chooses an arc $(v,v') \in E$ and then the next turn starts with the pebble on v'. Let V^+ and V^ω be the set of all finite and infinite sequences on alphabet V, respectively. Given $\pi \in V^\omega$, let $\mathrm{Inf}(\pi)$ be the set of all and only those vertices $v \in V$ that appear infinitely often in π; namely,

$$\mathrm{Inf}(\pi) \triangleq \big\{v \in V \mid \forall_{j\in\mathbb{N}}\, \exists_{k\in\mathbb{N}} \text{ such that } k > j \text{ and } v_k = v\big\}.$$

Generally, a *winning condition* is any $\texttt{Win} \subseteq V^\omega$. The pair $(\Gamma, \texttt{Win}, v_s)$ is called a *game*, where it is given initial starting position $v_s \in V$. A *play* is any infinite path π in Γ:

$$\pi \triangleq v_0 v_1 \cdots v_n \cdots \in V^\omega.$$

The *alphabet* $\Xi(\pi)$ of a play π is the set of all vertices $v \in V$ appearing in π at least once. Player 0 is declared the *winner* of the play π iff $\pi \in \texttt{Win}$. Generally, it is interesting to considering those winning conditions that reflect the acceptance conditions in ω-automata [61]. We will not describe here the theory of ω-automata, for which we refer the reader to the excellent reference text [61], but we just recall the main – so called, *ω-regular* – winning conditions which can be defined for infinite games on graphs:

- *Müller condition:* given a family of subsets $\mathcal{F} \subseteq 2^V$, then:

$$\texttt{Win}_{\mathcal{F}} \triangleq \{\pi \in V^\omega \mid \mathrm{Inf}(\pi) \in \mathcal{F}\};$$

- *Rabin condition:* given a family $\mathcal{F} \triangleq \{(E_0,F_0),(E_1,F_1),\ldots,(E_{m-1},F_{m-1})\}$, where $E_i, F_i \subseteq V$ for every i, then:

$$\texttt{Win}_{\mathcal{F}} \triangleq \{\pi \in V^\omega \mid \exists_k\, \mathrm{Inf}(\pi) \cap E_k = \emptyset \wedge \mathrm{Inf}(\pi) \cap F_k \neq \emptyset\};$$

- *Street condition:* given a family $\mathcal{F} \triangleq \{(E_0,F_0),(E_1,F_1),\ldots,(E_{m-1},F_{m-1})\}$, where $E_i, F_i \subseteq V$ for every i, then:

$$\texttt{Win}_{\mathcal{F}} \triangleq \{\pi \in V^\omega \mid \forall_k\, \mathrm{Inf}(\pi) \cap E_k \neq \emptyset \vee \mathrm{Inf}(\pi) \cap F_k = \emptyset\};$$

- *Rabin chain condition:* given a family $\mathcal{F} \triangleq \{(E_0,F_0),(E_1,F_1),\ldots,(E_{m-1},F_{m-1})\}$, where $E_i, F_i \subseteq V$ for every i, and $E_0 \subsetneq F_0 \subsetneq E_1 \subsetneq F_1 \subsetneq \cdots \subsetneq E_{m-1} \subsetneq F_{m-1}$, then: it goes like the Rabin condition;

- *Parity conditions:* given a coloring function $c : V \to \mathbb{N}$ on the vertices, then:

$$\text{min-parity condition: } \texttt{Win}_c \triangleq \{\pi \in V^\omega \mid \min(\mathrm{Inf}(\pi)) \text{ is even}\};$$

$$\text{max-parity condition: } \texttt{Win}_c \triangleq \{\pi \in V^\omega \mid \max(\mathrm{Inf}(\pi)) \text{ is even}\};$$

- *Büchi condition:* given $\mathcal{F} \subseteq V$, then:

$$\mathtt{Win}_{\mathcal{F}} \triangleq \{\pi \in V^{\omega} \mid \mathrm{Inf}(\pi) \cap \mathcal{F} \neq \emptyset\};$$

As argued in [61], the winning conditions can be transformed into one another, the transformations being exponential in the size of the transformed arenas. Under this prospect, the main result says that it is enough to consider the *parity games* (see Chapter 2, Subsection 2.4.2, Theorem 2.7 in [61]).

Concerning automata theory, let us just mention that the *Müller* condition plays a special role also in the theory of ω-automata, which are (roughly speaking) finite-state automata taking ω-words as input (see [61] for more details). The usual definitions of deterministic and nondeterministic automata are adapted to the case of ω-input-words by introducing new acceptance – so-called, *ω-regular acceptance* – conditions. For this purpose one introduces an *acceptance component* in the specification of the automaton, which may arise in different formats. The acceptance component can be given as a set of states, as a family of sets of states, or as a function from the set of states to a finite set of natural numbers. Indeed, for every ω-regular winning condition as defined above there is a corresponding ω-regular *acceptance condition* which leads to specific families of acceptance components. Then one may consider Büchi, Müller, Rabin, Street and Parity ω-automata. A remarkable result of *McNaughton* and *Müller* asserts:

Theorem 1.6 (Determinization of nondeterministic Büchi ω-automata [61,84])**.** *Every nondeterministic Büchi automaton with n states can be transformed into an equivalent (i.e., one accepting the same ω-language) deterministic Müller automaton with $2^{O(n \log n)}$ states.*

Let us proceed by discussing the determinacy properties of ω-regular games. To this end, we recall next the notions of *forgetful* and *memoryless* strategy.

Definition 1.21. *For any $i \in \{0,1\}$, a* strategy *of Player i is any function,*

$$\sigma_i : V^* \times V_i \to V,$$

such that for every finite path $p'v$ in G^{Γ}, where $p' \in V^*$ and $v \in V_i$, it holds that $(v, \sigma_i(p', v)) \in E$. A play $v_0 v_1 \dots v_n \dots$ is consistent *with a strategy $\sigma \in \Sigma_i$ if $v_{j+1} = \sigma(v_0 v_1 \dots v_j)$ whenever $v_j \in V_i$. A strategy $\sigma_i \in \Sigma_i$ is said to be* finite memory *(or* forgetful*) if there exists a finite set M, an element $m_I \in M$, and two functions,*

$$\delta : V \times M \to M \text{ and } g : V \times M \to V,$$

such that the following holds. When $p = v_0 v_1 \cdots v_{l-1}$ is a prefix of a play which is consistent with σ_i and the sequence $m_0, m_1, \dots, m_l$ is determined by $m_0 \triangleq m_I$ and $m_{i+1} \triangleq \delta(v_i, m_i)$, then it holds that:

$$\sigma_i(v_0 v_1 \cdots v_{l-1}, v_l) = g(v_l, m_l).$$

A strategy σ_i of Player i is positional *(or* memoryless*) if it doesn't need any memory at all; namely, if it is forgetful for some singleton M, one such that $|M| = 1$. The set of all the positional (memoryless) strategies of Player i is denoted by Σ_i^M.*

Definition 1.22. *Given a starting position $v_s \in V$, the* outcome *of strategies $\sigma_0 \in \Sigma_0$ and $\sigma_1 \in \Sigma_1$, denoted $\texttt{outcome}^{\Gamma}(v_s, \sigma_0, \sigma_1)$, is the unique play that starts at v_s and is consistent with both strategies $\sigma_0 \in \Sigma_0$ and $\sigma_1 \in \Sigma_1$.*

Definition 1.23. *Given a memoryless strategy $\sigma_i \in \Sigma_i^M$ of Player i in Γ, then $G^{\Gamma}_{\sigma_i} = (V, E_{\sigma_i}, w)$ is the graph obtained from G^{Γ} by removing all arcs $(v, v') \in E$ such that $v \in V_i$ and $v' \neq \sigma_i(v)$; we say that $G^{\Gamma}_{\sigma_i}$ is obtained from G^{Γ}* by projection *w.r.t. σ_i.*

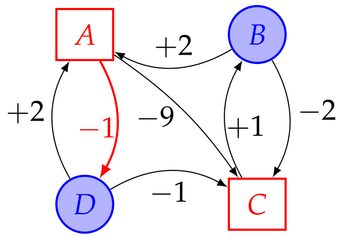


Figure 1.4: An arena obtained by projection.

There are several questions to ask when one is confronted with a ω-regular game as introduced above.

- One may ask whether the game is *determined*, i.e., whether or not one of the players can move so that, regardless of how the other moves, the outcome play will always be winning for him. In that case, one can consider *winning* strategies and *winning regions* $\mathcal{W}_0$ and $\mathcal{W}_1$; where $\mathcal{W}_i$ is the subset of vertices $v \in V$ such that the Player i wins the game that starts at $v_s = v$;

- One may ask whether it is possible to effectively (and maybe efficiently) compute which of the two players wins the game by starting from a given position $v_s \in V$;

- It is not only interesting to know who wins a game, but also how a winning strategy looks like, i.e., one may ask to automatically synthesize a winning strategy for the winning player.

It can be proved that, in every ω-regular game, both players win forgetful. This is called *"forgetful (or finite memory) determinacy"* of ω-regular games (for a full proof see Chapter 2, Subsection 2.4.2, Corollary 2.15 in [61]). It is also worth mentioning that in every Rabin game, Player 0 has a memoryless winning strategy on his winning region. Symmetrically, in every Streett game, Player 1 has a memoryless strategy on his winning region (this is Theorem 2.16 in Chapter 2, Subsection 2.4.2, in [61]).

But, in parity games, *both* players have a memoryless winning strategy on their winning regions (see Chapter 6 in [61]). The following theorem holds.

Theorem 1.7 (Determinacy of Parity Games [50, 122]). *Parity Games, as well as all of the other ω-regular games, lie at the third level of the Borel Hierarchy, i.e., $\Delta_3^0 = \Sigma_3^0 \cap \Pi_3^0$. Hence, they are all determined by the Borel Determinacy Theorem.*

Particularly, Parity Games *are memoryless determined.*

Proofs of memoryless determinacy of parity games can be found e.g., in [50, 122]. The determinacy and the memorylessness of parity games is exploited in various areas. The word and emptiness problem for alternating tree automata as well as model-checking in modal μ-calculus can be reduced to deciding the winner of a parity game. In fact, model checking μ-calculus is equivalent via linear time reduction to determining parity games (for a proof see [50, 53, 61, 118]). Also, parity games offers an elegant tool to simplify Rabin's proof of the decidability of the monadic second-order theory of the binary infinite tree (see e.g., [16, 50, 99]). In summary, the following result holds.

Theorem 1.8 ([50, 53, 61, 118]). *The model-checking problem of modal μ-calculus is linear-time equivalent to the problem of deciding if Player 0 has a winning strategy from a given starting position in a given parity game; the game constructed from a transition system of size m and a formula of size n has size $O(mn)$. Conversely, from a given parity game one can construct an equivalent transition system and a formula; the transition system is of the same size as the parity game.*

1.4.3 From model-checking of L_μ to parity games

Let us provide a sketch of the construction from model-checking of modal μ-calculus to determination of winning regions in parity games. Given a sentence α of the modal μ-calculus, and a state $s \in S$ of a given transition system $\mathcal{M} = \big(S, \{R_a\}_{a\in\mathcal{A}}, \{P_i\}_{i\in\mathbb{N}}\big)$, the model-checking problem asks to decide whether α holds in s, i.e., $\mathcal{M}, s \models \alpha$. We aim at constructing a parity game $\mathcal{G}(\mathcal{M}, \alpha)$ in which Player 0 admits a winning strategy from a starting position corresponding to s iff $\mathcal{M}, s \models \alpha$. Actually, $\mathcal{G}(\mathcal{M}, \alpha)$ is needed also for formulas α with free variables, so valuations $\mathcal{V}$ are also taken into account. So we are going to define a parity game $\mathcal{G}_{\mathcal{V}}(\mathcal{M}, \alpha)$, where $\mathcal{G}_{\mathcal{V}}(\mathcal{M}, \alpha) = \mathcal{G}(\mathcal{M}, \alpha)$ when α is a sentence. To outline this construction, given a formula α, let us consider the *closure* $\mathtt{cl}(\alpha)$ of α, i.e., the smallest set containing α and closed under subformulas and unfolding.

The game is defined as follows:

$$\mathcal{G}_{\mathcal{V}}(\mathcal{M}, \alpha) \triangleq \Big(V_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}, A_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}, p_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}, \langle V0_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}, V1_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}\rangle\Big),$$

where the vertex set is:

$$V_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)} \triangleq \Big\{(s, \beta) \mid s \text{ is a state of } \mathcal{M} \text{ and } \beta \in \mathtt{cl}(\alpha)\Big\},$$

moreover, concerning $V_{0\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}$ and $V_{1\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}$:

$$\{(s,p) \mid s \in S \setminus P\} \cup \{(s,\neg p) \mid s \in S \cap P\} \subseteq V_{0\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}$$
$$\{(s,p) \mid s \in S \cap P\} \cup \{(s,\neg p) \mid s \in S \setminus P\} \subseteq V_{1\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}$$

where the intended interpretation is that $(s,p) \in S \times \mathcal{P}$ will be declared winning for Player 0 (i.e., it will be a sink for Player 1) iff $\mathcal{M},s \models p$, i.e., $s \in P$; otherwise, it will be winning for Player 1 and thus a sink vertex for Player 0. Symmetrically for $(s,\neg p)$. Similarly, concerning variables, one prescribes that:

$$\{(s,X) \mid s \in S \setminus \mathcal{V}(X)\} \subseteq V_{0\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)},$$
$$\{(s,X) \mid s \in S \cap \mathcal{V}(X)\} \subseteq V_{1\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)};$$

so that $(s,X) \in S \times \mathtt{Var}$ will be declared winning for Player 0 (i.e., a sink for Player 1) iff $\mathcal{M},s \models X$, i.e., $s \in \mathcal{V}(X)$; otherwise, it will be winning for Player 1 and thus a sink vertex for Player 0.

Also, for every $s \in S$ and formulas α,β:

$$(s,\alpha \vee \beta) \in V_{0\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)},$$
$$(s,\alpha \wedge \beta) \in V_{1\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}.$$

Finally, for every $s \in S$, for every $a \in \mathcal{A}$ and formula β:

$$(s,\langle a \rangle \beta) \in V_{0\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)},$$
$$(s,[a]\beta) \in V_{1\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}.$$

Concerning positions such as $(s,\sigma X.\beta(X))$, since they will have exactly one outgoing arc in $A_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}$ (see below), they can be controlled by anyone of the two players. The arc set $A_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}$ is defined by induction on the structure of α:

- every position $(s,p),(s,\neg p)$ for $s \in S$ and $p \in \mathcal{P}$ is a sink, i.e., $N^{\text{out}}(s,p) = N^{\text{out}}(s,\neg p) = \emptyset$ for every $s \in S$ and $p \in \mathcal{P}$;
- similarly, every position (s,X) for $s \in S$ and $X \in \mathtt{Var}$ is a sink;
- from positions $(s,\alpha \wedge \beta),(s,\alpha \vee \beta)$ there's one arc to (s,α) and one to (s,β);
- from positions $(s,\langle a \rangle \beta),(s,[a]\beta)$ there's one to (t,β) whenever $(s,t) \in R_a$;
- fix $\sigma \in \{\mu,\nu\}$, from position $(s,\sigma X.\beta(X))$ there's one to $(s,\beta(\sigma X.\beta(X)))$.

The priorities, $p_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)} : V_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)} \to \mathbb{N}$, are assigned as follows. Here the interesting formulas are the fixpoint formulas $\sigma X.\beta(X)$, the μ formulas will have odd priority and the ν formulas will have it even. All of the others formulas will have zero priority. The priorities of the fixpoint formulas can be assigned by relying on the alternation depth.

For any $s \in S$ and any subformula β of α:

$$p_{\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)}(s,\beta) \triangleq \begin{cases} 2 \cdot \lfloor \frac{\mathtt{adepth}(X)}{2} \rfloor & \text{if } \beta \text{ is of the form } \nu X.\gamma(X); \\ 2 \cdot \lfloor \frac{\mathtt{adepth}(X)}{2} \rfloor + 1 & \text{if } \beta \text{ is of the form } \mu X.\gamma(X); \\ 0 & \text{otherwise.} \end{cases}$$

Notice that the alternation depth of X is considered, not that of β, as this allows one to assert a monotonicity property that is crucial for proving correctness of the construction (see [13] for the proof). To determine the winner of the game, the *max*-parity condition is adopted where the highest priority wins.

This concludes the description of $\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha)$. When α is a sentence, it is fine to denote $\mathcal{G}_{\mathcal{V}}(\mathcal{M},\alpha) = \mathcal{G}(\mathcal{M},\alpha)$. At this point, the following holds.

Theorem 1.9 ([13,50]). *For every sentence α of the modal μ-calculus, for every transition system $\mathcal{M} = \big(S,\{R_a\}_{a\in\mathcal{A}},\{P_i\}_{i\in\mathbb{N}}\big)$, and for every state $s \in S$:*

$$\mathcal{M},s \models \alpha \iff \textit{Player 0 has a winning strategy starting from } (s,\alpha) \textit{ in } \mathcal{G}(\mathcal{M},\alpha).$$

The problem of deciding the winner of a parity game belongs to the complexity classes $\mathsf{NP} \cap \text{co-}\mathsf{NP}$. *Marcin Jurdzínski* [70] proved a tighter $\mathsf{UP} \cap \text{co-}\mathsf{UP}$ complexity bound and developed more efficient algorithms.

Definition 1.24. UP *is the complexity class of decision problems solvable in polynomial time on a* unambiguous *non-deterministic Turing Machine; namely, one in which there is at most one accepting path for each input. Moreover, co-*UP *is the complexity class of decision problems whose complement lies in* UP.

In summary, the following result holds.

Theorem 1.10 (Complexity of model-checking [52,54,70]). *The model-checking problem of the modal μ-calculus lies in* $\mathsf{NP} \cap$ *co-*NP*; particularly, it lies in* $\mathsf{UP} \cap$ *co-*UP.

1.4.4 Mean-payoff games

In turn, the problem of determining parity games turns out to be reducible in polynomial-time to that of determining another family of infinite games on graphs, which is now recalled.

Definition 1.25 (Mean Payoff Games). *A* Mean Payoff Game *(MPG) [14,49, 123] is a game played on some arena Γ for infinitely many rounds by two opponents, Player* 0 *gains a payoff defined as the long-run average weight of the play, whereas Player* 1 *loses that value. Formally, the Player* 0*'s* payoff *of a play $v_0v_1\ldots v_n\ldots$ in Γ is defined as follows:*

$$\mathit{MP}_0(v_0v_1\ldots v_n\ldots) \triangleq \liminf_{n\to\infty} \frac{1}{n}\sum_{i=0}^{n-1} w(v_i,v_{i+1}).$$

The value secured *by a strategy $\sigma_0 \in \Sigma_0$ in a vertex $v \in V$ is defined as:*

$$\mathtt{val}^{\sigma_0}(v) \triangleq \inf_{\sigma_1\in\Sigma_1} \mathit{MP}_0\big(\mathit{outcome}^{\Gamma}(v,\sigma_0,\sigma_1)\big),$$

Notice that payoffs and secured values can be defined symmetrically for the Player 1 *(i.e., by interchanging the symbol* 0 *with* 1 *and* inf *with* sup*).*

Ehrenfeucht and *Mycielski* [49] proved that each vertex $v \in V$ admits a unique *value*, denoted $\texttt{val}^{\Gamma}(v)$, which each player can secure by means of a *memoryless* (or *positional*) strategy. Moreover, *uniform* positional optimal strategies do exist for both players, in the sense that for each player there exist at least one positional strategy which can be used to secure all the optimal values, independently with respect to the starting position v_s. Thus, for every MPG Γ:

$$\exists_{\sigma_0 \in \Sigma_0^M} \forall_{v \in V} \Big(\texttt{val}^{\sigma_0}(v) \geq \texttt{val}^{\Gamma}(v) \Big),$$

and,

$$\exists_{\sigma_1 \in \Sigma_1^M} \forall_{v \in V} \Big(\texttt{val}^{\sigma_1}(v) \leq \texttt{val}^{\Gamma}(v) \Big).$$

Indeed, the *(optimal) value* of a vertex $v \in V$ in the MPG Γ is given [49, 123] by:

$$\texttt{val}^{\Gamma}(v) = \sup_{\sigma_0 \in \Sigma_0} \texttt{val}^{\sigma_0}(v) = \inf_{\sigma_1 \in \Sigma_1} \texttt{val}^{\sigma_1}(v).$$

Definition 1.26 (Optimal and Winning Strategies in MPGs)**.** *A strategy $\sigma_0 \in \Sigma_0$ is* optimal *iff $\texttt{val}^{\sigma_0}(v) = \texttt{val}^{\Gamma}(v)$ for all $v \in V$. A strategy $\sigma_0 \in \Sigma_0$ is said to be* winning *for Player* 0 *iff $\texttt{val}^{\sigma_0}(v) \geq 0$, and $\sigma_1 \in \Sigma_1$ is winning for Player* 1 *iff $\texttt{val}^{\sigma_1}(v) < 0$. Correspondingly, a vertex $v \in V$ is a* winning starting position *for Player* 0 *iff $\texttt{val}^{\Gamma}(v) \geq 0$; otherwise it is winning for Player* 1*.*

1.4.5 From parity games to mean-payoff games

We are now in the position to recall the reduction from parity games to MPGs.

Theorem 1.11 (Reduction from Parity Games to Mean Payoff Games [70])**.** *The problem of deciding the winner in a Parity Game reduces in polynomial-time to the problem of deciding the winner in a Mean-Payoff Game.*

Basically, in such reduction, given a parity game $\Gamma = (V, A, p, \langle V_0, V_1 \rangle)$ with coloring/priority function $p : V \to \mathbb{N}$, one constructs an MPG with the same graph as Γ and where all arcs are weighted with the following weight function (see [70]):

$$w(a) \triangleq (-|V|)^{p(u)}, \quad \text{for every arc } a = (u, v) \in A.$$

We shall study the problem of computing optimal values and optimal positional strategies in MPGs in Chapters 6 and 7, where it is offered an improved pseudo-polynomial upper bound on the computational complexity of that problems. Moreover, MPGs turn out to be the fundamental model underpinning the results offered from Chapters 2 to 4 concerning automated temporal planning and dynamic consistency checking of conditional temporal networks.

Part I

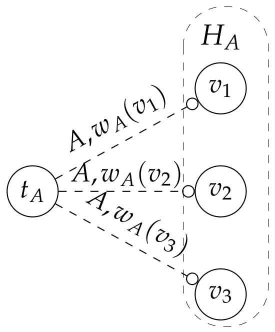
Temporal Constraint Networks

2 Hyper Temporal Networks

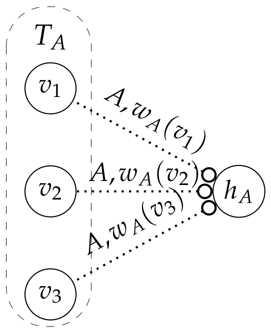
Chapter Abstract

Simple Temporal Networks (STNs) provide a powerful and general tool for representing conjunctions of maximum delay constraints over ordered pairs of temporal variables. In this chapter we introduce Hyper Temporal Networks (HyTNs), a strict generalization of STNs, to overcome the limitation of considering only conjunctions of constraints but maintaining a practical efficiency in the consistency check of the instances. In a Hyper Temporal Network a single temporal hyperarc constraint may be defined as a set of two or more maximum delay constraints which is satisfied when at least one of these delay constraints is satisfied. HyTNs are meant as a light generalization of STNs offering an interesting compromise. On one side, there exist practical pseudo-polynomial time algorithms for checking consistency and computing feasible schedules for HyTNs. On the other side, HyTNs offer a more powerful model accommodating natural constraints that cannot be expressed by STNs like *"Trigger off exactly δ min before (after) the occurrence of the first (last) event in a set."*, which are used to represent synchronization events in some process aware information systems/workflow models proposed in the literature.

This chapter is a revised version of [33].



(a) Multi-Head Hyperarc $A = (t_A, H_A, w_A)$.



(b) Multi-Tail Hyperarc $A = (T_A, h_A, w_A)$.

A graphical representation of the two kinds of hyperarcs.

2.1 Introduction

In many areas of Artificial Intelligence (AI), including planning, scheduling and workflow management systems, the representation and management of quantitative temporal aspects is of crucial importance [7,25,26,48,95,107]. Examples of possible quantitative temporal aspects are: constraints on the earliest start time and latest end time of activities, constraints over the minimum and maximum temporal distance between activities, etc.

In many cases these constraints can be represented as an instance of a *Simple Temporal Network (STN)* [44], a directed weighted graph where each node represents a time-point variable (timepoint), usually corresponding to the beginning or the end of an activity, and each arc specifies a binary constraint on the scheduling times to be assigned to its endpoints. In [44], each arc is labeled with a closed interval of real values: for example, the labeled arc $u \xrightarrow{[x,y]} v$ encodes the binary constraint $x \leq v - u \leq y$ over its endpoints u and v. A more uniform and elementary representation of an STN is provided by its *distance graph*[1] [44], a graph having the same set of nodes as the original one, but where each arc $u \xrightarrow{[x,y]} v$ is replaced by two arcs, each labeled with a single real value: arc $u \xrightarrow{y} v$ to express the constraint $v - u \leq y$, and arc $v \xrightarrow{-x} u$ to express the constraint $u - v \leq -x$, i.e., $x \leq v - u$.

An STN is said to be *consistent* if it is possible to assign a real value to each timepoint so that all temporal constraints are satisfied. The consistency property can be verified by searching for negative cycles in the distance graph and it is well known that the consistency check and the determination of the earliest/latest value for each timepoint can be done in polynomial time [44].

However, STNs do not allow the expression of constraints like *"trigger off an event exactly δ min after the occurrence of the last of its predecessors"*, which are a quite natural constraints to represent synchronization events in a process aware information system plan/workflow schema [65]. This is because in STNs, and in some of their natural extensions, (1) it is not possible to represent a single constraint involving more than two timepoints and (2) all constraints have to be satisfied in order to have the network consistent. On the contrary, the above constraint about a synchronization event can be represented as a set of distance constraints, each involving a different pair of timepoints, that is considered satisfied when at least one of set components is satisfied. In order to represent and analyze disjunctive constraints like the above one, it is then necessary to consider models like *Disjunctive Temporal Problem* (DTP) [108] where a constraint is a set of disjunctive difference constraints over the timepoints. The drawback of such model is that the consistency check problem is NP-complete [108].

[1]Distance graph is also called *constraint graph* by other authors [42]. Moreover, Bellman [4] was the first to describe the relation between shortest paths and difference constraints in a constraint graph.

2.1.1 Contribution

In this chapter we propose to generalize STN to *Hyper Temporal Network* (HyTN), which allows also the expression of constraints like the above one regarding synchronization events, but where the consistency check is amenable of effective solution algorithms.

Also, we show an interesting link between the consistency check of HyTNs and resolution in Mean Payoff Games (MPG), a family of perfect information infinite pebble games played on finite graphs by two opponents [49], for which some pseudo-polynomial time algorithms for determining winning strategies are known [14, 123].

A preliminary version of this chapter appeared in the proceedings of TIME symposium [32]. Here, as in [33], we extend the presentation as follows: (1) the definition of HyTN has been extended in order to allow the presence of two kinds of hyperarcs; (2) the motivating example section has been revised to show how the new kind of hyperarc can be used; (3) some further issues and pertinent properties about HyTN have been introduced and proved; (4) several proofs have been expanded and clarified; (5) the experimental analysis of the consistency check algorithm has been improved considering more recent algorithms [14] for finding winning strategies for MPGs. This has improved the performances significantly.

2.1.2 Organization

The rest of the chapter is organized as follows. In Section 2.2 we present a motivating example from the domain of the workflow-based process management to bring out HyTNs. Section 6.2 introduces some definitions and well-known results for STNs and introduces some definitions about hypergraphs. The generalization of STNs into HyTNs and the definition of consistency problem for HyTNs are presented in Section 2.4. In Section 2.5 we recall the main facts and results about Mean Payoff Games which are useful for the following sections. Section 2.6 presents the investigation into the link between the HyTN consistency problem and Mean Payoff Games deriving pseudo-polynomial time algorithms for checking the consistency of HyTNs and computing feasible schedules whenever they exist. Some empirical evaluations of the proposed algorithms are reported in Section 2.7. In Section 2.8, some related works are presented and discussed with respect to our approach. Section 2.9 summarizes the main facts brought to light in this chapter and presents a possible future development of the work we are currently carrying on.

2.2 Motivating Examples

In the introduction we have briefly recalled a kind of constraint that cannot be expressed within STNs. In this section, we describe in more detail two examples of temporal constraints that cannot be fully described in an STN in order to introduce and motivate the new expressive capability of our model. As a further motivation, at the end of the section we also spotlight how this new capability has been recently exploited to check the consistency of Conditional Simple Temporal Networks (CSTNs) [113] in a more efficient way.

Let us consider an example in the domain of the workflow-based process management, a domain concerned with the coordination and control of business processes using information technology. A *workflow* is a representation of a business process as the coordinated execution of activities by human or automatic executors (agents). A *Workflow management system* (WfMS) is a software system that supports the automatic execution of workflows [65]. In a WfMS, the management of temporal aspects is a critical component and in the literature there are many proposals on how to extend a workflow in order to represent and manage temporal constraints of a business process [7, 20, 25–28, 47, 48, 60]. In particular, in [7, 20, 28, 47, 48] authors show how to represent and manage some kinds of temporal constraints using specific algorithms, while in [25–27, 60] authors show how it is possible to represent and manage a wider class of temporal constraints exploiting models like Time Petri Nets [86] or STNs/STNUs [89].

In this chapter we consider an excerpt of the conceptual temporal model proposed by Combi *et al.* [27], where the specification of a temporal workflow is given by a *workflow schema*, a directed graph (also called workflow graph) where nodes correspond to activities and arcs represent control flows that define activity dependencies on the order of execution. Both nodes and arcs may be associated to temporal ranges to specify temporal constraints. There are two different types of activity: tasks and connectors. Tasks represent elementary work units that will be executed by external agents. Each task is graphically represented by a box containing a name and a temporal range that specifies the allowed temporal span for its execution. Connectors represent internal activities executed by the WfMS to achieve a correct and coordinated execution of tasks. They are graphically represented by diamonds and, as with tasks, each of them has a temporal range that gives the temporal span allowed to the WfMS for executing it. Every arc has a temporal property that gives the allowed times that can be spent by the WfMS for possibly delaying the consideration of the next activity after the end of the previous one. There are different kinds of connector that allow one to modify a control flow. *Split* connectors are nodes with one incoming arc and two or more outgoing arcs: after the execution of the predecessor, (possibly) several successors have to be considered for the execution. The set of nodes that can start their execution is given by the kind of split connector. A split connector can be: Parallel, Alternative or Conditional. *Join* connectors are nodes with two or more incoming arcs and one outgoing arc only. The types of activities considered in [27] are a subset of the possible activities specified by the Workflow Management Coalition [65, 116].

Fig. 2.1 shows a simple workflow schema where the Parallel connector $+_1$ splits the flow into three parallel flows of execution (one for the sequence of tasks T_1 and T_2, one for task T_3, and one for task T_4 and T_5) that have to be joined (synchronized) by the AND join connector $+_2$ before continuing the execution; all temporal ranges are in minutes.

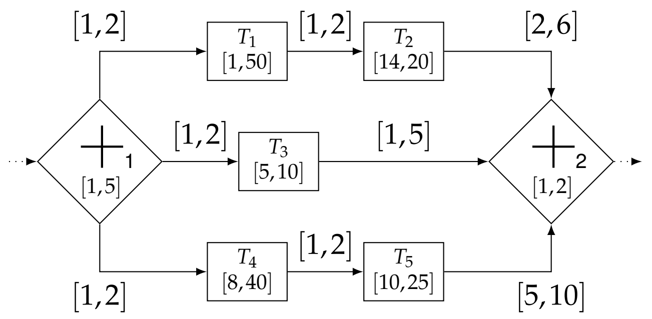


Figure 2.1: A simple workflow schema excerpt with three parallel flows of execution.

Let us consider the connector $+_2$; according to the recommendations from the Workflow Management Coalition (WfMC) [65] and the temporal specification from [27], the execution of this connector requires to wait all incoming flows and, after the last incoming flow, to wait a time according to the connector temporal range before following the outgoing arc. In other words, the incoming flows can arrive at different instants but only when the last one arrives, the connector has to be activated in order to continue with the execution.

Combi *et al.* [27] proposed a method to translate workflow schemata to STNs/STNUs [89] in order to analyze and validate all temporal aspects in a rigorous way. As already noted in [24] and [25], such translation cannot specifically represent the behavior of an AND join connector, because the kind of constraints in an STN/STNU is limited. Therefore, in [25], the authors proposed an adjustment of the translation of an AND join connector introducing for each incoming arc of the connector a *buffer* node connected with some determined new arcs and assuming a reasonable but fixed execution algorithm for the STN. In more detail, let us consider Fig. 2.2 that depicts the representation of workflow of Fig. 2.1 by means of an STN following partially the method described in [25] (without loss of generality, here we convert task constraints as STN arcs instead of STNU contingent ones because we are interested only in the AND join conversion). Each activity of the workflow is represented by two STN nodes, one to represent the begin timepoint, B_i, one for the end one, E_i, and temporal ranges in the workflow are represented by STN arc labels. Regarding the translation of the AND join node $+_2$, nodes representing the task endings on parallel flows, E_{T_2}, E_{T_3}, and E_{T_5}, are connected to *buffer* nodes b_1, b_2, and b_3 that allow the parallel flows to complete their execution following only their temporal constraints. Then, b_1, b_2, and b_3 are connected to node B_{+_2} (which represents the begin instant of the AND join connector) by temporal constraints $[0, t_1], [0, t_2]$ and $[0, t_3]$, where the values t_1, t_2, and t_3 are determined during the workflow-to-STN conversion as explained in [25].

Now, let us consider a possible execution scenario. If b_1, b_2, and b_3 occur all together at instant 20, then, following the proposed temporal semantics [27],

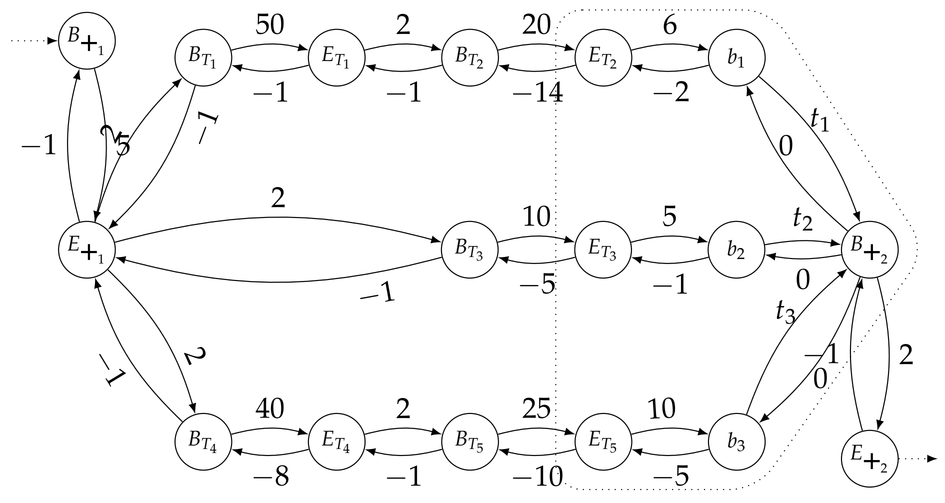


Figure 2.2: An STN representing temporal aspects of the workflow depicted in Fig. 2.1. The dotted region emphasizes, within the workflow excerpt, the connections to an AND join connector.

the only possible instant value for B_{+_2} must be 20 while the updated STN allows any value in the range $[20, 20 + \min\{t_1, t_2, t_3\}]$. In [25], the authors showed that the right value is always the lower bound of such extended range and, therefore, it is sufficient to adopt an early execution strategy in order to choose the right value for timepoint B_{+_2}.

In other words, the proposed translation has two drawbacks: (1) it requires some preliminary computations for determining t_1, t_2, and t_3 values, and (2) the resulting STN admits some solutions that are not admissible by the semantics of the AND join connector.

To specifically represent the behavior of an AND join connector with respect to its predecessor time points without auxiliary conditions or analysis, it is necessary to introduce a new kind of constraint based on hyperarcs, as shown in Fig. 2.3. In the figure the *multi-tail hyperarc* A consists of three dashed arcs—called *components*—and replaces the arcs from b_i $(i = 1, 2, 3)$ to B_{+_2} of Fig. 2.2. We say that a multi-tail hyperarc is satisfied if at least one of its components is satisfied. In Fig. 2.3 dashed arcs define the hyperarc A that is satisfied if B_{+_2} is 0 distant from at least one time point among b_1, b_2, and b_3. Since B_{+_2} is constrained to occur at the same instant or after each time point b_1, b_2, and b_3 by the arcs between B_{+_2} and b_i, $i = 1, 2, 3$, the result is that to satisfy A it is necessary that B_{+_2} occurs at the same instant of the last time point among b_1, b_2, and b_3, as required originally. In more general, a multi-tail hyperarc is defined as a set of distance constraints (components) between some time points and a common end point.

The use of hyperarcs allows also the representation of temporal aspect of other advanced connectors as, for example, the Structured Discriminator [116].

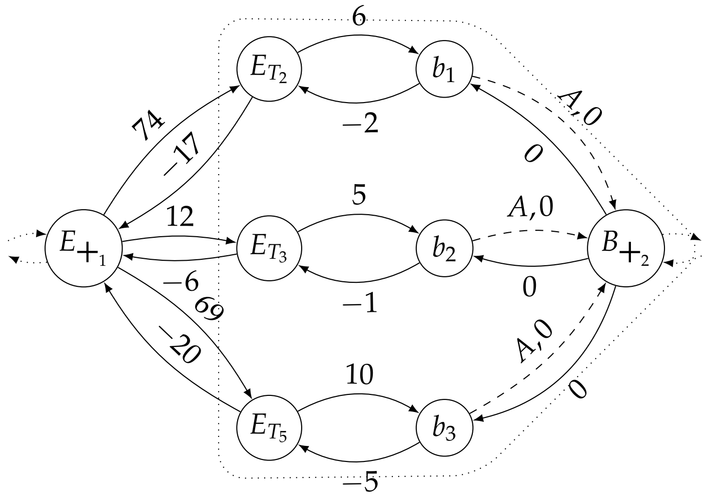
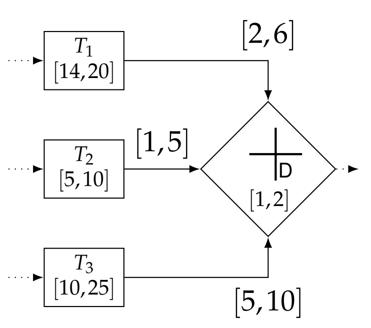


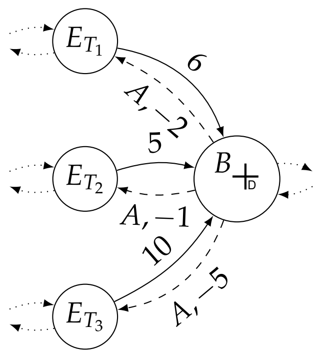
Figure 2.3: An augmented STN, that we call HyTN, where dashed arcs represent components of hyperarcs, a new kind of constraint. This HyTN improves the representation of the STN of Fig. 2.2. To emphasize the changes, here we have summed all arcs outside the dotted region.

The Structured Discriminator connector provides a means of merging two or more distinct flows in a workflow instance into a single subsequent. In particular, it triggers the subsequent flow as soon as the the first incoming flow arrives. The arrival of other incoming flows thereafter have no effect on the subsequent flow. As such, the Structured Discriminator provides a mechanism for progressing the execution of a process once the first of a series of concurrent tasks has completed and according to the connector temporal range. Fig. 2.4a depicts an excerpt of a workflow schema containing a structured discriminator connector, $+_{\mathrm{D}}$, that joins three parallel flows.

At the best of our knowledge, currently there are no proposals for the representation of temporal constraints of a discriminator connector in any temporal workflow model or process-aware information system [77]. Even exploiting the methodology proposed in [25], it is easy to verify that it is not possible to represent such connector as an STN because in a consistent STN all constraints have to be satisfied while here it is necessary to allow the possibility that only one constraint of a set has to be satisfied in order to specifically represent a discriminator connector. A possible way for specifically managing a discriminator connector consists in following the approach suggested by [25] for representing activities and considering a variant of the multi-tail hyperarc, called *multi-head hyperarc*, for representing its temporal constraint, as depicted in Fig. 2.4b. In the figure there is a multi-head hyperarc A that connects the node representing the beginning instant of the discriminator activity to all nodes representing the end instant of the activities that precede the considered discriminator and are directly connected to it. In general a multi-head hyperarc is defined as



(a) An excerpt of a workflow schema containing a structured discriminator connector, $+_{\mathrm{D}}$.



(b) A possible representation of temporal aspects of a discriminator connectors by means of a multi-head hyperarc.

Figure 2.4: A structured discriminator connector and a possible representation of its temporal aspects.

a set of distance constraints (components) between one time point and some end points. We say that a multi-head hyperarc is satisfied if at least one of its components is satisfied. In Fig. 2.4b, dashed arcs define the hyperarc A that is satisfied if $B_{+_{\mathrm{D}}}$ is at least 2 distant from E_{T_1} or 1 distant from E_{T_2} or 5 distant from E_{T_3}. It is sufficient that one of such previous nodes is executed and that the delay represented in the corresponding connecting arc is passed to execute $B_{+_{\mathrm{D}}}$, as required by the structured discriminator connector semantics.

HyTNs are not only suitable for better representing temporal constraints originating from temporal workflow, but also for better representing more general temporal constraint networks like Conditional Simple Temporal Network [113].

A Conditional Simple Temporal Network (CSTN) is an enriched graph for representing and reasoning about temporal constraints in domains where some constraints may apply only in certain condition settings (scenarios). Each *condition* in a CSTN is represented by a propositional letter whose truth value is *observed* in real time as the outcome of the execution of an *observation time-point*. An execution strategy for a CSTN has to determine an execution time for each time-point guaranteeing that all temporal constrains that are significant in the resulting scenario are satisfied. An execution strategy can be dynamic in that its execution decisions can react to the information obtained from such observations. The Conditional Simple Temporal Problem (CSTP) consists in determining whether a given CSTN admits a dynamic execution strategy for any possible combination of propositional outcomes happens to be observed over time. If such a strategy exists, the CSTN is said to be dynamically consistent (DC).

Tsamardinos *et al.* [113] solved the CSTP by first encoding it as a meta-level Disjunctive Temporal Problem (DTP), then feeding it to an off-the-shelf DTP solver. Although of theoretical interest, this approach is not practical because the CSTP-to-DTP encoding has exponential size, and the DTP solver itself runs in exponential time. To our knowledge, this approach has never been empirically evaluated [69].

In [34, 40] and in Chapter 3, we propose a novel representation of CSTNs in terms of HyTNs allowing the determination of the first singly exponential-time algorithm for checking the dynamic consistency of Conditional Simple Temporal Networks. More precisely, a CSTN instance is represented as a suitable HyTN where each possible scenario is represented and connected to other scenarios in an appropriate way and, then, such HyTN instance is solved in pseudo-polynomial time by the algorithms analyzed in the present chapter.

In summary, HyTNs allow the representation of temporal constraints that are more general of those represented in STNs [44], because they allow disjunctions involving more than two time points, but less general than those represented in DTPs [108] because all disjunctions related to a multi-head(tail) hyperarcs have to contain a common variable. Such kind of STN generalization not only allows the compact representation of some common temporal constraints in the domains like the workflow-based process management but also allows the determination of new interesting algorithm for checking dynamic-consistency in richer models like CSTN.

2.3 Background and Notation

The reader is referred to Subsection 1.2.2, Chapter 1, where we introduce some definitions, notations and well-know results about graphs and conservative graphs; moreover, where we recalled the relation between the consistency property of STNs and the conservative property of weighted graphs.

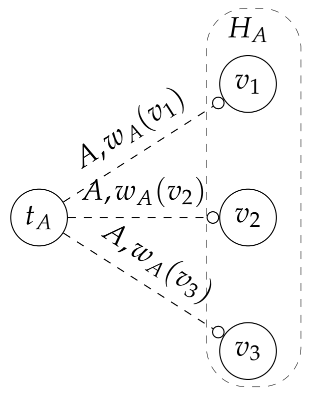
In this chapter, we also deal with directed weighted hypergraphs.

Definition 2.1 (Hypergraph). *A* hypergraph $\mathcal{H}$ *is a pair* $(V, \mathcal{A})$*, where* V *is the set of nodes, and* $\mathcal{A}$ *is the set of* hyperarcs*. Each hyperarc* $A \in \mathcal{A}$ *is either a* multi-head *or a* multi-tail *hyperarc.*

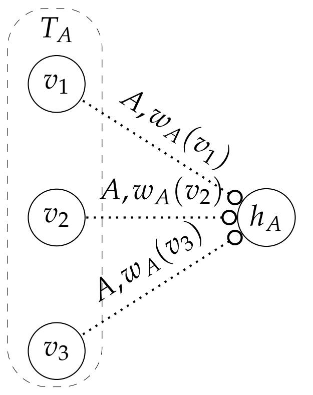
A multi-head *hyperarc* $A = (t_A, H_A, w_A)$ *has a distinguished node* t_A*, called the* tail *of* A*, and a nonempty set* $H_A \subseteq V \setminus \{t_A\}$ *containing the* heads *of* A*; to each head* $v \in H_A$ *is associated a* weight $w_A(v) \in \mathbb{R}$*. Fig. 2.5a depicts a possible representation of a multi-head hyperarc: the tail is connected to each head by a dashed arc labeled by the name of the hyperarc and the weight associated to the considered head.*

A multi-tail hyperarc $A = (T_A, h_A, w_A)$ *has a distinguished node* h_A*, called the* head *of* A*, and a nonempty set* $T_A \subseteq V \setminus \{h_A\}$ *containing the* tails *of* A*; to each tail* $v \in T_A$ *is associated a* weight $w_A(v) \in \mathbb{R}$*. Fig. 2.5b depicts a possible representation of a multi-tail hyperarc: the head is connected to each tail by a dotted arc labeled by the name of the hyperarc and the weight associated to the considered tail.*

The *cardinality* of a hyperarc $A \in \mathcal{A}$ is given by $|A| \triangleq |H_A \cup \{t_A\}|$ if A is



(a) Multi-Head Hyperarc $A = (t_A, H_A, w_A)$.



(b) Multi-Tail Hyperarc $A = (T_A, h_A, w_A)$.

Figure 2.5: A graphical representation of the two kinds of hyperarcs.

multi-head, and $|A| \triangleq |T_A \cup \{h_A\}|$ if A is multi-tail; if $|A| = 2$, then $A = (u, v, w)$ is a standard arc. The *order* and *size* of a hypergraph $(V, \mathcal{A})$ are denoted by $n \triangleq |V|$ and $m \triangleq \sum_{A \in \mathcal{A}} |A|$, respectively.

2.4 HyTN and Consistency Property

We introduce now *Hyper Temporal Networks* (HyTNs), a strict generalization of STNs to partially overcome the limitation of allowing only conjunctions of constraints. Compared to STN distance graphs, which they naturally extend, HyTNs allow a greater flexibility in the definition of temporal constraints.

A HyTN is a directed weighted hypergraph $\mathcal{H} = (V, \mathcal{A})$ where a node represents a time point variable (timepoint), and a multi-head/multi-tail hyperarc represents a set of temporal distance constraints between the tail/head and the heads/tails, respectively.

For example, the multi-tail hyperarc $A = (T_A, B_{+_2}, w_A)$ in Fig. 2.3, where $T_A = \{b_1, b_2, b_3\}$ and $w_A(b_i) = 0$ for $i = 1,2,3$, stands for the set of distance constraints $\{B_{+_2} - b_i \le 0 \mid i = 1,2,3\}$.

In general, we say that a hyperarc is *satisfied* when at least one of its distance constraints is satisfied. Then, we say that a HyTN is *consistent* when it is possible to assign a value to each time-point variable so that all of its hyperarcs are satisfied.

More formally, in the HyTN framework the consistency problem is defined as the following decision problem.

Definition 2.2 (General-HyTN-Consistency). *Given a HyTN $\mathcal{H} = (V, \mathcal{A})$, decide whether there exists a scheduling $s : V \to \mathrm{R}$ such that, for every hyperarc $A \in \mathcal{A}$, the following holds:*

- *if $A = (t,h,w)$ is a standard arc, then*

$$s(h) - s(t) \leq w;$$

- *if $A = (t_A, H_A, w_A)$ is a multi-head hyperarc, then*

$$s(t_A) \geq \min_{v \in H_A} \{s(v) - w_A(v)\};$$

- *if $A = (T_A, h_A, w_A)$ is a multi-tail hyperarc, then*

$$s(h_A) \leq \max_{v \in T_A} \{s(v) + w_A(v)\}.$$

Any such scheduling $s : V \to \mathrm{R}$ is called *feasible*. A HyTN that admits at least one feasible scheduling is called *consistent*.

Comparing the consistency of HyTNs with the consistency of STNs, the most important aspect of novelty is that, while in a distance graph of a STN each arc represents a distance constraint and all such constraints have to be satisfied by a feasible schedule, in a HyTN each hyperarc represents a set of one or more distance constraints and a feasible scheduling has to satisfy at least one such distance constraints for each hyperarc.

Let us show some interesting properties about the consistency problem for HyTNs.

The first interesting property is that any integral-weighted HyTN admits an integral feasible schedule when it is consistent, as proved in the following lemma.

Lemma 2.1. *Let $\mathcal{H} = (V, \mathcal{A})$ be an integral-weighted and consistent HyTN. Then $\mathcal{H}$ admits an integral feasible scheduling:*

$$s : V \to \{-T, -T+1, \dots, T-1, T\},$$

where $T = \sum_{A \in \mathcal{A}, v \in V} |w_A(v)|$.

Proof. Since $\mathcal{H}$ is consistent, then there exists a feasible scheduling $\tilde{s} : V \to \mathrm{R}$. The idea in this proof is to project the HyTN $\mathcal{H}$ over a conservative graph $G_{\mathcal{H}}$, by selecting for each hyperarc $A \in \mathcal{A}$ one standard arc that is feasible according to $\tilde{s}$ (more details below); and then, in that setting, to exploit the integrality properties of potentials as stated in Theorem 1.1. Note that $G_{\mathcal{H}}$ is asked to resolve the non-determinism contained in the disjunctive nature of the hyperarcs (i.e., which choices within the hyperarcs of $\mathcal{H}$, i.e., which standard arcs, should be selected to construct $G_{\mathcal{H}}$?); in order to sort out such non-determinism, the projection is built considering the given feasible scheduling $\tilde{s}$ as follows.

For each hyperarc $A \in \mathcal{A}$, a weighted directed arc e_A is defined as follows:

- if $A = (u,v,w)$ is a standard arc, then $e_A \triangleq (u,v,w)$. Note that $\tilde{s}(v) \leq \tilde{s}(u) + w$ follows by the feasibility of $\tilde{s}$;

- if $A = (t_A, H_A, w_A)$ is a multi-head hyperarc, then

  $$e_A \triangleq (t_A, v_A, w_A(v)) \text{ where } v_A = \arg\min_{v \in H_A} \{\tilde{s}(v) - w_A(v)\}.$$

  Here, $\tilde{s}(v_A) \leq \tilde{s}(t_A) + w_A(v)$ follows by the feasibility of $\tilde{s}$;

- if $A = (T_A, h_A, w_A)$ is a multi-tail hyperarc, then

  $$e_A \triangleq (v_A, h_A, w_A(v)) \text{ where } v_A = \arg\max_{v \in T_A} \{\tilde{s}(v) + w_A(v)\}.$$

  Here, $\tilde{s}(h_A) \leq \tilde{s}(v_A) + w_A(v)$ follows by the feasibility of $\tilde{s}$.

Now, a weighted directed graph $G_{\mathcal{H}} = (V, E)$ with $E \triangleq \{e_A \mid A \in \mathcal{A}\}$ is defined. G is integral-weighted and conservative graph since it admits $\tilde{s}$ as a potential function. Therefore, G admits an integral potential function $s : V \to \{-T, -T+1, \ldots, T-1, T\}$. Indeed, such a function s is obtained by applying the Bellman-Ford algorithm on G. To conclude, we observe that s is also an integral feasible scheduling for $\mathcal{H}$. □

The following theorem states that General-HyTN-Consistency is NP-complete.

Theorem 2.1. General-HyTN-Consistency *is an* NP*-complete problem even if input instances $\mathcal{H} = (V, \mathcal{A})$ are restricted to satisfy $w_A(\cdot) \in \{-1, 0, 1\}$ and $|H_A|, |T_A| \leq 2$ for every $A \in \mathcal{A}$.*

Proof. If $\mathcal{H} = (V, \mathcal{A})$ is integral-weighted and consistent, then it admits an integral feasible scheduling $s : V \to \{-T, \ldots, T\}$ by Lemma 2.1. Moreover, any such feasible scheduling can be verified in polynomial time w.r.t. the size of the input; hence, General-HyTN-Consistency is in NP.

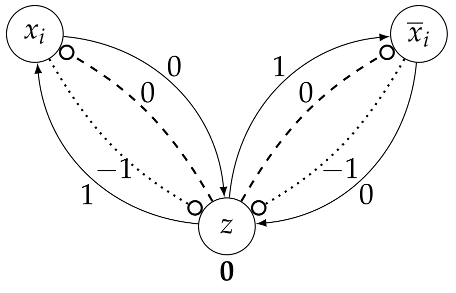
To show that the problem is NP-hard, we describe a reduction from 3-SAT.

Let us consider a boolean 3-CNF formula with $n \geq 1$ variables and $m \geq 1$ clauses:

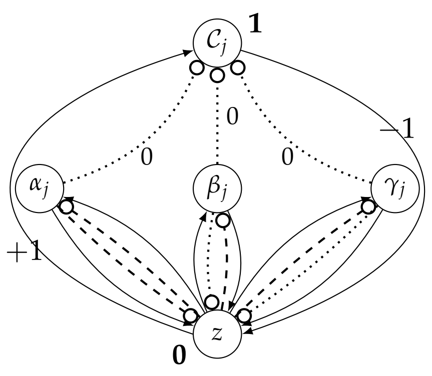
$$\varphi(x_1, \ldots, x_n) = \bigwedge_{i=1}^{m} (\alpha_i \vee \beta_i \vee \gamma_i)$$

where $\mathcal{C}_i = (\alpha_i \vee \beta_i \vee \gamma_i)$ is the i-th clause of φ and each $\alpha_i, \beta_i, \gamma_i \in \{x_j, \overline{x}_j \mid 1 \leq j \leq n\}$ is either a positive or a negative literal.

We associate to φ a HyTN $\mathcal{H}_\varphi = (V, \mathcal{A})$, where each boolean variable x_i occurring in φ is represented by two nodes, x_i and $\overline{x}_i$. V also contains node z that represents the reference initial node for the HyTN $\mathcal{H}_\varphi$, i.e., the first node that has to be executed. For each pair x_i and $\overline{x}_i$, $\mathcal{H}_\varphi$ contains a pair of hyperarc constraints as depicted in Fig. 2.6a: one with multi-head $\{x_i, \overline{x}_i\}$ and tail in z and the other multi-tail $\{x_i, \overline{x}_i\}$ and head in z. If $\mathcal{H}_\varphi$ is consistent, the pair of hyperarcs associated to $x, \neg x$ assures that $\mathcal{H}_\varphi$ admits a feasible scheduling s such that $s(x_i)$ and $s(\overline{x}_i)$ are coherently set with values in $\{0, 1\}$ (see Lemma 2.1). In this way, s is forced to encode a truth assignment on



(a) Gadget for a 3-SAT variable x_i.



(b) Gadget for a 3-SAT clause $\mathcal{C}_j = (\alpha_j \vee \beta_j \vee \gamma_j)$ where each $\alpha_j, \beta_j, \gamma_j$ is a positive or negative literal.

Figure 2.6: Gadgets used in the reduction from 3-SAT to General-HyTN-Consistency.

the x_i's. The HyTN $\mathcal{H}_\varphi$ contains also a node $\mathcal{C}_j$ for each clause $\mathcal{C}_j$ of φ; each node $\mathcal{C}_j$ is connected by a multi-tail hyperarc with head in $\mathcal{C}_j$ and tails over the literals occurring in $\mathcal{C}_j$ and by two standard and opposite arcs with node z as displayed in Fig. 3.6b. Such setting of arcs assures that if $\mathcal{H}_\varphi$ admits a feasible scheduling s, then s assigns value 1 at least to one of the node representing the literals connected with the hyperarc.

More formally, $\mathcal{H}_\varphi = (V, \mathcal{A})$ is defined as follows:

- $V = \{z\} \cup \{x_i \mid 1 \le i \le n\} \cup \{\overline{x}_i \mid 1 \le i \le n\} \cup \{\mathcal{C}_j \mid 1 \le j \le m\}$;
- $\mathcal{A} = \bigcup_{i=1}^{n} \mathrm{Var}_i \cup \bigcup_{j=1}^{m} \mathrm{Cla}_j$, where:
  - $\mathrm{Var}_i = \Big\{(z, x_i, 1), (x_i, z, 0), (z, \overline{x}_i, 1), (\overline{x}_i, z, 0),$
    $(\{x_i, \overline{x}_i\}, z, [w(x_i), w(\overline{x}_i)] = [-1, -1]),$
    $(z, \{x_i, \overline{x}_i\}, [w(x_i), w(\overline{x}_i)] = [0, 0])\Big\}$.
    This defines the variable gadget for x_i as depicted in Fig. 2.6a;
  - $\mathrm{Cla}_j = \Big\{(z, \mathcal{C}_j, 1), (\mathcal{C}_j, z, -1),$
    $(\{\alpha_j, \beta_j, \gamma_j\}, \mathcal{C}_j, [w(\alpha_j), w(\beta_j), w(\gamma_j)] = [0, 0, 0])\Big\}$.
    This defines the clause gadget for clause $\mathcal{C}_j = (\alpha_i \vee \beta_i \vee \gamma_i)$ as depicted in Fig. 3.6b.

Notice that $|V| = 1 + 2n + m = O(m + n)$ and $m_{\mathcal{A}} = 8n + 5m = O(m + n)$; therefore, the transformation is linearly bounded in time and space.

We next show that φ is satisfiable if and only if $\mathcal{H}_\varphi$ is consistent.

Any truth assignment $\nu : \{x_1,\dots,x_n\} \to \{\texttt{true},\texttt{false}\}$ satisfying φ can be translated into a feasible scheduling $s : V \to \mathbb{Z}$ of $\mathcal{H}_\varphi$ as follows. For node z, let $s(z) = 0$, and let $s(\mathcal{C}_j) = 1$ for each $j = 1,\dots,m$; then, for each $i = 1,\dots,n$, let $s(x_i) = 1$ and $s(\overline{x}_i) = 0$ if the truth value of x_i, $\nu(x_i)$, is $\texttt{true}$, otherwise let $s(x_i) = 0$ and $s(\overline{x}_i) = 1$. It is easy to verify that, using this scheduling s, all the constraints comprising each single gadget are satisfied and, therefore, the network is consistent.

Vice versa, assume that $\mathcal{H}_\varphi$ is consistent. Then, it admits an integral feasible scheduling s by Lemma 2.1. After the translation $s(v) = s(v) - s(z)$, we can assume that $s(z) = 0$. Hence, $s(\mathcal{C}_j) = 1$ for each $j = 1,\dots,m$, as enforced by the two standard arcs incident at $\mathcal{C}_j$ in the clause gadget, and $\{s(x_i), s(\overline{x}_i)\} = \{0,1\}$ for each $i = 1,\dots,n$, as enforced by the constraints comprising the variable gadgets. Therefore, the feasible scheduling s can be translated into a truth assignment $\nu : \{x_1,\dots,x_n\} \to \{\texttt{true},\texttt{false}\}$ defined by $\nu(x_i) = \texttt{true}$ if $s(x_i) = 1$ (and $s(\overline{x}_i) = 0$); $\nu(x_i) = \texttt{false}$ if $s(x_i) = 0$ (and $s(\overline{x}_i) = 1$) for every $i = 1,\dots,n$.

To conclude, we observe that any hyperarc $A \in \mathcal{A}$ of $\mathcal{H}_\varphi$ has weights $w_A(\cdot) \in \{-1,0,1\}$ and size $|A| \le 3$. Since any hyperarc with three heads (tails) can be replaced by two hyperarcs each having at most two heads (tails), the consistency problem remains NP-Complete even if $w_A(\cdot) \in \{-1,0,1\}$ and $|A| \le 2$ for every $A \in \mathcal{A}$. □

Theorem 2.1 motivates the study of consistency problems on hypergraphs having either only multi-head or only multi-tail hyperarcs. In the former case, the consistency problem is called Head-HyTN-Consistency, while in the latter it is called Tail-HyTN-Consistency. In the following theorem we observe that the two problems are inter-reducible, i.e., we can solve consistency for any one of the two models in $f(m,n,W)$ time whenever we have a $f(m,n,W)$ time procedure for solving consistency for the other one.

Theorem 2.2. Head-HyTN-Consistency *and* Tail-HyTN-Consistency *are inter-reducible by means of* log*-space, linear-time, local-replacement reductions.*

Proof. We show the reduction from multi-tail to multi-head hypergraphs; the other direction is symmetric. Informally, what we will do is to reverse all the arcs (so that what was multi-tail becomes multi-head), and, contextually, we invert the time-axis (to account for the inversion of the direction of all arcs).

Let $\mathcal{H} = (V, \mathcal{A})$ be a multi-tail hypergraph, we associate to $\mathcal{H}$ a multi-head hypergraph $\mathcal{H}' = (V, \mathcal{A}')$ by reversing all multi-tail hyperarcs. Formally, we define

$$\mathcal{A}' = \{(v,S,w) \mid (S,v,w) \in \mathcal{A}\}.$$

For example, a standard arc $(t,h,w) \in \mathcal{A}$ is transformed into a reversed standard arc (h,t,w) in $\mathcal{A}'$ while a hyperarc with two weighted tails t_1 and t_2 becomes a hyperarc having t_1 and t_2 as its two weighted heads.

Now, $\mathcal{H}$ is consistent if and only if $\mathcal{H}'$ is consistent. To prove it, we note that each scheduling s for $\mathcal{H}$ can be associated, with a flip of the time direction, to the scheduling $s' \triangleq -s$. Then, it holds that s is feasible for $\mathcal{H}$ if and only if s' is feasible for $\mathcal{H}'$. Indeed, s satisfies the constraint represented by an hyperarc $A = (T_A, h_A, w_A) \in \mathcal{A}$, namely

$$s(h_A) \leq \max_{v \in T_A}\{s(v) + w_A(v)\},$$

or, equivalently

$$-s(h_A) \geq \min_{v \in T_A}\{-s(v) - w_A(v)\},$$

if and only if s' (that is, $-s$) satisfies the constraint represented by the reversed hyperarc $A' = (h_A, T_A, w_A)$, namely

$$s'(h_A) \geq \min_{v \in T_A}\{s'(v) - w_{A'}(v)\}.$$

$\square$

In the remainder of this work we shall adopt the multi-head hypergraph as our reference model. Hence, when considering hypergraphs and HyTNs, we will be implicitly referring to multi-head hyperarcs. Notably, we consider the following specialized notion of consistency for HyTNs.

Definition 2.3 (Head-HyTN-Consistency). *Given a (multi-head) HyTN, denoted by $\mathcal{H} = (V, \mathcal{A})$, decide whether there exists a scheduling $s : V \to \mathrm{R}$ such that:*

$$s(t_A) \geq \min_{v \in H_A}\{s(v) - w_A(v)\} \quad \forall A \in \mathcal{A}. \tag{2.1}$$

Remark 2.1. *Notice that this notion of consistency for HyTNs is a strict generalization of STN one. In general, the feasible schedules of an STN are the solutions of a linear system and, therefore, they form a convex polytope. Since an STN may be viewed as a HyTN, the space of feasible schedules of an STN can always be described as the space of feasible schedules of a HyTN. The converse is not true because feasible schedules for a HyTN do not form a convex polytope. Let us consider, for example, a HyTN of just three nodes x_1, x_2, x_3 and one single hyperarc with heads $\{x_1, x_2\}$ and tail x_3 expressing the constraint $x_3 \geq \min\{x_1, x_2\}$; $(0,2,2)$ and $(-2,0,2)$ are both admissible schedules, but $(1,1,0) = \frac{1}{2}(0,2,2) - \frac{1}{2}(-2,0,2)$ is not an admissible schedule. In conclusion, the STN model is a special case of the Linear Programming paradigm, whereas the HyTN model is not.*

In the rest of this section, we extend the characterization of STN consistency recalled in Section 6.2 to HyTNs.

Definition 2.4 (Reduced Slack Value $w_A^p(v)$). *With reference to a potential $p : V \to \mathrm{R}$, we define, for every arc $A \in \mathcal{A}$ and every $v \in H_A$, the reduced slack value $w_A^p(v)$ as $w_A(v) + p(t_A) - p(v)$ and the reduced slack w_A^p as*

$$w_A^p \triangleq \max\{w_A^p(v) \mid v \in H_A\}.$$

A potential p is said to be feasible *if and only if $w^p_A \geq 0$ for every $A \in \mathcal{A}$.*

Again, as it was the case for STNs, a mapping $f : V \to R$ is a feasible potential if and only if it is a feasible schedule. In order to better characterize feasible schedules, we introduce a notion of *negative cycle.*

Definition 2.5 (Negative Cycle)**.** *Given a multi-head HyTN $\mathcal{H} = (V, \mathcal{A})$, a cycle is a pair $(S, \mathcal{C})$ with $S \subseteq V$ and $\mathcal{C} \subseteq \mathcal{A}$ such that:*

1. *$S = \bigcup_{A \in \mathcal{C}}(H_A \cup \{t_A\})$ and $S \neq \emptyset$;*

2. *$\forall v \in S$ there exists an unique $A \in \mathcal{C}$ such that $t_A = v$.*

Moreover, we let $a(v)$ denote the unique arc $A \in \mathcal{C}$ with $t_A = v$ as required in previous item 2. Every infinite path in a cycle $(S, \mathcal{C})$ contains, at least, one finite cyclic sequence $v_i, v_{i+1}, \ldots, v_{i+p}$, where $v_{i+p} = v_i$ is the only repeated node in the sequence. A cycle $(S, \mathcal{C})$ is negative if for any finite cyclic sequence $v_1, v_2, \ldots, v_p$, it holds that

$$\sum_{t=1}^{p-1} w_{a(v_t)}(v_{t+1}) < 0.$$

An example of a cycle $(S, \mathcal{C})$ is shown in Fig. 2.7.

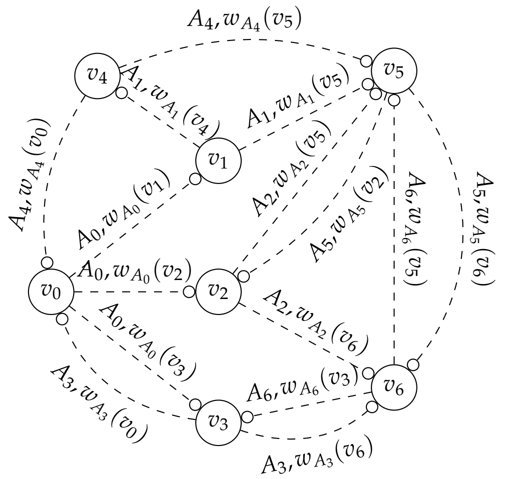


Figure 2.7: A Cycle $(S, \mathcal{C})$, where $S = \{v_0, \ldots, v_6\}$ and $\mathcal{C} = \{A_0, \ldots, A_6\}$.

There are two results about negative cycles as stated in the following lemmas.

Lemma 2.2. *A HyTN with a negative cycle admits no feasible schedule.*

Proof. By contraposition. Let $\mathcal{H}$ be a consistent HyTN and let p be a feasible potential for $\mathcal{H}$. Also, let $(S, \mathcal{C})$ be any cycle of $\mathcal{H}$; we will show that $(S, \mathcal{C})$ is not negative. For every $A \in \mathcal{C}$, let h_A be the head of A with maximum reduced slack value:

$$h_A \triangleq \arg\max_{v \in H_A} \{w^p_A(v)\}.$$

Let us consider the infinite path in $(S, \mathcal{C})$ built choosing, at each node v_t, $h_{a(v_t)}$ as the following node. As already seen, such a path contains at least one finite cyclic sequence $v_h, v_{h+1}, \ldots, v_k$ with $v_k = v_h$. The sum of weights of the finite cyclic sequence is given by:

$$\sum_{t=h}^{k-1} w_{a(v_t)}(v_{t+1}) = \sum_{t=h}^{k-1} w^p_{a(v_t)}(v_{t+1})$$

for every potential p; since p is feasible, all terms of the last sum are non-negative. It follows that $(S, \mathcal{C})$ is not negative. □

At first sight, it may appear that checking whether $(S, \mathcal{C})$ is a negative cycle might take exponential time since one should check a possibly exponential number of cyclic sequences. The next lemma shows instead that it is possible to check the presence of negative cycle in polynomial time.

Lemma 2.3. *Let $(S, \mathcal{C})$ be a cycle in a HyTN. Then checking whether $(S, \mathcal{C})$ is a negative cycle can be done in polynomial time.*

Proof. Consider the weighted graph $G = (S, \cup_{t \in S} A_t)$ where each hyperarc $a(t)$, for every $t \in S$, is transformed into a set of standard arcs as follows:

$$a(t) \rightsquigarrow A_t \triangleq \{(t, v, -w_{a(t)}(v)) \mid v \in H_{a(t)})\}, \ \forall t \in S.$$

Notice that G is thus an STN. Checking whether $(S, \mathcal{C})$ is a negative cycle amounts to check whether all cycles in G have strictly positive weight. To do this, firstly, a potential function π for G is determined by Bellman-Ford algorithm. If the algorithm returns a negative cycle instead of π, then there is *no* negative cycle in $(S, \mathcal{C})$ and the check ends.

Otherwise, since $w(C) = w^\pi(C) \geq 0$ for every cycle C of G, it is necessary to verify that no cycle in G has $w^\pi(C) = 0$. This check can be done by verifying the acyclicity of the subgraph of G comprising only arcs a of G with $w^\pi(a) = 0$. The check that a graph is acyclic can be done in linear time by a depth first visit [42]. □

A hypergraph $\mathcal{H}$ is called *conservative* when it contains no negative cycle. In the next sections we will provide a pseudo-polynomial time algorithm that always returns either a feasible scheduling or a negative cycle, thus extending the validity of the classical good-characterization of STN consistency to general HyTN consistency. Here, we anticipate the statement of the main result in order to complete this general introduction of HyTNs.

Theorem 2.3. *A HyTN $\mathcal{H}$ is consistent if and only if it is conservative. Moreover, when all weights are integral, then $\mathcal{H}$ admits an integral scheduling if and only if it is conservative.*

Proof. If $\mathcal{H}$ is consistent, then it is conservative by Lemma 2.2. If $\mathcal{H}$ is not consistent, then there is a negative cycle as shown in Theorem 2.7-(3). The existence of an integral scheduling when all weights are integral is guaranteed by Lemma 2.1. □

2.5 Mean Payoff Games

In this section, we propose an introduction to Mean Payoff Games (MPGs) tailored to the needs of the present work. MPGs represent a well-studied model for representing some kinds of two-player dynamics and we will show in Section 6 that there is a substantial equivalence between the MPG and the HyTN model, which will allow us to exploit some important algorithmic and structural results.

An MPG is a weighted directed graph $G = (V_0 \cup V_1, E)$ whose node set V is partitioned into two disjoint sets V_0 and V_1, where, for $p = 0,1$, the nodes in V_p are those under control of Player p. Even with these graphs we have no loops and no parallel arcs. It is also assumed that every node has at least one outgoing arc. Notice that, in general, (V_0, V_1) does not need to be a bipartiton of G, i.e., E may contain arcs with both endpoints in V_0, or with both endpoints in V_1. An example is shown in Fig. 2.8, where nodes in V_0 are white and those in V_1 are filled in black.

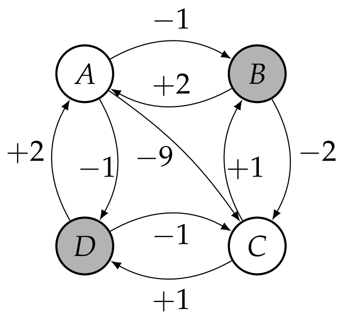


Figure 2.8: An MPG Γ.

Each play starts with a pebble placed at some node $v_0 \in V_0 \cup V_1$ and consists in a sequence of moves. Move t begins with the pebble placed in node v_{t-1} and is played by the Player p such that $v_{t-1} \in V_p$: the player chooses any arc $e \in E$ with tail $t_e = v_{t-1}$ and moves the pebble along e; at the end of the move the pebble is in node $v_t = h_e$. The game ends as soon as $v_t = v_{t'}$ for some $t > t'$, i.e., when the pebble comes back to an already visited node $v_{t'}$. At this point, the pebble has traversed a cyclic sequence of arcs $e_{t'+1}, \ldots, e_t$ and Player 0 "pays" to Player 1 the average weight of the visited cycle:

$$\frac{1}{t-t'} \sum_{i=t'+1}^{t} w(e_i).$$

If this amount is negative, then Player 0 *wins* the game, otherwise the winner is Player 1.

A *strategy* for Player p is a mapping that, given all the previous visited nodes and the current node, returns which node has to be visited in the next move; a strategy is said to be *positional* (or *memoryless*) if it depends only on the current position v_t and does not take into account all the previous history. If $s \in V_0 \cup V_1$ and Player p has a strategy leading him to win any possible play starting at $v_0 = s$, then we say that s is a *winning start position* for Player p. We denote by W_p the set of winning start positions for Player p. A *winning strategy* for Player p leads Player p to win every play started from any node in W_p. Since these finite games are zero-sum, i.e., what won by a player is what lost by the other one, then they admit a *game value* ν: for each start position $s \in V$ of the game, there exists a $\nu_s \in \mathsf{R}$ such that Player 0 has a strategy ensuring payoff at most ν_s, while Player 1 has a strategy ensuring payoff at least ν_s.

It is worthwhile to consider an infinite variant of the model, in which the game does not stop, and continues for an infinite number of steps. In this model, Player 1 wants to maximize the limit inferior of the average weight:

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} w(v_{t-1}, v_t)$$

Symmetrically, Player 0 wants to minimize the limit superior of the same average weight:

$$\limsup_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} w(v_{t-1}, v_t)$$

In their *Determinacy Theorem*, Ehrenfeucht and Mycielski [49] proved that any infinite game admits a value ν^{∞}, and that this value equals the one of the finite counterpart game on every start position, i.e., $\nu_s^{\infty} = \nu_s$ for every $s \in V_0 \cup V_1$. Moreover, they proved the existence of positional strategies which are *optimal* for both variants of the model: when Player p limits himself to an optimal strategy π_p, i.e., when, for every $v \in V_p$, he disregards all arcs with tail in v except the one with head in $\pi_p(v)$, then he will secure himself the optimal payoff ν in every play, finite or infinite, however the adversary plays. The graph G_{π_p} obtained from G by dropping all arcs with tail in V_p not prescribed by π_p is called the *projection* of the game G on π_p, and is a solitaire game whose value can be easily computed by means of a simple variant of Bellman-Ford algorithm. Therefore, the Ehrenfeucht and Mycielski's results are already sufficient for determining a simple exponential time algorithm computing the node values and the two optimal positional strategies in an MPG: the algorithm consists in evaluating each possible strategy for one of the two players as a solitaire game for determining the optimal one. In the literature there are many local search algorithms that explore this space in a more efficient way [9, 15, 103, 104] and some of them have been proven to be practically efficient in many settings by experiments [15, 103]. Moreover, the global optimization problem of computing the best strategies for one player, according to a given metric, has been shown to have the property that every local optimum is also a global one for many complete metrics [9].

As another line of research, Zwick and Paterson [123] proposed pseudo-polynomial time algorithms for computing values of games, as well as positional optimal strategies. In particular, they considered the following four algorithmic problems:

1. MPG-Decision(ν, s): given a real number ν and a start position s, decide whether $\nu_s \geq \nu$;
2. MPG-Threshold(T): given a real number T, determine for which nodes $s \in V$ it holds that $\nu_s \geq T$;
3. MPG-Value: compute the optimal values ν_s for all $s \in V$.
4. MPG-Synthesis: assuming $\nu_s \geq 0$ $(\nu_s < 0)$ for every $s \in V$, *synthesize* a positional winning strategy for Player 1 (Player 0);

and they proved the following theorem:

Theorem 2.4 ([123])**.** *Let $G = (V, E)$ be a mean payoff game. Assume all weights are integers and let $W = \max_{e \in E} |w(e)|$. Then the following hold:*

1. *MPG-Threshold(T) can be solved in time $\mathcal{O}(|V|^2|E|W)$ when $T \in \mathbb{Z}$, whereas it can be solved in time $\mathcal{O}(|V|^3|E|W)$ when $T \in \mathsf{R}$;*
2. *MPG-Value can be solved in time $\mathcal{O}(|V|^3|E|W)$;*
3. MPG-Synthesis *can be solved in time $\mathcal{O}(|V|^4|E|\log(|E|/|V|)W)$.*

Then, they observed that MPG-Decision is the basic decision problem for MPGs in the sense that several natural questions for MPGs, like evaluating the value ν_s for every node s or constructing the optimal positional strategies, may all be Turing-reduced to it. They also pointed out that the existential results of Ehrenfeucht and Mycielski [49] already implies that $MPG - Decision \in \mathsf{NP} \cap \mathsf{coNP}$ and asked whether there might exist a strongly polynomial time decision procedure. Proving the existence of such algorithm is an open problem [14]. Finally, they showed how to reduce mean payoff games to other important families of games on graphs, like discounted payoff games and simple stochastic games.

The complexity status of MPG-Decision has been since updated by proving that it lays in $\mathsf{UP} \cap \mathsf{coUP}$ by Jurdziński in [70].

In recent years, some other interesting results have been proven. Notably, in 2007 Lifshits, Pavlov [79] proposed a *potential theory* for MPGs and in 2011 Brim et al. [14] obtained faster algorithms exploiting results obtained in the the fields of *Energy Games* and *energy progress measures*, which are intimately related to the potentials studied in [79].

Their algorithmic results are summarized in the following theorem.

Theorem 2.5 ([14]). *For MPGs in which all weights are integers and for $T \in \mathbb{Z}$, the Value Iteration Algorithm [14] solves MPG-Threshold(T) and* MPG-Synthesis *in time $O(|V||E|W)$, where $W = \max_{e \in E} |w(e)|$.*

We remark that both the algorithm of Paterson and Zwick [123] and the Value Iteration Algorithm [14] prescribe well defined procedures even if the weights on the arcs are real values. What is lost in running these algorithms on real weights is only the pseudo-polynomial upper bound on their running time.

For our purposes, the family of pseudo-polynomial algorithms for MPGs is the best option. Indeed, in most of temporal workflow graphs all weights are expressed by integers of relatively small magnitude with respect to the intrinsic temporal granularity of the considered workflow. For example, in a workflow containing temporal distance constraints of days, the commonly adopted temporal granularity is the "minute" (m) and, therefore, all weights can be assumed to be less than 10^4 as order of magnitude. In such circumstances, Brim's algorithm offers the guarantee to terminate within short computation times. For these reasons we opted for integrating the procedures of Zwick and Paterson, as well as the faster procedures of Brim et al. [14], in order to efficiently solve instances of Head-HyTN-Consistency and compute feasible schedules.

Furthermore, as will be discovered in the experimental section, if these algorithms are suitably adapted—so as to allow them to terminate earlier as soon as certain evidences of inconsistency have been collected—then their observed behavior outperforms by orders of magnitude what predicted by their theoretical pseudo-polynomial bounds even on input instances containing very large integer values.

Based on these findings, we think that these pseudo-polynomial algorithms are to be considered (and probably adopted) even for solving HyTN instances where weights are floating point values whose magnitudes may differ in a significant way. In case the running time results to be unacceptable for a real application, one could then consider the possibility to round the weights to integer values. This rounding would clearly require special care: a very accurate approximation might lead to very high computation times while a gross approximation might not represent the original instance in a correct way.

2.6 The Reductions

This section presents the direct connection and the computational equivalence between MPG-Threshold and Head-HyTN-Consistency. The equivalence is formally proven by offering one reduction in each direction.

The reduction of Head-HyTN-Consistency to MPG-Threshold allows to apply, in the context of HyTNs, any of the algorithms known for MPGs, included the exponential and subexponential ones.

Vice versa, in consideration of the fact that the $MPG - Decision \overset{?}{\in} \mathsf{P}$ question is an open problem [9, 14, 70, 103, 123], the reduction of MPG-Decision to Head-HyTN-Consistency confirms that Head-HyTN-Consistency offers an algorithmically more ambitious and mathematically steeper generalization of STN-Consistency (see also Remark 2.1). Moreover, the reduction gives a further evidence that, within STNs, a new algorithmic approach is necessary in order to manage temporal aspects of event like the synchronization one presented in the Introduction.

Let us start considering the first reduction.

Theorem 2.6. *There exists a* log*-space*[2]*, linear-time, local-replacement*[3] *reduction from* Head-HyTN-Consistency *to MPG-Threshold.*

Since this reduction plays a main role in the algorithmic solutions proposed in this chapter, we firstly describe how it works and, secondly, we prove its correctness by means of two lemmas, Lemma 2.4 and Lemma 2.5.

The reduction goes as follows.

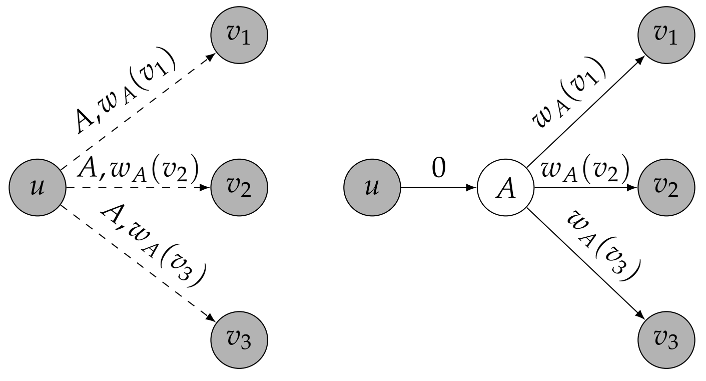
Let $\mathcal{H} = (V, \mathcal{A})$ be a HyTN. We assume that every $v \in V$ is the tail of some arc $A \in \mathcal{A}$. This assumption is not a restriction since, if $\mathcal{H}$ contains a *sink node* v, i.e., a node v with no arc $A \in \mathcal{A}$ having tail in it, then $\mathcal{H}$ is consistent if and only if so is $\mathcal{H}_v$, the HyTN obtained from $\mathcal{H}$ by removing node v and every hyperarc having v as an head. Indeed, any feasible scheduling $s : V \mapsto \mathsf{R}$ for $\mathcal{H}$, once projected onto $V \setminus \{v\}$, gives a feasible scheduling for $\mathcal{H}_v$ since every constraint involving v has been dropped and no constraint has been added; conversely, any feasible scheduling s for $\mathcal{H}_v$ can be easily extended to a feasible scheduling s for $\mathcal{H}$ by exploiting the property of v being a sink node: it is sufficient to set $s(v) \triangleq \min\{s(t_A) - w_A(v) \mid A \in \mathcal{A}, v \in H_A\}$.

Now, let us consider a mean payoff game $G_{\mathcal{H}} = (V_0 \uplus V_1, E)$ where: (1) $V_0 = V$, $V_1 = \mathcal{A}$, nodes in V_0 are colored by *black* while nodes in V_1 are colored by *white*, and (2) for each $A \in \mathcal{A}$, the following weighted arcs are added to E:

- an arc of weight 0 from the black node t_a to the white node A, i.e., arc $(t_A, A, 0)$;
- for each head node $h \in H_A$, an arc of weight $w_A(h)$ from the white node A to the black node h, i.e., arc $(A, h, w_A(h))$.

[2] A strong and basic-form of reduction introduced by Papadimitriou in [96].

[3] A restricted kind of Karp reduction introduced in [58].



(a) Hyperarc A. (b) MPG representation of A.

Figure 2.9: The conversion of a hyperarc into a white node and its incident arcs.

Algorithm 1: `makeACorrespondingGame`$(\mathcal{H})$

```
   // a HyTN $\mathcal{H} = (V, \mathcal{A})$
1  $V_0 \leftarrow V$;
2  $V_1 \leftarrow \mathcal{A}$;
3  $E \leftarrow \emptyset$;
4  foreach $A \in \mathcal{A}$ do
5      $E \leftarrow E \cup (t_A, A, 0)$;
6      foreach $h \in H_A$ do
7          $E \leftarrow E \cup (A, h, w_A(h))$;
   Output: The MPG $G_{\mathcal{H}} = (V_0 \uplus V_1, E)$
```

Algorithm 1: The algorithm implementing the reduction from a HyTN to the corresponding MPG.

In short, $G_{\mathcal{H}} = (V_0 \uplus V_1, \mathcal{A})$, with $V_0 = V$, $V_1 = \mathcal{A}$, $E = \{(t_A, A, 0) \mid A \in \mathcal{A}\} \cup \{(A, h, w_A(h)) \mid A \in \mathcal{A}, h \in H_A\}$. Fig. 2.9 depicts how a hyperarc is transformed into a MPG subnetwork while Algo. 1 reports a pseudocode for the whole construction process, i.e., Algorithm 1.

$G_{\mathcal{H}}$ has $|V| + |\mathcal{A}|$ nodes and $O(m)$ arcs and can be constructed in linear time. Moreover, $G_{\mathcal{H}}$ is a bipartite graph with bipartition (V_0, V_1) and it has been obtained from $\mathcal{H}$ by a simple local replacement rule: replace every hyperarc $A \in \mathcal{A}$ by a claw subgraph as depicted in Fig. 2.9. For each single object, it is necessary only to manage a constant number of indexes, each of them having a polynomial size; thus the reduction is log-space. Fig. 2.10 depicts an MPG obtained applying the reduction to the motivating example HyTN depicted in Fig. 2.3; we remark that the MPG depicted in Fig. 2.10 has been obtained by considering the (equivalent) multi-head HyTN transformation of the multi-tail HyTN shown in Fig. 2.3, indeed, Theorem 2.2 allows us to consider both

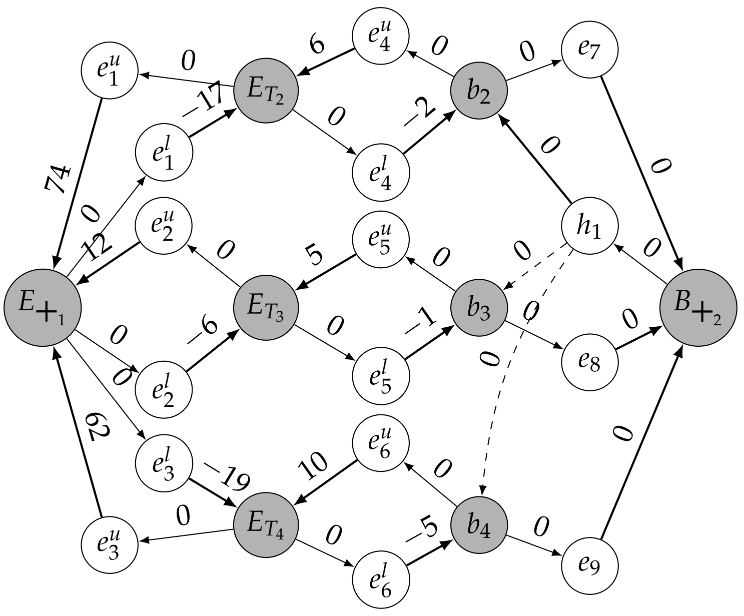


Figure 2.10: The MPG equivalent to the HyTN depicted in Fig. 2.3, obtained by considering the (equivalent) multi-head HyTN transformation of the multi-tail HyTN shown in Fig. 2.3. A winning positional strategy π_1 for Player 1 is highlighted by thick arcs. The dashed arcs are those not prescribed by strategy π_1, i.e., they are removed when projecting the MPG on π_1.

the multi-head or the (equivalent) multi-tail HyTN without concerns w.r.t. the consistency checking problem.

Now, let us introduce the formal proof of Theorem 2.6 by the following two lemmas.

Lemma 2.4. *If $\mathcal{H}$ is consistent then every node of $G_{\mathcal{H}}$ is a winning start position for Player* 1.

Proof. Since $\mathcal{H}$ is consistent, there exists a feasible scheduling $s : V \to \mathrm{R}$ such that, for each hyperarc $A \in \mathcal{A}$, the reduced slack weight is non-negative $w^s_A \geq 0$. Consider the following positional strategy π_1 for Player 1: for each $A \in V_1$,

$$\pi_1(A) = \arg\min_{h \in H_A} \{s(h) - w_A(h)\}.$$

We claim that π_1 ensures Player 1 the win, wherever node the game starts from and however Player 0 moves. In order to show this, we prove that the projection graph G_{π_1} is conservative by exhibiting a feasible potential p. Let $p : V_0 \cup V_1 \to \mathrm{R}$ be defined as follows:

$$p(v) \triangleq \begin{cases} s(v) & \text{if } v \in V_0, \\ s(t(v)) & \text{if } v \in V_1. \end{cases} \tag{2.2}$$

Now, let $a = (u,v,w)$ be any arc of G_{π_1}:
Case 1: if $v \in V_1$, then v is a hyperarc of $\mathcal{H}$ with $t(v) = u$ and $w = 0$; therefore, $p(v) = s(t(v)) = s(u) = p(u)$ since $u \in V_0$. Then $w^p(u,v) = w - p(v) + p(u) = 0 \geq 0$ follows;
Case 2: if $v \in V_0$, then $u \in V_1$ and $w = w_u(v)$. Moreover, $v = \pi_1(u)$, which implies that $v = \arg\min_{h \in H_u}\{s(h) - w_u(h)\}$. Therefore, recalling that $w_u^s \geq 0$, i.e., $s(t(u)) \geq \min_{h \in H_u}\{s(h) - w_u(h)\}$:

$$p(u) = s(t(u)) \geq \min_{h \in H_u}\{s(h) - w_u(h)\} = s(v) - w_u(v) = p(v) - w.$$

Hence, $w^p(u,v) = w - p(v) + p(u) \geq 0$.
In conclusion, G_{π_1} is conservative. Therefore, the positional strategy π_1 certifies that any node of $G_{\mathcal{H}}$ is a winning start position for Player 1. □

Lemma 2.5. *If every node of $G_{\mathcal{H}}$ is a winning start position for Player 1 then $\mathcal{H}$ is a consistent HyTN.*

Proof. If every node is a winning start position for Player 1, then there exists a positional strategy π_1 which is everywhere winning for Player 1. Notice that G_{π_1} must be conservative since Player 0 can clearly win any play starting from a node located on a negative cycle. Let $p : V_0 \cup V_1 \to \mathrm{R}$ be a feasible potential for G_{π_1}. We claim that the restriction of p onto V_0 is a feasible scheduling for $\mathcal{H}$. Indeed, for any hyperarc A of $\mathcal{H}$, $(t_A, A, 0)$ is an arc of G_{π_1}, whence $p(A) \leq p(t_A)$. Moreover, $(A, \pi_1(A), w_A(\pi_1(A)))$ is also an arc of G_{π_1}, whence $p(\pi_1(A)) \leq p(A) + w_A(\pi_1(A))$. Since $\pi_1(A) \in H_A$, then the following holds:

$$\begin{aligned} p(t_A) \geq p(A) &\geq p(\pi_1(A)) - w_A(\pi_1(A)) \\ &\geq \min_{h \in H_A}\{s(h) - w_A(h)\}. \end{aligned}$$

Hence, the restriction of p onto V_0 is a feasible scheduling for $\mathcal{H}$. Thus, $\mathcal{H}$ is consistent. □

In Fig. 2.11 the values under the nodes represent a feasible potential for the projection of the MPG depicted in Fig. 2.10. By Lemma 2.5, the restriction of such a feasible potential on the black nodes is also a feasible scheduling for the corresponding HyTN depicted in Fig. 2.3. Now, we have all the necessary results to prove the following theorem.

Theorem 2.7. *Let $\mathcal{H} = (V, \mathcal{A})$ be an integral-weighted HyTN, $m = \sum_{A \in \mathcal{A}} |A|$, and $W = \max_{A \in \mathcal{A}}\{\max_{h \in A} |w_A(h)|\}$ the maximal weight value present in $\mathcal{H}$. The following propositions hold:*

1. *There exists an $O((|V| + |\mathcal{A}|)mW)$ pseudo-polynomial time algorithm deciding* Head-HyTN-Consistency *for $\mathcal{H}$;*

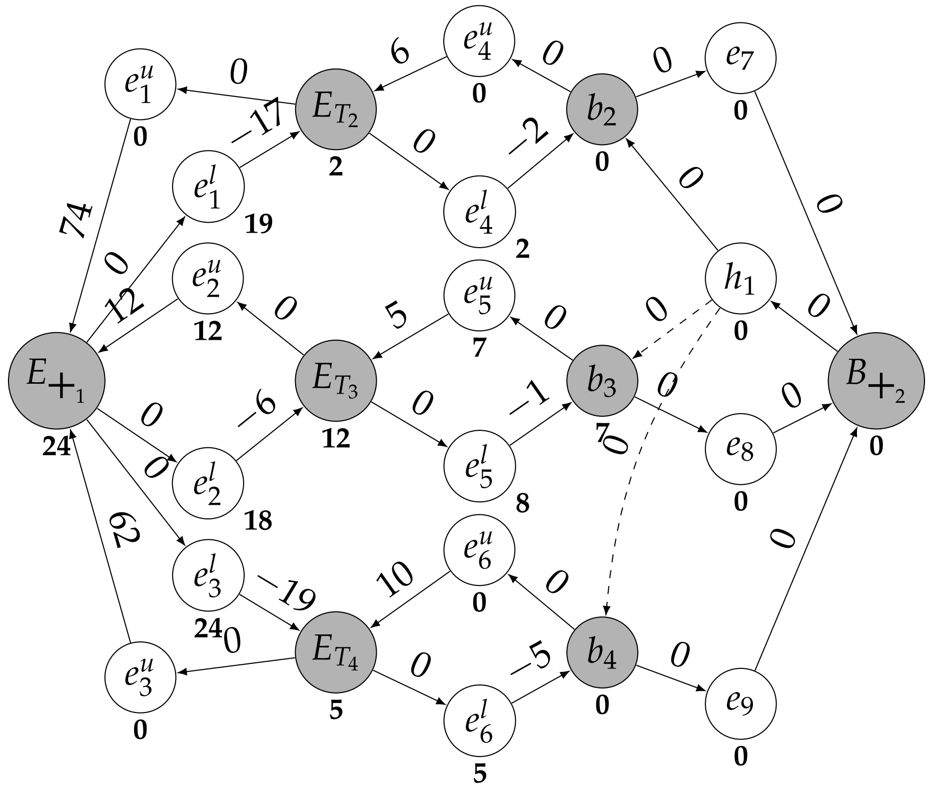


Figure 2.11: The integer labels under the nodes are a feasible potential for the projection on π_1 of the MPG depicted in Fig. 2.10. The restriction of this potential on the black nodes (those in V_0) is a feasible scheduling for the HyTN depicted in Fig. 2.3 as explained in the proof of Lemma 2.5.

2. *There exists an $O((|V| + |\mathcal{A}|)mW)$ pseudo-polynomial time algorithm such that, given on input any consistent HyTN $\mathcal{H}$, it returns as output a feasible scheduling $s : V_{\mathcal{H}} \to \mathbb{Z}$ of $\mathcal{H}$;*

3. *There exists an $O((|V| + |\mathcal{A}|)mW)$ pseudo-polynomial time algorithm such that, given on input any not-consistent HyTN $\mathcal{H}$, it returns as output a negative cycle $(S,\mathcal{C})$ of $\mathcal{H}$.*

Proof. 1. The decision algorithm is sketched in Algo. 2. It takes in input a HyTN $\mathcal{H} = (V, \mathcal{A})$ and, in line 1, constructs the corresponding MPG $G_{\mathcal{H}}$ as described in Theorem 2.6. This first step takes $O(m)$ time and yields a graph with $|V| + |\mathcal{A}|$ nodes and $O(m)$ arcs. Then, in line 2, the instance of MPG-Threshold with $T = 0$ on graph $G_{\mathcal{H}}$ is solved in $O((|V| + |\mathcal{A}|)mW)$ time by the Value Iteration Algorithm (see Theorem 2.5). The output consists in a partition of $G_{\mathcal{H}}$ nodes into two sets: $W_1 = \{v \in V \cup \mathcal{A} \mid \nu_v \geq 0\}$ and $W_0 = \{v \in V \cup \mathcal{A} \mid \nu_v < 0\}$. If W_0 is empty, then $\mathcal{H}$ is consistent by Lemma 2.5, otherwise it is not consistent by Lemma 2.4.

```
Algorithm 2: isConsistent($\mathcal{H}$)
  // a HyTN $\mathcal{H} = (V, \mathcal{A})$ of unknown consistency state
1 $G_{\mathcal{H}} \leftarrow$ makeACorrespondingGame($\mathcal{H}$); // See Algorithm 1
2 $(W_0, W_1) \leftarrow$ solveMPG-Threshold($G_{\mathcal{H}}$,0); // Brim's algorithm, see Theorem 2.5
3 if $(W_0 = \emptyset)$ then Output: YES;
4 else Output: NO;
```

Algorithm 2: Pseudocode of the algorithm for deciding Head-HyTN-Consistency.

```
Algorithm 3: computeAFeasibleSchedule($\mathcal{H}$)
  // a consistent HyTN $\mathcal{H} = (V, \mathcal{A})$
1 $G \leftarrow$ makeACorrespondingGame($\mathcal{H}$); // See Algorithm 1
2 $\pi_1 \leftarrow$ MPG-Synthesis ($G$); // Compute a positional winning strategy for Player 1;
    see Theorem 2.5
3 $G_{\pi_1} \leftarrow$ compute the subgraph of $G$ induced by $\pi_1$;
  // Recall $G_{\pi_1} = (V_0 \uplus V_1, E)$, where $V_0 = V$ and $V_1 = \mathcal{A}$.
4 $s \leftarrow$ a new node; // $s \notin V_0 \cup V_1$
5 Add $s$ to $V_1$ and add an arc $(s, v, 0)$ for each $v \in V_0$;
6 $p \leftarrow$ Bellman-Ford($G_{\pi_1}$,$s$); // compute a potential function $p$
  Output: the restriction of $p$ onto $V$
```

Algorithm 3: Pseudocode of the algorithm for computing a feasible schedule.

2. In case W_0 is empty, a feasible scheduling is obtained as shown in Algorithm 3. First, in line 2, the algorithm computes a positional winning strategy π_1 for Player 1. This takes $O((|V| + |\mathcal{A}|)mW)$ time by Theorem 2.5. Next, in line 3, it builds the graph G_{π_1} which is conservative since π_1 is a positional winning strategy for Player 1. Then, in lines 4-5, it adds a new node s to V_1 and a new arc $e_v = (s, v, 0)$ for each node $v \in V_0$ in G_{π_1}. Let $G'_{\pi_1} = (V_0 \uplus (V_1 \cup \{s\}), E')$ the graph thus obtained. Observe that every node of G'_{π_1} is reachable from s. Indeed, every node $A \in V_1 = \mathcal{A}$ can be reached by traversing two arcs: from s to t_A along the arc $e_{t_A} = (s, t_A, 0)$, which belongs to G'_{π_1} as $t_A \in V_0$, then from t_A to A along the arc $(t_A, A, 0)$, which belongs to G_{π_1} (and hence to G'_{π_1}) since $t_A \in V_0$.
   Since the added node s is a source, then G'_{π_1} is conservative too. Therefore, in G'_{π_1}, the set of distances from node s, computed calling the Bellman-Ford algorithm in line 6, forms a feasible potential $p : V_0 \cup V_1 \cup \{s\} \to \mathbb{Z}$ and the restriction of p onto $V_0 = V$ is a feasible scheduling for $\mathcal{H}$.

3. In case W_0 is not empty, a negative cycle is determined by Algorithm 4. Let $G[W_0]$ be the subgraph of G induced by W_0, i.e., the graph obtained from G by removing all nodes not in W_0 and all the arcs incident into

Algorithm 4: `computeANegativeCycle`$(\mathcal{H}, W_0)$

```
// a HyTN $\mathcal{H} = (V, \mathcal{A}) = (V_0 \uplus V_1, \mathcal{A})$ which is not consistent
// the non-empty set $W_0 = \{v \in V \mid \nu_v < 0\}$
1 $G \leftarrow$ makeACorrespondingGame$(\mathcal{H})$; // See Algorithm 1
2 $G[W_0] \leftarrow$ compute the subraph of $G$ induced by $W_0$;
3 $\pi_0 \leftarrow$ MPG-Synthesis $(G[W_0])$; // Compute a positional winning strategy for
    Player 0; see Theorem 2.5
4 $\overline{W}_0 \leftarrow W_0 \cap V_0$;
5 $\mathcal{C} \leftarrow \{\pi_0(v)\}_{v \in \overline{W}_0}$;
  **Output**: $(\overline{W}_0, \mathcal{C})$
```

Algorithm 4: Pseudocode of the algorithm for computing a negative cycle.

them. Notice that every node $v \in W_0$ is a winning start position for Player 0 in game $G[W_0]$ because v is a winning start position for Player 0 in game G, and no winning strategy for Player 0 in G can prescribe a move from a node in W_0 to a node in W_1; therefore, that same winning strategy remains valid on $G[W_0]$. This implies that, for every $u \in W_0$, there exists at least one arc (u, v) with $v \in W_0$. In particular, since (V_0, V_1) is a bipartition of G, then $\overline{W}_0 \triangleq W_0 \cap V_0 \neq \emptyset$. In line 3, a positional winning strategy π_0 for Player 0 on $G[W_0]$ is determined. By Theorem 2.5, this computation takes time $O((|V| + |\mathcal{A}|)mW)$. Consider the set of hyperarcs $\mathcal{C} = \{\pi_0(v)\}_{v \in \overline{W}_0}$; the pair $(\overline{W}_0, \mathcal{C})$ returned by the algorithm is a negative cycle. Indeed, for any $v \in \overline{W}_0$, $\pi_0(v) \in V_1$ is a hyperarc of $\mathcal{H}$. Thus the head set $H_{\pi_0(v)} \subseteq V_0$. Also, $H_{\pi_0(v)} \subseteq W_0$, since v is a winning start position for Player 0 and π_0 is a winning strategy for Player 0. Combining, $H_{\pi_0(v)} \subseteq \overline{W}_0$ determining that $(\overline{W}_0, \mathcal{C})$ is a negative cycle.

□

Remark 2.2. *In Theorem 2.7 Item 2), a set of feasible potentials may be obtained without executing the Bellman-Ford algorithm. Actually, if the partition (W_0, W_1) is computed by the Value Iteration Algorithm [14], then a feasible scheduling for $\mathcal{H}$ can be directly derived from the* progress measure *computed within the algorithm. In more detail, let $G = (V_0 \uplus V_1, E)$ be an MPG weighted by $w : E \to \mathbb{Z}$. An* energy progress measure *is a function $f : V_0 \cup V_1 \to \mathbb{N} \cup \{+\infty\}$ such that: if $v \in V_0$, then for every $(v, v', w) \in E$ it holds $f(v) \geq f(v') - w$; otherwise, $v \in V_1$ and there exists $(v, v', w) \in E$ such that $f(v) \geq f(v') - w$. An energy progress measure $f : V_0 \cup V_1 \to \mathbb{N} \cup \{+\infty\}$ such that $0 \leq f(v) < +\infty$ for every $v \in V_0 \cup V_1$ is provided by the resolution algorithm of Theorem 2.5 in time $O((|V| + |\mathcal{A}|)mW)$.*

The progress measure f is already a feasible scheduling for $\mathcal{H}$: in fact, for every hyperarc $A \in \mathcal{A}$, it holds $(t_A, A, 0) \in E$ and $(A, v, w_A(v)) \in E$, for every $v \in H_A$;

Algorithm 5: `computeAFeasibleSchedule-Remark2.2`(H)

```
// a consistent HyTN $\mathcal{H} = (V, \mathcal{A}) = (V_0 \uplus V_1, \mathcal{A})$
// ref. Remark 2.2 and Theorem 2.5 [14]
1 $G \leftarrow$ makeACorrespondingGame$(\mathcal{H})$; // ref. Algorithm 1
2 $f \leftarrow$ Value-Iteration$(G)$; // compute an energy progress measure for $G$ as in
    Theorem 2.5
Output: $f$
```

Algorithm 5: Pseudocode of the algorithm of Remark 2.2 for computing a feasible schedule.

combining these two last facts, it follows that:

$$f(t_A) \geq f(A) \geq \min_{v \in H_A} \{f(v) - w_A(v)\},$$

i.e., f is a scheduling satisfying all constrains $A \in \mathcal{A}$. This allow us to employ the algorithm depicted in Algo. 5 instead of the one depicted in Algo. 3 in the case that $W_1 = V$.

The computational equivalence between MPG-Decision problem and Head-HyTN-Consistency can be now determined by showing that also MPG-Decision can be reduced to Head-HyTN-Consistency.

Theorem 2.8. *There exists a* log*-space, linear-time, local-replacement reduction from MPG-Decision to* Head-HyTN-Consistency*.*

Proof. Let $G = (V_0 \uplus V_1, E)$ be an MPG. For each node $u \in V_0 \cup V_1$, let $N_G(u)$ denote the outgoing neighborhood of u in G, i.e., $N_G(u) \triangleq \{v \in V_0 \cup V_1 \mid (u,v) \in E\}$.

A corresponding HyTN $\mathcal{H} = (V, \mathcal{A})$, where $V = V_0 \uplus V_1$, is constructed from G as follows. For every $u \in V_1$, a hyperarc $A_u \in \mathcal{A}$ is added to $\mathcal{H}$, where:

$$A_u \triangleq (u, N_G(u), w_{A_u}),$$

with weight $w_{A_u}(v) \triangleq w(u,v)$ for every $v \in N_G(u)$. Moreover, for every $u \in V_0$ and every $v \in N_G(u)$, a hyperarc $A_{uv} \in \mathcal{A}$ is added to $\mathcal{H}$, where:

$$A_{uv} \triangleq (u, v, w(u,v)).$$

This construction requires a log-space and linear-time computation.

Now, we firstly prove that if $\mathcal{H}$ is consistent then every node of G is a winning start position for Player 1.

Indeed, let $s : V \to \mathrm{R}$ be a feasible scheduling for $\mathcal{H}$. Thus, $w^s_A \geq 0$ for every hyperarc $A \in \mathcal{A}$. Notice that, by construction, for each $u \in V_1$ there exists a unique hyperarc $A_u \in \mathcal{A}$ with tail $t_{A_u} = u$; moreover, it holds that $H_{A_u} \triangleq N_G(u)$. Hence, for each $u \in V_1$, we can define a positional strategy π_1 for Player 1 as follows:

$$\pi_1(u) \triangleq \arg\min_{h\in H_{A_u}} \{s(h) - w_{A_u}(h)\}.$$

Now, consider the potential function $p : V \to \mathrm{R}$ defined as: $p(u) \triangleq s(u)$ for every $u \in V$. We argue that p is a feasible potential for G_{π_1}.

In fact, let $a = (u,v,w) \in E$ be any arc of G:

Case 1: Assume that $u \in V_0$. Then, by construction, $a = A_{uv}$. Hence, from $p \triangleq s$ and the feasibility of s, we have:

$$\begin{array}{rcl} p(u) = s(u) & \geq & \min_{h\in H_{A_{uv}}}\{s(h) - w_{A_{uv}}(h)\} \\ & = & s(v) - w_{A_{uv}}(v) \\ & = & p(v) - w \end{array}$$

Hence, $w^p(u,v) = w - p(v) + p(u) \geq 0$;

Case 2: Assume that $u \in V_1$. Then, by construction, $A = (u, N_G(u), w_A)$, where $w_A(z) = w(u,z)$ for every $z \in N_G(u)$; moreover, notice that $v = \pi_1(u) \in N_G(u) = H_A$. Hence, from $p \triangleq s$, the feasibility of s, and the definition of π_1, we have:

$$\begin{array}{rcl} p(u) = s(u) & \geq & \min_{h\in H_u}\{s(h) - w_{A_u}(h)\} \\ & = & s(\pi_1(u)) - w_{A_u}(\pi_1(u)) \\ & = & s(v) - w_{A_u}(v) \\ & = & p(v) - w \end{array}$$

Hence, $w^p(u,v) = w - p(v) - p(u) \geq 0$.

Thus, G_{π_1} is conservative. This implies that every node of G is a winning start for Player 1.

Secondly, we prove that if every node of G is a winning start position for Player 1, then H is consistent.

Let π_1 be a positional winning strategy for Player 1. It follows that G_{π_1} is conservative and, therefore, it admits a feasible potential $p : V \to \mathrm{R}$. Now, consider the scheduling function $s : V \to \mathrm{R}$ for $\mathcal{H}$ defined as: $s(u) \triangleq p(u)$ for every $u \in V$. We argue that s is a feasible scheduling of $\mathcal{H}$.

In fact, let $A = (t_A, H_A, w_A) \in \mathcal{A}$ be any hyperarc of $\mathcal{H}$:

Case 1: assume $t_A \in V_0$. Then, by construction, $A = (u,v,w)$ for some $v \in N_G(u), w \in \mathrm{R}$ and $u = t_A$. Hence, from $s \triangleq p$ and the feasibility of p, we have:

$$\begin{array}{rcl} s(t_A) = p(u) & \geq & p(v) - w \\ & = & s(v) - w_A(v) \\ & = & \min_{h\in H_A}\{s(h) - w_A(h)\} \end{array}$$

Hence, s satisfies A, i.e., $w^s_A \geq 0$;

Case 2: assume $t_A \in V_1$. Then, by construction, $A = (u, N_G(u), w_A)$ for $u = t_A$ and $w_A(v) = w(u,v) \in \mathrm{R}$ for every $v \in N_G(u)$; moreover, if $v \triangleq \pi_1(u)$, then $v \in N_G(u) = H_A$. Hence, from $s \triangleq p$ and the feasibility of p, we have:

$$\begin{array}{rcl} s(t_A) = p(u) & \geq & p(v) - w \\ & = & s(v) - w_A(v) \\ & \geq & \min_{h \in H_A}\{s(h) - w_A(h)\} \end{array}$$

Hence, s satisfies A, i.e., $w^s_A \geq 0$.

This proves that s satisfies every hyperarc $A \in \mathcal{A}$. Then s is a feasible scheduling of $\mathcal{H}$, which is thus consistent. □

2.7 Computational Experiments

This section describes our empirical evaluation of the proposed consistency checking algorithms to evaluate the performances and the general applicability of the proposed HyTN model. Both Algorithm 3 and Algorithm 4 consist of one single call to Algorithm 2, plus some extra computation of lower asymptotic complexity. Since the cost of these further computations was confirmed to be practically negligible in some preliminary experiments, we report on the results of our experimental investigations only for Algorithm 2.

All algorithms and procedures employed in this empirical evaluation have been implemented in C/C++ and executed on a Linux machine having the following characteristics:

- 2 CPU AMD Opteron 4334;
- 64GB RAM;
- Ubuntu server 14.04.1 Operating System.

The source code and all HyTNs used in the experiments are freely available [30].

The main goal of this empirical evaluation was to determine the average computation time of Algorithm 2, with respect to randomly-generated HyTNs following different criteria, in order to give an idea of the practical behavior of the algorithm. According to Theorem 2.7, the worst-case time complexity of Algorithm 2 is $O((n + m')mW)$, where $n = |V|$, $m' = |\mathcal{A}|$, $m = \sum_{A \in \mathcal{A}} |A|$, and $W = \max_{A \in \mathcal{A}}\{\max_{h \in A} |w_A(h)|\}$. Hence, we implemented different experiments with respect to the parameters n, m', m, and W. Here we propose a summary of the obtained results presenting a brief report about four tests, Test 1, Test 2, Test 3 and Test 4.

In Test 1 the average computation time was determined for different HyTN orders n to emphasize the practical computation time dependency on n. In Test 2 the average computation time was determined for different HyTN maximal edge-weights W to understand how much the practical computation time is dependent on W. In Test 3 we investigated how some execution times affect the value of the standard deviation, with the goal to determine how many instances require a significant greater computation time with respect to the average time of a data set. Finally, in Test 4 the average computation time was determined with respect to different values of the number of possible strategies of Player 1 $\prod_{A \in \mathcal{A}} |H_A|$ in order to give an idea about the possible practical relation between execution time and number of possible strategies.

The generation of random HyTN instances was carried out exploiting two generators. The first generator was the random workflow schema generator provided by ATAPIS toolset [76]: it produces random workflow graphs according to different input parameters that allow to control the minimal and maximal number of activities, probability for having parallel branches, the minimal and maximal probability of inter-task temporal constraints, etc. on the generated graphs. We verified that this tool allows the determination of graphs that are not only a closer approximation to real-world examples, but also more difficult to check than those generated at random without particular criteria.

We generated benchmarks as follows:

1. First, temporal workflow graphs were generated by fixing the probability for parallel branches to 10% and maximal value for each activity duration or delay between activities to a value W, where W was chosen accordingly to the test type;

2. Then, each workflow graph was translated into an equivalent HyTN $\mathcal{H}$ by the simple transformation algorithm exemplified in Section 2.2.

It is worth noting that different random workflow graphs all having the same number of activities may translate into HyTNs having different orders n because the original workflow graphs may have different number of connector nodes. Considering the transformation algorithm exemplified in Section 2.2, it is easy to verify that a workflow with N activities can translate into a CSTN having between $2N + 2$ nodes (when the workflow is a simple sequence) and $5N + 2$ nodes (when the workflow is a sequence of groups of two parallel activities).

ATAPIS toolset has been designed to generate graphs with strongly connected components (Andreas Lanz, personal communication, October 6, 2015). In particular, it has been optimized for small graphs with up to 50 activities. This design choice was motivated by the widely accepted seven process modeling guidelines [85] which suggests to always "decompose a (workflow) model with more than 50 elements (activities)". Therefore, we used the tool for generating random workflow graphs with 100 activities at maximum and, consequently, obtaining HyTNs having 502 nodes at most.

In Table 2.1 we report the orders of the smallest and the largest HyTN determined from each set of random generated workflow graphs having N activities for $N \in \{10, 20, \ldots, 100\}$.

Table 2.1: Orders of the smallest and biggest HyTN determined for each set of random generated workflows having N activities.

N	Order of smallest HyTN	Order of biggest HyTN
10	26	50
20	48	94
30	78	138
40	104	196
50	136	236
60	164	268
70	196	306
80	222	350
90	262	394
100	292	410

Table 2.2: Comparison between different kinds of queue implementation in the Value-Iteration procedure. All values are in seconds.

	FIFO Queue	LIFO Queue	LIFO Queue + Stopping-Criterion	Max-Priority Queue
μ	90.55	11.77	6.98	184.69
σ	487.69	64.10	34.61	653.26

In order to study the scalability of the algorithm with respect to the number of nodes, we had to rely on a second generator of random HyTN graphs. Our choice has been to use the `randomgame` procedure of `pgsolver` suite [98], that can produce parity games instances for any given number of nodes. In particular, we exploited `randomgame` in the following way:

1. First, `randomgame` was used to generate random directed graphs with out-degree fixed to 3;
2. Then, the resulting graphs were translated into MPGs by weighting each arc with an integer randomly chosen in the interval $[-W, W]$, where W was chosen accordingly to the test type;
3. Finally, each MPG G was translated into a HyTN $\mathcal{H}_G$ by the reduction algorithm of Theorem 2.8. During the translation from MPG to HyTN, only 10% of the hyperarcs were maintained having multiple heads, while 90% of hyperarcs were transformed into standard arcs. This 10%-rule stems from the fact that we are considering workflow based applications where the percentage of (multi-headed) hyperarcs is less than 10% compared to standard arcs in general.

With such settings, the resulting HyTNs are characterized by $m, m' \in \Theta(n)$.

Before presenting the summary of results, it is worthwhile to present some implementation choices about Algorithm 2 that we had to adopt. The core of the algorithm consists of calls to the procedure `makeACorrespondingGame(`$\mathcal{H}$`)`, that transforms the given HyTN $\mathcal{H}$ into a MPG $G_{\mathcal{H}}$, and to the procedure `solveMPG-Threshold(`$G_{\mathcal{H}}$`,0)` (Value Iteration algorithm), that determines for which game nodes s it holds that $v_s \geq 0$. The `makeACorrespondingGame()` implementation didn't require significant choices thanks to the simple structure of the algorithm.

On the contrary, in the implementation of `solveMPG-Threshold()` we introduced some further ideas in order to speed-up the algorithm and avoid unnecessary computations. Particularly, it is not necessary for the procedure `solveMPG-Threshold()` to continue to determine other potential value $v_{s'}$ as soon as it determines a value $v_s < 0$: at this point we can already conclude that the network is not consistent and, with a lower computational cost, we can yield a generalized negative circuit assessing this fact (Lemmas 2.4 and 2.5).

Moreover, we verified that there is an important data structure in the original Value Iteration algorithm, a queue, that is not further specified by the authors and that different implementations of it affect the performance of the algorithm. Therefore, we decided to verify whether `solveMPG-Threshold()` performance could be appreciably improved adding a suitable stopping criterion and a proper queue implementation. Table 2.2 reports the obtained results, mean execution time μ and its standard deviations σ, determined running the following different versions of `solveMPG-Threshold()` on the same data set of 10^3 not consistent HyTNs[4] having $|V| = 10^6$ and $W = 10^3$:

1. **FIFO Queue**: the original queue is implemented as a FIFO queue;
2. **LIFO Queue:** the original queue is implemented as a a LIFO queue (stack);
3. **LIFO Queue+Stopping Criterion:** the queue is implemented as stack and the computation is halted either when all potential values are stable or when any of them is negative;
4. **Max-Priority Queue:** the original queue is implemented as a Fibonacci's heap.

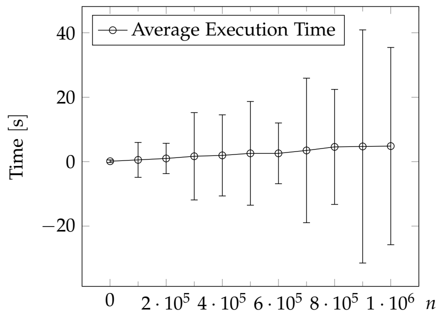
The results show that, in general, `solveMPG-Threshold()` performance becomes better if the original queue is implemented as a stack and, in particular, a further improvement can be obtained if the stopping criterion is also considered. Nevertheless, such improvements can only partially reduce the statistics variability of the running time, as it is shown in the following experimental results.

As mentioned above, the goal of Test 1 was to determine the average computation time of Algorithm 2 implementation for different values of n to study the practical computation time dependency on such parameter.

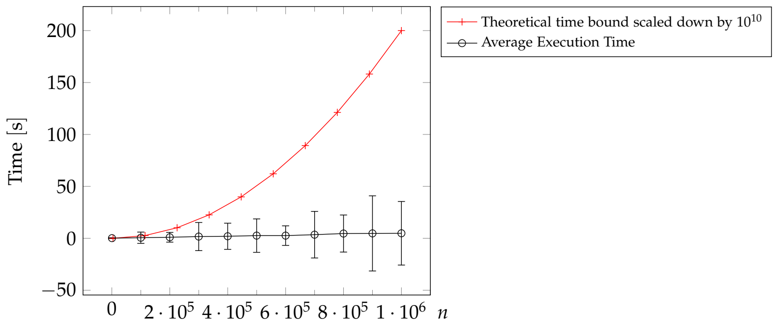
[4]We considered not consistent HyTNs because they practically required more time to be solved.

n	μ (sec)	σ
$< 4 \cdot 10^2$	0.13	0.42
$1 \cdot 10^5$	0.55	5.41
$2 \cdot 10^5$	0.99	4.71
$3 \cdot 10^5$	1.67	13.55
$4 \cdot 10^5$	1.95	12.59
$5 \cdot 10^5$	2.58	16.10
$6 \cdot 10^5$	2.58	9.43
$7 \cdot 10^5$	3.48	22.43
$8 \cdot 10^5$	4.58	17.85
$9 \cdot 10^5$	4.72	36.19
$10 \cdot 10^5$	4.83	30.62

(a) Test 1 results.



(b) Interpolation of average execution times of Test 1.



(c) Comparison between theoretical computation times and experimental ones.

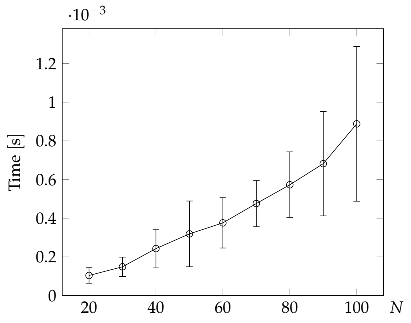
Figure 2.12: Results of Test 1: average execution times (μ) and relative standard deviations (σ) over a range of different HyTN orders n. Times are in seconds. Each data set comprised of 1600 HyTN instances of unknown consistency state.

The instances in Test 1 come from the `randomgame` generator, except those for the first row of the table in Fig. 2.12a which have been built by the ATAPIS workflow random generator. In particular, for each $n \in \{1 \cdot 10^5, 2 \cdot 10^5, \dots, 10 \cdot 10^5\}$, 1600 HyTN instances with maximum weight $W := 1000$ and unknown consistency state were generated by `randomgame`, whereas 1600 HyTNs of unknown consistency state and order n around 400 were generated by ATAPIS. The results of the test are summarized in Fig. 2.12, where each execution mean time is depicted as a point with a vertical bar representing its confidence interval determined according to its standard deviation.

The depicted function interpolating the mean values shows that the practical performance of the algorithms is definitely better than the theoretical worst-case bound of $O((n + m')mW)$; in our case this last is $O(n^2)$ since in the generated data sets W is constant and $m, m' \in \Theta(n)$. Fig. 2.12c depicts the interpolating function of experimental execution times and, in red, the func-

N	μ (sec)	σ
10	$6.42 \cdot 10^{-5}$	$1.22 \cdot 10^{-5}$
20	$1.05 \cdot 10^{-4}$	$4.85 \cdot 10^{-5}$
30	$1.50 \cdot 10^{-4}$	$5.7 \cdot 10^{-5}$
40	$2.43 \cdot 10^{-4}$	$1.04 \cdot 10^{-4}$
50	$3.20 \cdot 10^{-4}$	$1.78 \cdot 10^{-4}$
60	$3.77 \cdot 10^{-4}$	$1.38 \cdot 10^{-4}$
70	$4.77 \cdot 10^{-4}$	$1.28 \cdot 10^{-4}$
80	$5.73 \cdot 10^{-4}$	$1.80 \cdot 10^{-4}$
90	$6.82 \cdot 10^{-4}$	$2.79 \cdot 10^{-4}$
100	$8.89 \cdot 10^{-4}$	$4.10 \cdot 10^{-4}$

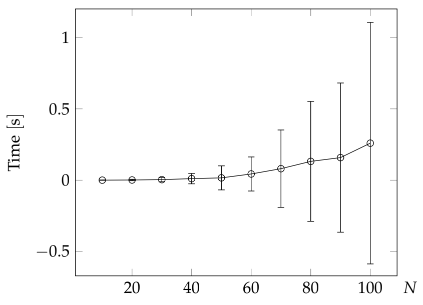
(a) Average execution times for consistent HyTNs obtained from workflow graphs with N activities.



(b) Interpolation of average execution times of Table 2.13a.

N	μ (sec)	σ
10	$4.45 \cdot 10^{-4}$	$1.38 \cdot 10^{-3}$
20	$1.50 \cdot 10^{-3}$	$5.10 \cdot 10^{-3}$
30	$4.04 \cdot 10^{-3}$	$1.48 \cdot 10^{-2}$
40	$1.10 \cdot 10^{-2}$	$3.62 \cdot 10^{-2}$
50	$1.64 \cdot 10^{-2}$	$8.42 \cdot 10^{-2}$
60	$4.36 \cdot 10^{-2}$	$1.20 \cdot 10^{-1}$
70	$8.08 \cdot 10^{-2}$	$2.71 \cdot 10^{-1}$
80	$1.31 \cdot 10^{-1}$	$4.20 \cdot 10^{-1}$
90	$1.59 \cdot 10^{-1}$	$5.22 \cdot 10^{-1}$
100	$2.59 \cdot 10^{-1}$	$8.46 \cdot 10^{-1}$

(c) Average execution times for not consistent HyTNs obtained from random workflow graphs with N activities.



(d) Interpolation of average execution times of Table 2.13c.

Figure 2.13: Average execution times obtained in Test 1 calculated considering samples of either all consistent or all not consistent HyTNs obtained from workflow graphs.

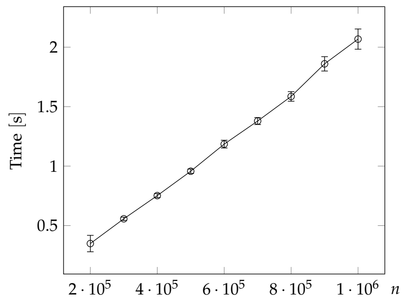
tion $n^2/10^{10}$ as a reasonable surrogate for the worst-case execution time. The comparison shows that the algorithm performs very well in real case executions.

However, since the standard deviation observed in the experiment is not negligible, we further investigated the behavior of the algorithm and we discovered that there is a correlation between the execution time of the algorithm and the consistency state of the input HyTN. Therefore, μ and σ were recalculated considering two kind of HyTN sets: one having all consistent HyTNs, and the other having all not consistent HyTNs.

Fig. 2.13 depicts average execution times obtained in Test 1 calculated considering samples of either all consistent or all not consistent HyTNs obtained from workflow graphs. Fig. 2.14 offers the same view but for HyTNs obtained from MPG graphs. In general, the mean execution times for consistent HyTNs

n	μ (sec)	σ
$1 \cdot 10^5$	0.16	0.04
$2 \cdot 10^5$	0.35	0.07
$3 \cdot 10^5$	0.56	0.01
$4 \cdot 10^5$	0.75	0.02
$5 \cdot 10^5$	0.96	0.02
$6 \cdot 10^5$	1.18	0.03
$7 \cdot 10^5$	1.38	0.03
$8 \cdot 10^5$	1.59	0.04
$9 \cdot 10^5$	1.86	0.06
$10 \cdot 10^5$	2.07	0.08

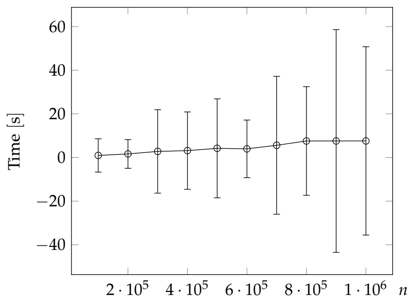
(a) Average execution times for consistent HyTNs obtained from MPGs.



(b) Interpolation of average execution times of Table 2.14a.

n	μ (sec)	σ
$1 \cdot 10^5$	0.95	7.63
$2 \cdot 10^5$	1.64	6.60
$3 \cdot 10^5$	2.79	19.11
$4 \cdot 10^5$	3.15	17.73
$5 \cdot 10^5$	4.21	22.67
$6 \cdot 10^5$	3.98	13.19
$7 \cdot 10^5$	5.60	31.58
$8 \cdot 10^5$	7.58	24.89
$9 \cdot 10^5$	7.58	51.03
$10 \cdot 10^5$	7.60	43.14

(c) Average execution times for not consistent HyTNs obtained from MPGs.



(d) Interpolation of average execution times of Table 2.14c.

Figure 2.14: Average execution times obtained in Test 1 calculated for samples of either all consistent or all not consistent HyTNs obtained from MPG graphs.

are smaller than the corresponding ones for not consistent HyTNs; furthermore, they also exhibit a negligible standard deviation. However, for samples of consistent HyTNs obtained from workflows, the standard deviation is not negligible even for samples with size $N = 20$. Part of the reasons for this behavior is given by the structure of the data sets: in each data set HyTNs can differ a lot with respect to their order and, therefore, they may require very different execution times. For example, the data set relating to workflow graphs with 20 activities contains HyTNs with order in range [48,94]. Since the number of activities is usually considered as main parameter in workflow community, we wanted to maintain such structure of data set and experiment results to emphasize the dependency of execution time with respect to such number.

On the other side, for consistent HyTNs determined from MPGs, the observed standard deviation σ is always less that the 10% of the average execution time μ with 99% level of confidence, while for not consistent HyTNs it has not been possible to determine any confidence level because the observed

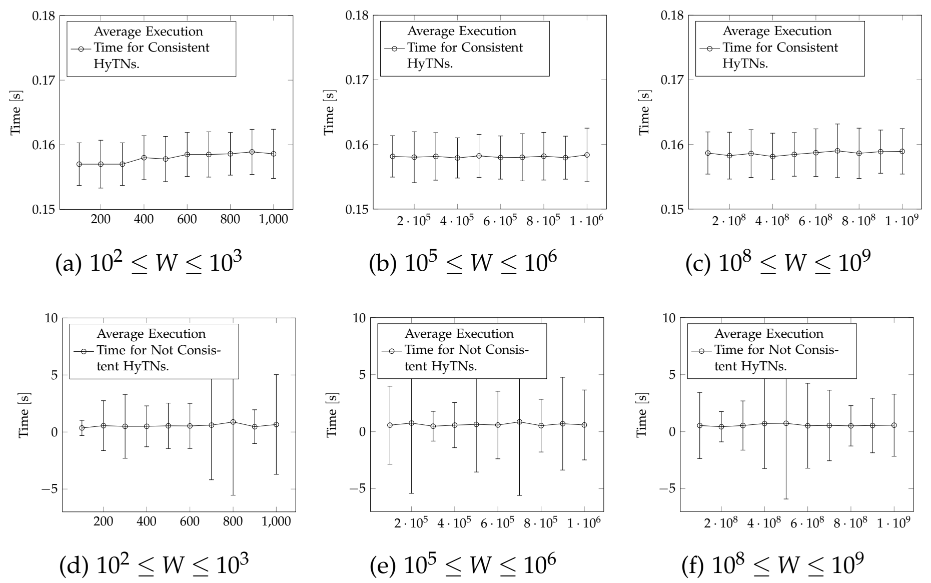
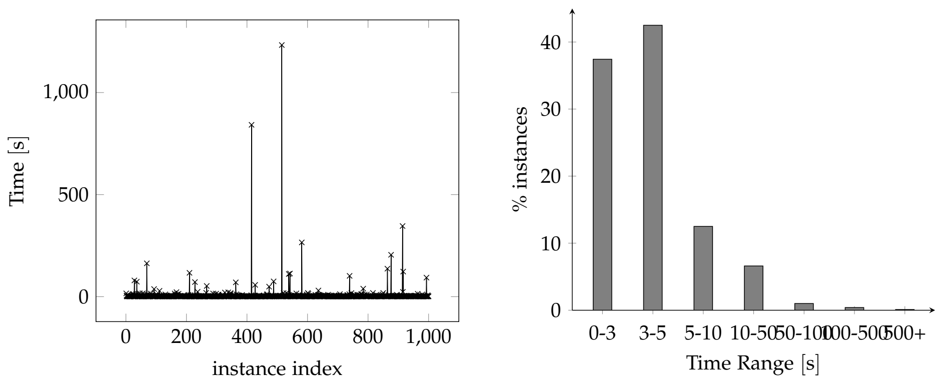


Figure 2.15: Average execution times in Test 1 calculated considering samples of either all consistent or all not consistent HyTNs.

standard deviation σ resulted to be always high due to some hard instances.

Even though procedure `solveMPG-Threshold()` could require up to $\Theta(W)$ updates according to the theoretical worst-case bound, our experiments suggest that, in practice, some kind of dependency of the running time on W is appreciable only for a few MPG games, all associated to not consistent HyTN instances.

The goal of Test 2 was to determine the average computation time of Algorithm 2 for different values of W, in order to understand how much the practical computation time is dependent on W. Therefore, we considered three possible edge weight ranges, $[10^2,10^3]$, $[10^5,10^6]$, and $[10^8,10^9]$, and for each of them two data sets have been built using the `randomgame` generator, one comprising only consistent HyTNs, and the other only not consistent ones. Each data set comprised of 800 HyTNs instances having $|V| = 10^5$ nodes, $m, m' \in \Theta(n)$ and edge weights in the corresponding weight range. Fig. 2.15 depicts the results on these six data sets. Applying the worst-case analysis for these data sets, it results that the time complexity should be $O(W)$ since n, m and m' are constants. On the contrary, the determined interpolation functions representing the experimental execution times do not show any clear dependence on W. This result suggests that, in practice, uniform random weighted instances are decided very quickly with respect to the magnitude of their weights and that the algorithm does not seem to exhibit the worst-case pseudo-polynomial behavior predicted in the asymptotic analysis. Moreover, the average execution times for each data set comprising only consistent



(a) Execution times of 10^3 not consistent HyTNs instances with $n = 10^6$ and $W = 10^3$.

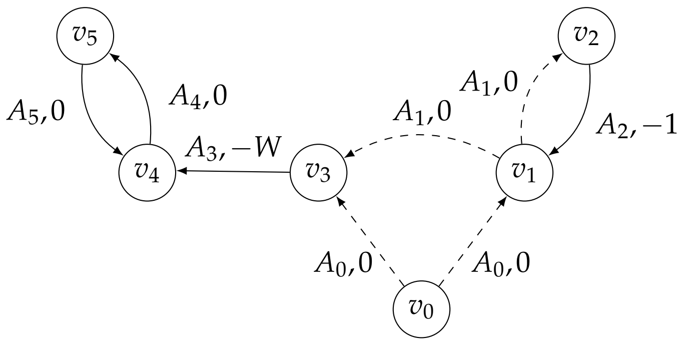
(b) Instance classification with respect to `solveMPG-Threshold()` execution time.

Figure 2.16: `solveMPG-Threshold()` execution times obtained in Test 3 determined considering samples of all not consistent HyTN instances.

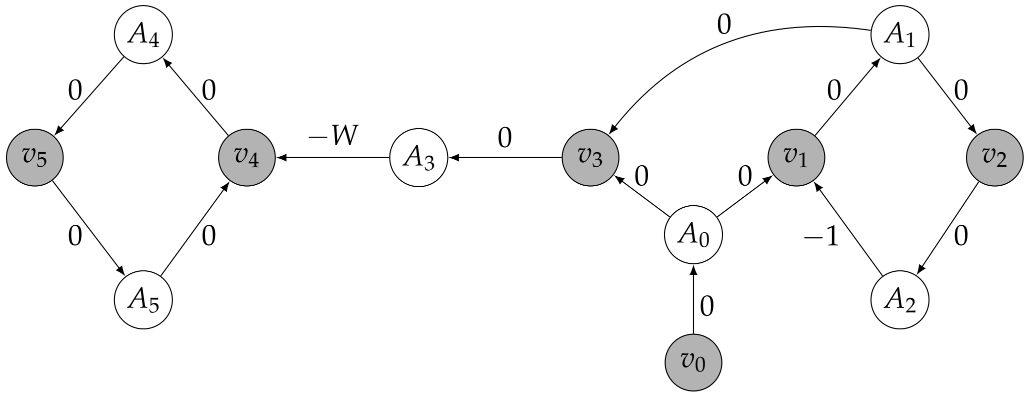
HyTNs are less than those for the corresponding data of only not consistent HyTNs. Only for consistent HyTNs data sets the standard deviation was below 7% than the average execution time with a confidence of 99%.

In order to better understand how some execution times affect the value of the standard deviation, we conducted a third experiment, Test 3, with the goal to visualize the distribution of the instances with computation times significantly above the average. Procedure `solveMPG-Threshold()` has been executed on 10^3 randomly generated not consistent HyTNs, each having order $n = 10^6$ and $W \approx 10^3$. The determined running times are depicted in Fig. 2.16a: most of the instances are decided very quickly, i.e., in a time between 0 and 10 seconds, while a smaller portion of the HyTNs required a time between 10 and 500 seconds. In more details, in repeated tests we verified that, approximately, 1% of the HyTN instances required a time between 50 and 100 seconds to be decided, 0.4% required a time between 100 and 500 seconds, and, finally, only 0.1% required more than 500 seconds. These results are shown in Fig. 2.16b.

Such behavior has been confirmed in other tests with different graph orders and maximum edge weight values. In several experiments we conducted, we observed that the maximum execution time of the algorithm keeps growing as we enlarge the size of the dataset. This explains why the standard deviation can't be reduced. If we could characterize such hard instances in general, we would be making a major progress in understanding the computational complexity of MPGs. We didn't find any pattern or property that characterizes the found hard instance. Here we can only show a simple family of HyTNs instances in which the execution time grows linearly with W. The family is given by just one single HyTN graph where only W changes, as depicted in Fig. 2.17a. The corresponding MPG is shown in Fig. 2.17b and provides a clear



(a) A HyTN $\mathcal{H}_W$ for which Algorithm 2 takes $\Theta(W)$ time.



(b) The corresponding MPG $G_{\mathcal{H}_W}$ computed by Algorithm 1.

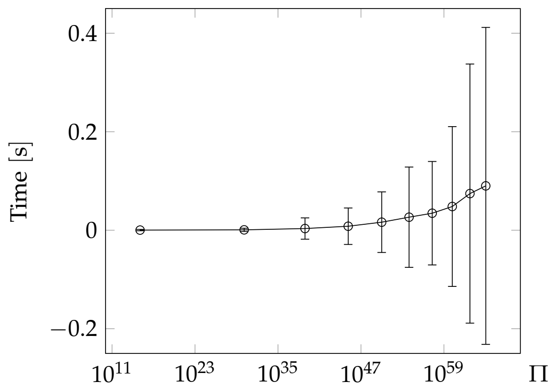
Figure 2.17: A HyTN which requires $\Theta(W)$ computation time by Algorithm 2.

example where Brim's Value Iteration algorithm [14] performs poorly. It is worth noting that in the context of MPGs this example can be reduced down to 6 nodes.

Finally, in order to show how much the running time is dependent on the number of different positional strategies of one player, in Test 4 the average computation time has been calculated with respect to different values of the product of the heads of hyperarcs (i.e., $\prod_{A\in\mathcal{A}}|H_A|$) in a HyTN. In particular, for each $\Pi \in \{10^{15}, 10^{30}, \ldots, 10^{65}\}$, 2500 HyTNs instances $(V, \mathcal{A})$ each having $|V| = 50$ nodes, $\prod_{A\in\mathcal{A}}|H_A| \approx \Pi$, and $W = 10^3$ have been generated by means of `randomgame` generator. The results of the evaluation are depicted in Fig. 2.18, where Π values are drawn in logarithmic scale. Analyzing the diagram in the figure it is possible to say that, experimentally, the average execution time increases only logarithmically with respect to the number of different positional strategies of one player. This results is quite interesting because, considering the HyTN in Fig. 2.17a, it is evident that the time for checking a HyTN is more dependent on the edge weight magnitude than on the number of different positional strategies of one player.

Π	μ (sec)	σ
$1.13 \cdot 10^{15}$	$3.02 \cdot 10^{-4}$	$1.36 \cdot 10^{-3}$
$1.27 \cdot 10^{30}$	$7.78 \cdot 10^{-4}$	$3.34 \cdot 10^{-3}$
$8.08 \cdot 10^{38}$	$3.45 \cdot 10^{-3}$	$2.15 \cdot 10^{-2}$
$1.43 \cdot 10^{45}$	$8.13 \cdot 10^{-3}$	$3.70 \cdot 10^{-2}$
$1.00 \cdot 10^{50}$	$1.63 \cdot 10^{-2}$	$6.15 \cdot 10^{-2}$
$9.10 \cdot 10^{53}$	$2.65 \cdot 10^{-2}$	$1.02 \cdot 10^{-1}$
$2.02 \cdot 10^{57}$	$3.46 \cdot 10^{-2}$	$1.05 \cdot 10^{-1}$
$1.61 \cdot 10^{60}$	$4.82 \cdot 10^{-2}$	$1.62 \cdot 10^{-1}$
$5.80 \cdot 10^{62}$	$7.44 \cdot 10^{-2}$	$2.63 \cdot 10^{-1}$
$1.13 \cdot 10^{65}$	$9.01 \cdot 10^{-2}$	$3.21 \cdot 10^{-1}$

(a) Average execution times obtained for different values of the product of the heads of hyperarcs Π.



(b) Interpolation of average execution times in Table 2.18a.

Figure 2.18: Average Execution Times obtained in Test 4.

2.8 Related Works

In the literature there are some extension proposals of the STN model to augment the capability to represent temporal constraints.

In the STN seminal paper [44], Dechter *et al.* firstly proposed to consider the Temporal Constraint Satisfaction Problem (TCSP). A binary constraint in a TCSP is represented using a set of intervals rather than a single interval as in an STN. In particular, a binary constraint $C_{ij} = \{[a_1,b_1],[a_2,b_2],\ldots,[a_l,b_l]\}$ between time points x_i and x_j represents the disjunction $a_1 \leq x_j - x_i \leq b_1 \vee a_2 \leq x_j - x_i \leq b_2 \vee a_l \leq x_j - x_i \leq b_l$. The problem of verifying consistency of a TCSP is NP-complete as the same authors showed in the paper; hence, they finally propose to consider STNs as a tractable simplified model.

A similar kind of generalization considering disjunction of temporal distance constraints was proposed by Stergiou and Koubarakis [108] defining the Disjunctive Temporal Problem (DTP). A DTP consists of a set of variables

$X = \{x_1, x_2, \ldots, x_n\}$ having continuous domains and representing time points and a set of disjunctive difference constraints between the time points in the form: $a_1 \leq x_{i_1} - x_{j_1} \leq b_1 \vee a_2 \leq x_{i_2} - x_{j_2} \leq b_2 \vee \ldots \vee a_k \leq x_{i_k} - x_{j_k} \leq b_k$; where $x_{i_1}, x_{j_1}, \ldots, x_{i_k}, x_{j_k}$ are time points from X and $a_1, b_1, \ldots, a_k, b_k$ are real numbers. A DTP is consistent if there exists an instantiation of variables X to real numbers satisfying all the constraints. Since DTPs are a generalization of TCSPs, also for DTPs the consistency check problem is NP-complete. In [108] the authors presented some of the theoretical results characterizing the possible backtracking algorithms that solve the consistency problem in terms of search nodes visited and consistency checks performed.

In 2005, Kumar proposed to consider a restricted class of DTP in order to maintain some of the expressive power of DTPs but, at the same time, allowing an efficient consistency check. In particular, in [102], RDTPs (restricted DTPs) is defined as a disjunctive temporal problem where a constraint is one of the following types: (Type 1) $(l \leq x_i - x_j \leq u)$, (Type 2) $(l_1 \leq x_i \leq u_1) \vee (l_2 \leq x_i \leq u_2) \ldots (l_j \leq x_i \leq u_j)$, (Type3) $(l_1 \leq x_i \leq u_1) \vee (l_2 \leq x_j \leq u_2)$, where x_i and x_j represent a timepoint variable, and l_i, u_i real values. An RDTP instance can be solved in strongly polynomial-time deterministic algorithm transforming it into a binary Constraint Satisfiability Problem (CSP) over meta variables representing constraints of Type 2 or Type 3 and, then, showing that such binary constraints are also *connected row-convex (CRC)* constraints, and, then, exploiting the properties of CRC constraints. An instantiation of a consistency check algorithm for RDTPs that further exploits the structure of CRC constraints has a running time complexity of $O((|TP_2| + |TP_3|)^3 d_{\max}^2 + (|TP_2| + |TP_3|)^2(NM + d_{\max}^2))$, where TP_2 is the set of Type 2 constraints, TP_3 is the set of Type 3 ones, $d_{\max}$ is the maximum number of disjuncts in any constraint, and N/M is the number of the nodes/arcs of the instance, respectively. In the same paper, Kumar presented also a simpler and faster, but *randomized*, algorithm for the same class RDTP.

An attempt to model some aspects of STNs similar to those addressed by HyTNs was lead in [2], where fun-in and fun-out subgraphs much resembling our multi-tail and multi-head hyperarcs were considered. However, since the problem 1-in-3-SAT is NP-complete even when all the literals comprising the clauses are positive, it readily follows that their models lead to NP-complete problems even when fun-out subgraphs (or fun-in subgraphs) are banned. As such, the opportunity for tractability spotlighted in this chapter is missed in those models.

Another approach to extend STN is represented by the proposal of Khatib *et al.* [71,72]. They introduced the characterization of *hard* and *soft* constraints. STNs are able to model just hard temporal constraints, i.e., they can represent instances where all constraints have to be satisfied, and that the solutions of a constraint are all equally satisfying. However, such assumption can be too much restrictive in some real-life scenarios. For example, it may be that some solutions are preferred with respect to others and, hence, the main problem is to find a way to satisfy them optimally, according to the preferences specified. To address these kind of problems, in [72] the authors introduced a framework in which each temporal constraint is associated with a preference function specifying the preference for each distance or duration; a *soft* simple temporal constraint is a 4-tuple $\langle (X,Y), I, A, f \rangle$ consisting of (1) an ordered pair of variables (X,Y) over the integers, called the scope of the constraint; (2) an interval $I = [a,b]$, where a and b are integers such that $a \leq b$; (3) a set of preferences A; (4) a preference function f, where $f : [a,b] \mapsto A$ is a mapping of the elements belonging to interval I into preference values, taken from set A. An assignment v_x and v_y to the variables X and Y is said to satisfy the constraint $\langle (X,Y), I, A, f \rangle$ if and only if $a \leq v_y - v_x$. In such a case, the preference associated to the assignment by the constraint is $f(v_y - v_x)$. Using soft simple temporal constraint, a new model of temporal constraint network has been introduce: the *Simple Temporal Problem with Preferences (STPP)*. In general, each solution of a STPP has a global preference value, obtained by combining in a suitable way the preference levels at which the solution satisfies the constraints. The optimal solutions of an STPP are those solutions which are not dominated by any other solution in terms of global preference. It was shown in [72] that, in general, STPPs belongs to the class of NP-hard problems. When the preference functions are semi-convex and some other side conditions are observed, then the problem to find an optimal solutions of an STPP is tractable [71].

Finally, another kind of possible extension is represented by the use of $\neq$ operator instead of $\leq$ in the binary constrains of STNs. Koubarakis [73] showed that if in a STN temporal constraints are used together with disequations in the form $x - y \neq r$, where r is a real constant, then the problem of deciding consistency is still tractable. This extension does not allow the specification of alternative constraints but it is interesting because it allows to exclude some solutions maintaining the consistency problem tractable.

2.9 Conclusion

In the literature, there are different frameworks and approaches aimed to extend the
STN model allowing the representation of disjunctive temporal constraints [44, 108], but at cost of an exponential-time consistency check procedure. The only extension with a polynomial time consistency check procedure we are aware of is the one of Kumar [102] mentioned in Section 2.8.

In this chapter, we proposed a novel extension, called Hyper Temporal Network (HyTN), where it is possible to represent a new kind of disjunctive constraint, hyper constraint, and to check the consistency of a network in pseudo-polynomial time. A hyper constraint is a suitable set of
STN distance constraints and it is satisfied if at least one distance constraint is satisfied. There could be two kinds of hyperarc: multi-head and multi-tail. In a multi-head hyperarc, its distance constraints are between a common source timepoint and different destination timepoints. In a multi-tail hyperarc, its distance constraints are between different source timepoints and a common

destination timepoint.

A HyTN is said consistent if it is possible to determine an assignment for all its timepoints such that all hyperarcs are satisfied. The computational complexity of the consistency problem of a HyTN is NP-complete when instances contain both kinds of hyperarc.

On instances containing either only multi-tail hyperarcs, or only multi-head hyperarcs, the consistency problem can be solved by reducing it, in a very efficient way, to the search of a winning strategy in an equivalent Mean Payoff Game (MPG), and exploiting the known winning-strategy search algorithms for MPGs.

Moreover, we presented an empirical analysis of the efficiency of the resulting consistency check algorithm. The empirical analysis shows that the proposed algorithm can be effectively used in real cases and confirms the general robustess of our approach.

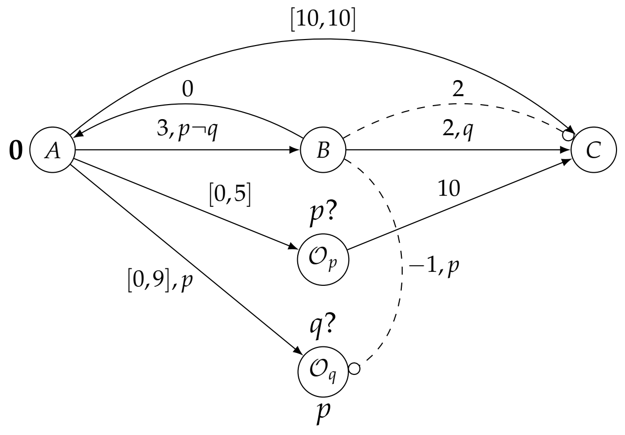
As future work we are investigating the frontier of practical efficient consistency checking for possible generalizations of the HyTN model as, for example, those including contingent constraints [117] or conditional ones [113].

3 Checking Dynamic Consistency of Conditional Hyper Temporal Networks via Mean Payoff Games

Chapter Abstract

Conditional Simple Temporal Network (CSTN) is a constraint-based graph-formalism for conditional temporal planning. It offers a more flexible formalism than the equivalent CSTP model of Tsamardinos, *et al.* [113], from which it was derived mainly as a sound formalization. Three notions of consistency arise for CSTNs: weak, strong, and dynamic. Dynamic consistency is the most interesting notion, but it is also the most challenging and it was conjectured to be hard to assess. Tsamardinos, *et al.* [113] gave a doubly-exponential time algorithm for checking dynamic consistency in CSTNs and to produce an exponentially sized dynamic execution strategy whenever the input CSTN is dynamically-consistent. CSTNs may be viewed as an extension of Simple Temporal Networks (STNs) [44], directed weighted graphs where nodes represent events to be scheduled in time and arcs represent temporal distance constraints between pairs of events. Recently, STNs have been generalized into *Hyper Temporal Networks* (HyTNs), by considering weighted directed hypergraphs where each hyperarc models a *disjunctive* temporal constraint named *hyperconstraint*; being directed, the hyperarcs can be either *multi-head* or *multi-tail*. The computational equivalence between checking consistency in HyTNs and determining winning regions in Mean Payoff Games (MPGs) was also pointed out; MPGs are a family of 2-player infinite pebble games played on finite graphs, which is well known for having applications in model-checking and formal verification. In this work we introduce the *Conditional Hyper Temporal Network (CHyTN)* model, a natural extension and generalization of both the CSTN and the HyTN model which is obtained by blending them together. We show that deciding whether a given CSTN or CHyTN is dynamically-consistent is coNP-hard; and that deciding whether a given CHyTN is dynamically-consistent is PSPACE-hard, provided that the input instances are allowed to include both multi-head and multi-tail hyperarcs. In light of this, we continue our study by focusing on CHyTNs that allow only multi-head hyperarcs, and we of-

fer the first deterministic (pseudo) singly-exponential time algorithm for the problem of checking the dynamic consistency of such CHyTNs, also producing a dynamic execution strategy whenever the input CHyTN is dynamically-consistent. Since CSTNs are a special case of CHyTNs, as a byproduct this provides the first sound-and-complete (pseudo) singly-exponential time algorithm for checking dynamic consistency in CSTNs. The proposed algorithm is based on a novel connection between CHyTNs and MPGs; due to the existence of efficient pseudo-polynomial time algorithms for MPGs, it is quite promising to be competitive in practice. The presentation of such connection is mediated by the HyTN model. In order to analyze the time complexity of the algorithm, we introduce a refined notion of dynamic consistency, named ϵ-dynamic consistency, and present a sharp lower bounding analysis on the critical value of the reaction time $\hat{\varepsilon}$ where a CHyTN transits from being, to not being, dynamically-consistent. The proof technique introduced in this analysis of $\hat{\varepsilon}$ is applicable more generally when dealing with linear difference constraints which include strict inequalities.



An example of a CHyTN.

This chapter is a revised version of [34,40].

3.1 Introduction and Motivation

In many areas of Artificial Intelligence (AI), including temporal planning and scheduling, the representation and management of quantitative temporal aspects is of crucial importance (see e.g., [7,25,26,48,95,107]). Examples of possible quantitative temporal aspects include constraints on the earliest start time and latest end time of activities and constraints over the minimum and maximum temporal distance between activities. In many cases these constraints can be represented by *Simple Temporal Networks* (STNs) [44], i.e., directed weighted graphs where nodes represent events to be scheduled in time and arcs represent temporal distance constraints between pairs of events. In Chapter 2, STNs have been generalized into *Hyper Temporal Networks* (HyTNs) [32, 33], a strict generalization of STNs introduced to overcome the limitation of considering only conjunctions of constraints, but maintaining a practical efficiency in the consistency checking of the instances. In a HyTN a single temporal hyperarc constraint is defined as a set of two or more maximum delay constraints which is satisfied when at least one of these delay constraints is satisfied. HyTNs are meant as a light generalization of STNs offering an interesting compromise. On one side, there exist practical pseudo-polynomial time algorithms for checking the consistency of HyTNs and computing feasible schedules for them. On the other side, HyTNs offer a more powerful model accommodating natural disjunctive constraints that cannot be expressed by STNs. In particular, HyTNs are weighted directed hypergraphs where each hyperarc models a disjunctive temporal constraint called *hyperconstraint*. The computational equivalence between checking consistency in HyTNs and determining winning regions in *Mean Payoff Games* (MPGs) [14, 49, 123] was also pointed out in [32, 33], where the approach was shown to be robust thanks to experimental evaluations (also see [15]). MPGs are a family of 2-player infinite pebble games played on finite graphs which is well known for having theoretical interest in computational complexity, being one of the few natural problems lying in $\mathsf{NP} \cap \mathsf{coNP}$, as well as various applications in model checking and formal verification [61].

However, in the representation of quantitative temporal aspects of systems, *conditional* temporal constraints pose a serious challenge for conditional temporal planning, where a planning agent has to determine whether a candidate plan will satisfy the specified conditional temporal constraints. This can be difficult, because the temporal assignments that satisfy the constraints associated with one conditional branch may fail to satisfy the constraints along a different branch (see, e.g., [113]). The present work unveils that HyTNs and MPGs are a natural underlying combinatorial model for checking the consistency of certain conditional temporal problems that are known in the literature and that are useful in some practical applications of temporal planning, especially, for managing the temporal aspects of Workflow Management Systems (WfMSs) [7,26] and for modeling Healthcare's Clinical Pathways [25]. Thus we focus on *Conditional Simple Temporal Networks (CSTNs)* [67, 113], a constraint-based model for conditional temporal planning. The CSTN formalism extends STNs in that: (1) some of the nodes are *observation events*, to each of them is associated a boolean variable whose value is disclosed only at execution time; (2) *labels* (i.e. conjunctions over the literals) are attached to all nodes *and* constraints, to indicate the situations in which each of them is required. The planning agent (or *Planner*) must schedule all the required nodes, meanwhile respecting all the required temporal constraints among them. This extended framework allows for the off-line construction of conditional plans that are guaranteed to satisfy complex networks of temporal constraints. Importantly, this can be achieved even while allowing for the decisions about the precise timing of actions to be postponed until execution time, in a least-commitment manner, thereby adding flexibility and making it possible to adapt the plan dynamically, in response to the observations that are made during execution. See [113] for further details and examples.

Three notions of consistency arise for CSTNs: weak, strong, and *dynamic*. Dynamic consistency (DC) is the most interesting one; it requires the existence of conditional plans where decisions about the precise timing of actions are postponed until execution time, but it nonetheless guarantees that all the relevant constraints will be ultimately satisfied. Still, it is the most challenging and it was conjectured to be hard to assess by [113]. Indeed, to the best of our knowledge, the tightest currently known upper bound on the time complexity of deciding whether a given CSTN is dynamically-consistent is doubly-exponential time [113]. It first builds an equivalent Disjunctive Temporal Problem (DTP) of size exponential in the input CSTN, and then applies to it an exponential-time DTP solver to check its consistency. However, this approach turns out to be quite limited in practice: experimental studies have already shown that the resolution procedures, as well as the currently known heuristics, for solving general DTPs become quite burdensome with 30 to 35 DTP variables (see e.g., [87, 93, 112]), thus dampening the practical applicability of the approach.

3.1.1 Contribution

In this work we introduce and study the *Conditional Hyper Temporal Network (CHyTN)* model, a natural extension and generalization of both the CSTN and the HyTN model which is obtained by blending them together. One motivation for studying it is to transpose benefits and opportunities for application, that have arisen from the introduction of HyTNs (see Chapter 2 and [32, 33]), to the context of *conditional* temporal planning. In so doing, the main and perhaps most important contribution is that to offer the first sound-and-complete deterministic (pseudo) singly-exponential time algorithm for checking the dynamic consistency of CSTNs. After having formally introduced the CHyTN model, we start by showing that deciding whether a given CSTN or CHyTN is dynamically-consistent is coNP-hard. Then, we offer a proof that deciding whether a given CHyTN is dynamically-consistent is PSPACE-hard, provided that the input CHyTN instances are allowed to include both multi-head and multi-tail hyperarcs. In light of this, we focus on CHyTNs that allow only multi-head hyperarcs. Concerning multi-head CHyTNs, perhaps most importantly, we unveil a connection between the problem of checking their dynamic consistency and that of determining winning regions in MPGs (of a singly-exponential size in the number of propositional variables of the input CHyTN), thus providing the first sound-and-complete (pseudo) singly-exponential time algorithm for this same task of deciding the dynamic consistency and yielding a dynamic execution strategy for multi-head CHyTNs. The resulting worst-case time complexity of the DC-Checking procedure is actually

$$O\big(2^{3|P|}|V||\mathcal{A}|m_{\mathcal{A}} + 2^{4|P|}|V|^2|\mathcal{A}||P| + 2^{4|P|}|V|^2 m_{\mathcal{A}} + 2^{5|P|}|V|^3|P|\big)W,$$

where $|P|$ is the number of propositional variables, $|V|$ is the number of event nodes, $|\mathcal{A}|$ is the number of hyperarcs, $m_{\mathcal{A}}$ is the size (i.e., roughly, the encoding length of $\mathcal{A}$), and W is the maximum absolute integer value of the weights of the input CHyTN. The algorithm is still based on representing a given CHyTN instance on an exponentially sized network, as first suggested in [113]. The difference, however, is that we propose to map CSTNs and CHyTNs on (exponentially sized) HyTNs/MPGs rather than on DTPs. This makes an important difference, because the consistency check for HyTNs can be reduced to determining winning regions in MPGs, as shown in [32, 33], which admits practical and effective pseudo-polynomial time algorithms (in some cases the algorithms for determining winning regions in MPGs exhibit even a strongly polynomial time behaviour, see e.g., [15, 19, 33, 121]). To summarize, we obtain an improved upper bound on the theoretical time complexity of the DC-checking of CSTNs (i.e., from 2-EXP to pseudo-$E \cap \text{NE} \cap \text{coNE}$) together with a faster DC-checking procedure, which can be used on CHyTNs with a larger number of propositional variables and event nodes than before. At the heart of the algorithm a suitable reduction to MPGs is mediated by the HyTN model, i.e., the algorithm decides whether a CHyTNs is dynamically-consistent by solving a carefully constructed MPG. In order to analyze the algorithm, we introduce a novel and refined notion of dynamic consistency, named ϵ-dynamic consistency (where $\epsilon \in \mathbb{R}_+$), and present a sharp lower bounding analysis on the critical value of the *reaction time* $\hat{\varepsilon}$ where a CHyTNs transits from being, to not being, dynamically-consistent. We believe that this contributes to clarifying (w.r.t. some previous literature, e.g., [67, 113]) the role played by the reaction time $\hat{\varepsilon}$ in checking the dynamic consistency of CSTNs. Moreover, the proof technique introduced in this analysis of $\hat{\varepsilon}$ is applicable more generally when dealing with linear difference constraints which include strict inequalities; thus it may be useful in the analysis of other models of temporal constraints.

A preliminary version of this chapter appeared in the proceedings of the TIME symposium [34]. Here, the presentation is extended as follows: (1) the definition of CSTN has been extended and generalized to that of CHyTN in order to allow the presence of hyperarcs as labeled temporal constraints already in the input instances; (2) some further facts and pertinent properties about

CSTNs and CHyTNs have been established; (3) for instance, the following hardness result: deciding whether a given CHyTN is dynamically-consistent is PSPACE-hard (the reduction goes from 3-CNF-TQBF), provided that the input instances are allowed to include both multi-head and multi-tail hyperarcs; (4) the proposed (pseudo) singly-exponential time algorithm is presented here in its full generality, i.e., w.r.t. the CHyTN model; (5) several proofs have been polished, expanded and clarified (e.g., those concerning the reaction time analysis of $\hat{\epsilon}$).

3.1.2 Organization

The rest of the chapter is organized as follows. Section 4.2 recalls the basic formalism, terminology and known results on STNs and HyTNs. Particularly, Subsection 3.2.1 deals with STNs; Subsection 3.2.2 deals with HyTNs, its computational equivalence with MPGs and the related algorithmic results. Section 3.3 surveys CSTNs and, then, it introduces CHyTNs, also presenting some basic properties of the model. Section 4.3 tackles on the algorithmics of dynamic consistency: firstly, we provide a coNP-hardness lower bound, then we offer a PSPACE-hardness lower bound. Next, it is described the connection with HyTNs/MPGs and it is devised a (pseudo) singly-exponential time DC-checking algorithm. Section 3.5 offers a sharp lower bounding analysis on the critical value of the *reaction time* $\hat{\varepsilon}$ where the CSTN transits from being, to not being, dynamically-consistent. In Section 4.4, related works are discussed. The chapter concludes in Section 2.9.

3.2 Background and Notation

This section recalls some background notions concerning STNs and HyTNs, necessary to follow the rest of the treatise. but the reader is referred back to Chapter 2 for some of those concepts.

3.2.1 Simple Temporal Networks

The reader is referred to Subsection 1.2.2, Chapter 1, for the basic notions and notation concerning STNs.

In this chapter, we also deal with directed weighted *hypergraphs*; see Definition 2.1 in Chapter 2. Also, recall that the *cardinality* of a hyperarc $A \in \mathcal{A}$ is given by $|A| \triangleq |H_A \cup \{t_A\}|$ if A is multi-head, and $|A| \triangleq |T_A \cup \{h_A\}|$ if A is multi-tail; if $|A| = 2$, then $A = (u, v, w)$ is a standard arc. The *order* and *size* of a general hypergraph $(V, \mathcal{A})$ are denoted by $n \triangleq |V|$ and $m_{\mathcal{A}} \triangleq \sum_{A \in \mathcal{A}} |A|$, respectively.

3.2.2 Hyper Temporal Networks

This subsection surveys the *Hyper Temporal Network* (HyTN) model, which is a strict generalization of STNs, introduced to partially overcome the limitation of allowing only conjunctions of constraints. HyTNs have been introduced in [32, 33], the reader is referred there for an in-depth treatment of the subject. Compared to STN distance graphs, which they naturally extend, HyTNs allow for a greater flexibility in the definition of the temporal constraints.

A general HyTN is a directed weighted general hypergraph $\mathcal{H} = (V, \mathcal{A})$ where a node represents a time-point variable (or event node), and where a multi-head/multi-tail hyperarc stands for a set of temporal distance constraints between the tail/heads and the head/tails (respectively). Also, we shall consider two special cases of the general HyTN model, one in which all hyperarcs are only multi-head, and one where they're only multi-tail. In general, we say that a hyperarc is *satisfied* when at least one of its distance constraints is satisfied. Then, we say that a HyTN is *consistent* when it is possible to assign a value to each time-point variable so that all of its hyperarcs are satisfied.

More formally, in the HyTN framework the consistency problem is defined as the General-HyTN-Consistency decision problem, see Definition 2.2 in Chapter 2. Comparing the consistency of HyTNs with the consistency of STNs, the most important aspect of novelty is that, while in a distance graph of a STN each arc represents a distance constraint and all such constraints have to be satisfied by any feasible schedule, in a HyTN each hyperarc represents a disjunction of one or more distance constraints and a feasible schedule has to satisfy at least one of such distance constraints for each hyperarc.

The reader is referred back to Chapter 2 for recalling interesting properties about the consistency problem for HyTNs, e.g., integrality (Lemma 2.1), NP-hardness (Theorem 2.1), the existence of negative cycle certificates (Definition 2.5 and Theorem 2.3), pseudo-polynomial time algorithms (Theorem 2.7), *etc*

In the rest of this work we shall adopt the multi-head hypergraph and the Head-HyTN-Consistency (Definition 2.3) consistency problem as our reference model; but we will consider general hypergraphs again in the forthcoming sections, when proving PSPACE-hardness. Let's say that, when considering hypergraphs and HyTNs, we will be implicitly referring to the multi-head variant unless it is explicitly specified otherwise.

In the forthcoming section we shall turn our attention to *conditional* temporal planning, where we generalize Conditional Simple Temporal Networks (CSTNs) by introducing Conditional Hyper Temporal Networks (CHyTNs).

3.3 Conditional Simple / Hyper Temporal Networks

In order to provide a formal support to the present work, this section recalls the basic formalism, terminology and known results on CSTPs and CSTNs. Since the forthcoming definitions concerning CSTNs are mostly inherited from the literature, the reader is referred to [113] and [67] for an intuitive semantic discussion and for some clarifying examples of the very same CSTN model. [113] introduced the *Conditional Simple Temporal Problem (CSTP)* as an extension of standard temporal constraint-satisfaction models used in non-conditional temporal planning. CSTPs augment STNs by including *observation* events, each one having a *boolean variable* (or *proposition*) associated with it. When an observation event is executed, the truth-value of its associated proposition becomes known. In addition, each event node and each constraint has a *label* that restricts the scenarios in which it plays a role. Although not included in the formal definition, [113] discussed some supplementary assumptions that any well-defined CSTP must satisfy. Subsequently, those conditions have been further analyzed and formalized by [67], leading to the definition of *Conditional Simple Temporal Network* (CSTN), which is now recalled.

Let P be a set of boolean variables, a *label* is any (possibly empty) conjunction of variables, or negations of variables, drawn from P. The *empty label* is denoted by λ. The *label universe* P^* is the set of all (possibly empty) labels whose (positive or negative) literals are drawn from P. Two labels, ℓ_1 and ℓ_2, are called *consistent*, denoted[1] by $Con(\ell_1, \ell_2)$, when $\ell_1 \wedge \ell_2$ is satisfiable. A label ℓ_1 *subsumes* a label ℓ_2, denoted[1] by $Sub(\ell_1, \ell_2)$, when the implication $\ell_1 \Rightarrow \ell_2$ holds. Let us recall the formal definition of the CSTN model from [67,113].

Definition 3.1 (CSTNs). *A* Conditional Simple Temporal Network (CSTN) *is a tuple* $\langle V, A, L, \mathcal{O}, \mathcal{O}V, P\rangle$ *where:*

- V *is a finite set of* events*;* $P = \{p_1, \ldots, p_q\}$ *(some* $q \in \mathbb{N}$*) is a finite set of* boolean variables *(or* propositions*);*
- A *is a set of* labeled temporal constraints (LTCs) *each having the form* $\langle v - u \leq w(u,v), \ell\rangle$*, where* $u, v \in V$*,* $w(u,v) \in \mathbb{R}$*, and* $\ell \in P^*$*;*
- $L : V \to P^*$ *is a map that assigns a label to each event node in* V*;* $\mathcal{O}V \subseteq V$ *is a finite set of* observation events*;* $\mathcal{O} : P \to \mathcal{O}V$ *is a bijection mapping a unique observation event* $\mathcal{O}(p) = \mathcal{O}_p$ *to each* $p \in P$*;*
- *The following* well definedness assumptions *must hold:*

  (WD1*) for any labeled constraint* $\langle v - u \leq w, \ell\rangle \in A$ *the label* ℓ *is satisfiable and subsumes both* $L(u)$ *and* $L(v)$*; i.e., whenever a constraint* $v - u \leq w$ *is required to be satisfied, both of its endpoints* u *and* v *must be scheduled (sooner or later) by the Planner;*

  (WD2*) for each* $p \in P$ *and each* $u \in V$ *such that either* p *or* $\neg p$ *appears in* $L(u)$*, we require:* $\mathrm{Sub}(L(u), L(\mathcal{O}_p))$*, and* $\langle \mathcal{O}_p - u \leq -\epsilon, L(u)\rangle \in A$ *for some (small) real* $\epsilon > 0$*; i.e., whenever a label* $L(u)$ *of an event node* u *contains a proposition* p*, and* u *gets eventually scheduled, the observation event* $\mathcal{O}_p$ *must have been scheduled strictly before* u *by the Planner.*

  (WD3*) for each labeled constraint* $\langle v - u \leq w, \ell\rangle$ *and* $p \in P$*, for which either* p *or* $\neg p$ *appears in* ℓ*, it holds that* $\mathrm{Sub}(\ell, L(\mathcal{O}_p))$*; i.e., assuming a required constraint contains proposition* p*, the observation event* $\mathcal{O}_p$ *must be scheduled (sooner or later) by the Planner.*

We are now in the position to introduce the *Conditional Hyper Temporal Network (CHyTN)*, a natural extension and generalization of both the CSTN and

[1]The notation $Con(\cdot,\cdot)$ and $Sub(\cdot,\cdot)$ is inherited from [67,113].

the HyTN model obtained by blending them together. Even though the original STN and CSTN models allow for real weights, hereafter we shall restrict ourselves to the integers in order to rely on Theorem 2.7. All of our CSTNs and CHyTNs will be integer weighted from now on.

Definition 3.2 (CHyTNs). *A general* Conditional Hyper Temporal Network (CHyTN) *is a tuple $\langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{OV}, P\rangle$, where $V, P, L, \mathcal{O}$ and $\mathcal{OV}$ are defined as in CSTNs (see Definition 3.1), and where $\mathcal{A}$ is a set of* labeled temporal hyper constraints (LTHCs)*, each having one of the following forms:*

- *$A = (t, h, w, \ell)$, where (t, h, w) is a standard arc and $\ell \in P^*$; in this case, A is called a* standard *LTHC.*
- *$A = (t_A, H_A, w_A, L_{H_A})$, where (t_A, H_A, w_A) is a* multi-head *hyperarc and $L_{H_A} : H_A \to P^*$ is a map sending each head $h \in H_A$ to a label ℓ_h in P^*; in this case, A is called a* multi-head *LTHC.*
- *$A = (T_A, h_A, w_A, L_{T_A})$, where $A = (T_A, h_A, w_A)$ is a* multi-tail *hyperarc and $L_{T_A} : T_A \to P^*$ is a map sending each tail $t \in T_A$ to a label ℓ_t in P^*; in this case, A is called a* multi-tail *LTHC.*
- *The following* well definedness assumptions *must hold:*

  *(*WD1′*) for any labeled constraint A:*

  - *if $A = (t, h, w, \ell)$ is a standard LTHC, the label ℓ is satisfiable and subsumes both $L(t)$ and $L(h)$;*
  - *if $A = (t_A, H_A, w_A, L_{H_A})$ is a multi-head LTHC, for each $h \in H_A$ the label $L_{H_A}(h)$ is satisfiable and subsumes both $L(t_A)$ and $L(h)$;*
  - *if $A = (T_A, h_A, w_A, L_{T_A})$ is a multi-tail LTHC, for each $t \in T_A$ the label $L_{T_A}(t)$ is satisfiable and subsumes both $L(h_A)$ and $L(t)$;*

  *(*WD2*) for each $p \in P$ and each $u \in V$ such that either p or $\neg p$ appears in $L(u)$, we require:* Sub$(L(u), L(\mathcal{O}_p))$*, and $\langle \mathcal{O}_p - u \leq -\epsilon, L(u)\rangle \in \mathcal{A}$ for some (small) real $\epsilon > 0$; this is the same WD2 as defined for CSTNs.*

  *(*WD3′*) for each labeled constraint $A \in \mathcal{A}$ and boolean variable $p \in P$:*

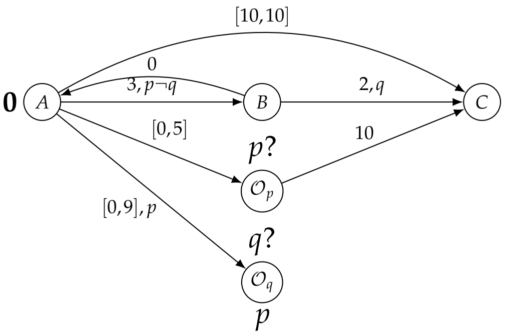
  - *if $A = (t, h, w, \ell)$ is a standard LTHC and p or $\neg p$ appears in ℓ, then* Sub$(\ell, L(\mathcal{O}_p))$*;*
  - *if $A = (t_A, H_A, w_A, L_{H_A})$ is a multi-head LTHC and either p or $\neg p$ appears in $L_{H_A}(h)$ for some $h \in H_A$, then* Sub$(L_{H_A}(h), L(\mathcal{O}_p))$*;*
  - *if $A = (T_A, h_A, w_A, L_{T_A})$ is a multi-tail LTHC and either p or $\neg p$ appears in $L_{T_A}(t)$ for some $t \in T_A$, then* Sub$(L_{T_A}(t), L(\mathcal{O}_p))$*;*

Of course every CSTN is a CHyTN (i.e., one having only standard LTHCs). We shall adopt the notation $x \xrightarrow{[a,b],\ell} y$, where $x, y \in V$, $a, b \in \mathbb{N}, a < b$ and $\ell \in P^*$, to compactly represent the pair $\langle y - x \leq b, \ell\rangle, \langle x - y \leq -a, \ell\rangle \in A$; also, whenever $\ell = \lambda$, we shall omit ℓ from the graphics, see e.g., Fig. 3.1a and Fig. 3.1b here below.

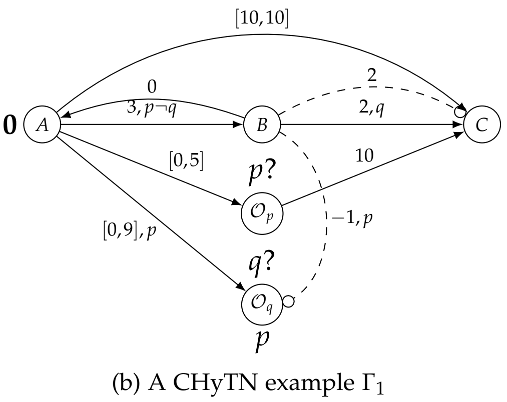
Example 3.1. *Fig. 3.1a depicts an example CSTN $\Gamma_0 = \langle V, A, L, \mathcal{O}, \mathcal{O}V, P\rangle$ having three event nodes A, B and C as well as two observation events $\mathcal{O}_p$ and $\mathcal{O}_q$. Formally, $V = \{A, B, C, \mathcal{O}_p, \mathcal{O}_q\}$, $P = \{p, q\}$, $\mathcal{O}V = \{\mathcal{O}_p, \mathcal{O}_q\}$, $L(v) = \lambda$ for every $v \in V \setminus \{\mathcal{O}_q\}$ and $L(\mathcal{O}_q) = p$, $\mathcal{O}(p) = \mathcal{O}_p, \mathcal{O}(q) = \mathcal{O}_q$. Next, the set of LTCs is: $A = \{\langle C - A \leq 10, \lambda\rangle, \langle A - C \leq -10, \lambda\rangle, \langle B - A \leq 3, p \wedge \neg q\rangle, \langle A - B \leq 0, \lambda\rangle, \langle \mathcal{O}_p - A \leq 5, \lambda\rangle, \langle A - \mathcal{O}_p \leq 0, \lambda\rangle, \langle \mathcal{O}_q - A \leq 9, p\rangle, \langle A - \mathcal{O}_q \leq 0, p\rangle, \langle C - B \leq 2, q\rangle, \langle C - \mathcal{O}_p \leq 10, \lambda\rangle$.*

Fig. 3.1b depicts an example of a multi-head CHyTN $\Gamma_1 = \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P\rangle$. Notice that $V, L, \mathcal{O}, \mathcal{O}V$ and P are the same as in the CSTN Γ_0, whereas $\mathcal{A}$ is defined as follows: $\mathcal{A} = A \cup \{\alpha \triangleq (B, \{C, \mathcal{O}_q\}, \langle w_\alpha(C), w_\alpha(\mathcal{O}_q)\rangle = \langle 2, -1\rangle, \langle L_\alpha(C), L_\alpha(\mathcal{O}_q)\rangle = \langle \lambda, p\rangle)\}$, where A is the set of LTCs of the CSTN Γ_0 and the additional constraint α is a multi-head LTHC with tail $t_\alpha = B$ and heads $H_\alpha = \{C, \mathcal{O}_q\}$.

Sometimes we will show the scheduling time of a node with a label in boldface on the sidelines of the node itself, as for A in Fig. 3.1.



(a) A CSTN example Γ_0.



(b) A CHyTN example Γ_1

Figure 3.1: An example CSTN (a), and an example CHyTN (b).

In the following definitions we will implicitly refer to some CHyTN which is denoted by $\Gamma = \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P\rangle$.

Definition 3.3 (Scenario [67, 113]). *A* scenario *over a subset $U \subseteq P$ of boolean variables is a truth assignment $s : U \to \{0,1\}$, i.e., s is a function that assigns a truth value to each proposition $p \in U$. When $U \subsetneq P$ and $s : U \to \{0,1\}$, then s is said to be a* partial *scenario; otherwise, when $U = P$, then s is said to be a* (complete) *scenario. The set comprising all of the complete scenarios over P is denoted by Σ_P. If $s \in \Sigma_P$ is a scenario and $\ell \in P^*$ is a label, then $s(\ell) \in \{0,1\}$ denotes the truth value of ℓ induced by s in the natural way.*

Notice that any scenario $s \in \Sigma_P$ can be described by means of the label $\ell_s \triangleq l_1 \wedge \cdots \wedge l_{|P|}$ such that, for every $1 \le i \le |P|$, the literal $l_i \in \{p_i, \neg p_i\}$ satisfies $s(l_i) = 1$.

Example 3.2. *Consider the set of boolean variables $P = \{p, q\}$. The scenario $s : P \to \{0,1\}$ defined as $s(p) = 1$ and $s(q) = 0$ can be compactly described by the label $\ell_s = p \wedge \neg q$.*

Definition 3.4 (Schedule [67, 113]). *A* schedule *for a subset of events $U \subseteq V$ is a map $\phi : U \to \mathbb{R}$ that assigns a real number to each event node in U. The set of all schedules over U is denoted by Φ_U.*

Definition 3.5 (Scenario Restriction). *Let $s \in \Sigma_P$ be a scenario. The* restriction *of V and $\mathcal{A}$ w.r.t. s are defined as:*

$$V_s^+ \triangleq \Big\{ v \in V \mid s(L(v)) = \top \Big\};$$

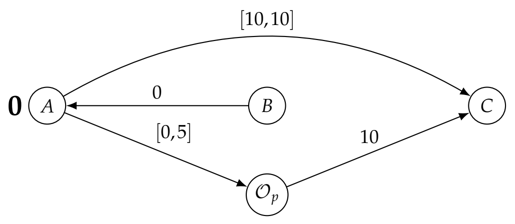
$$\begin{aligned}
\mathcal{A}_s^+ \triangleq \Big\{ & (u,v,w) \mid \exists (\ell \in P^*) \text{ s.t. } (u,v,w,\ell) \in \mathcal{A} \text{ and } s(\ell) = \top \Big\} \cup \\
\cup \Big\{ & (t, H'_A, w'_A) \mid \exists (H_A \supseteq H'_A; L_{H_A} : H_A \to P^*; w_A : H_A \to \mathbb{Z}) \text{ s.t. } (t, H_A, w_A, L_{H_A}) \in \mathcal{A}, \\
& \quad w'_A = w_{A|_{H'_A}}, \forall (h \in H_A) s(L_{H_A}(h)) = \top \iff h \in H'_A \Big\} \cup \\
\cup \Big\{ & (T'_A, h, w'_A) \mid \exists (T_A \supseteq T'_A; L_{T_A} : T_A \to P^*; w_A : T_A \to \mathbb{Z}) \text{ s.t. } (T_A, h, w_A, L_{T_A}) \in \mathcal{A}, \\
& \quad w'_A = w_{A|_{T'_A}}, \forall (t \in T_A) s(L_{T_A}(t)) = \top \iff t \in T'_A \Big\}.
\end{aligned}$$

The restriction of Γ w.r.t. s is defined as $\Gamma_s^+ \triangleq \langle V_s^+, \mathcal{A}_s^+ \rangle$.
Finally, it is worthwhile to introduce the notation $V_{s_1,s_2}^+ \triangleq V_{s_1}^+ \cap V_{s_2}^+$.

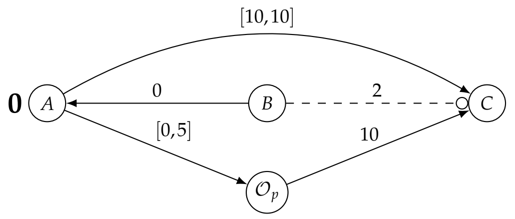
Note that if Γ is a CHyTN, then Γ_s^+ is a HyTN; and if Γ is a CSTN, then Γ_s^+ is an STN.

Example 3.3. *Fig. 3.2 depicts the restriction STN ${\Gamma_0}_s^+$ of the CSTN Γ_0, and the restriction HyTN ${\Gamma_1}_s^+$ of the CHyTN Γ_1 (see Example 3.1 and Fig. 3.1), w.r.t. the scenario $s(p) = s(q) = \bot$.*

Definition 3.6 (Execution-Strategy [67, 113]). *An* Execution-Strategy (ES) *for Γ is a mapping $\sigma : \Sigma_P \to \Phi_{V_s^+}$ such that, for any scenario $s \in \Sigma_P$, the domain of the schedule $\sigma(s)$ is V_s^+. The set of ESs of Γ is denoted by $\mathcal{S}_\Gamma$. The* execution time *of an event $v \in V_s^+$ in the schedule $\sigma(s) \in \Phi_{V_s^+}$ is denoted by $[\sigma(s)]_v$.*



(a) The restriction STN ${\Gamma_0}^+_s$ of the CSTN Γ_0 w.r.t. $s(p) = s(q) = \bot$



(b) The restriction HyTN ${\Gamma_1}^+_s$ of the CHyTN Γ_1 w.r.t. $s(p) = s(q) = \bot$

Figure 3.2: The restriction ${\Gamma_0}^+_s$ (a), and the restriction ${\Gamma_1}^+_s$ (b), w.r.t. the scenario $s(p) = s(q) = \bot$

Definition 3.7 (History [67,113]). *Let $\sigma \in \mathcal{S}_\Gamma$ be any ES, let $s \in \Sigma_P$ be any scenario and let $\tau \in \mathbb{R}$. The* history $\mathrm{Hst}(\tau,s,\sigma)$ *of τ in the scenario s under strategy σ is defined as:* $\mathrm{Hst}(\tau,s,\sigma) \triangleq \big\{\big(p,s(p)\big) \in P \times \{0,1\} \mid \mathcal{O}_p \in V_s^+, [\sigma(s)]_{\mathcal{O}_p} < \tau\big\}$.

The scenario history can be compactly expressed by the conjunction of the literals corresponding to the observations comprising it; thus, we may treat a scenario history as though it were a label.

Definition 3.8 (Viable Execution Strategy [67, 113]). *We say that $\sigma \in \mathcal{S}_\Gamma$ is a* viable *execution strategy whenever, for each scenario $s \in \Sigma_P$, the schedule $\sigma(s) \in \Phi_V$ is feasible for the restriction HyTN (or STN) Γ_s^+.*

Definition 3.9 (Dynamic-Consistency [67,113]). *An ES $\sigma \in \mathcal{S}_\Gamma$ is called* dynamic *if, for any $s_1,s_2 \in \Sigma_P$ and any $v \in V^+_{s_1,s_2}$, the following implication holds on $\tau \triangleq [\sigma(s_1)]_v$:*

$$\mathrm{Con}(\mathrm{Hst}(\tau,s_1,\sigma),s_2) \Rightarrow [\sigma(s_2)]_v = \tau.$$

We say that Γ is dynamically-consistent (DC) *if it admits $\sigma \in \mathcal{S}_\Gamma$ which is both viable and dynamic. The problem of checking whether a given CSTN is DC is named* DC-*Checking.*

Definition 3.10 (DC-Checking [67,113]). *The problem of checking whether a given CHyTN (which allows* both *multi-head and multi-tail LTHCs) is dynamically-consistent is named* General-CHyTN-DC*.*

That of checking whether a given CHyTN, allowing only *multi-head* or only *multi-tail LTHCs, is dynamically-consistent is named* CHyTN-DC*. Checking whether a given CSTN is dynamically-consistent is named* DC*.*

Example 3.4. *Consider the CHyTN Γ_1 of Fig. 3.1b, and let the scenarios s_1,s_2,s_3,s_4 be defined as: $s_1(p) = \top$, $s_1(q) = \top$; $s_2(p) = \top$, $s_2(q) = \bot$; $s_3(p) = \bot$, $s_3(q) = \top$; $s_4(p) = \bot$, $s_4(q) = \bot$. The following defines an execution strategy $\sigma \in \mathcal{S}_\Gamma$: $[\sigma(s_i)]_A = 0$ for every $i \in \{1,2,3,4\}$; $[\sigma(s_i)]_B = 8$ for every $i \in \{1,3,4\}$ and $[\sigma(s_2)]_B = 3$; $[\sigma(s_i)]_C = 10$ for every $i \in \{1,2,3,4\}$; $[\sigma(s_i)]_{\mathcal{O}_p} = 1$ for every $i \in \{1,2,3,4\}$. The reader can check that σ is viable and dynamic. Indeed, σ admits the tree-like representation depicted in Fig 3.3.*

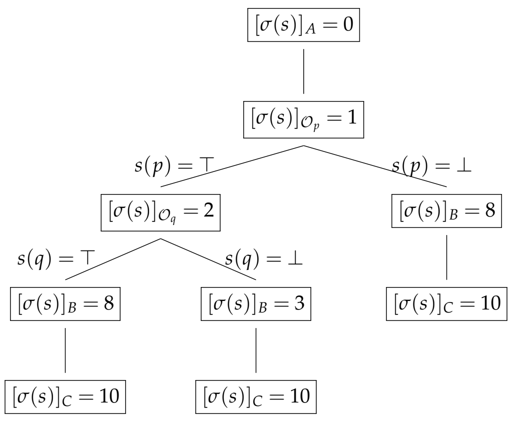


Figure 3.3: A tree-like representation of a dynamic execution strategy σ for the CHyTN Γ_1 of Fig. 3.1b, where s denotes scenarios and $[\sigma(s)]_X$ is the corresponding schedule.

Next, we recall a crucial notion for studying the dynamic consistency of CHyTNs: the *difference set* $\Delta(s_1;s_2)$.

Definition 3.11 (Difference-Set [67, 113]). *Let $s_1,s_2 \in \Sigma_P$ be any two scenarios. The set of observation events in $\mathcal{O}V^+_{s_1}$ at which s_1 and s_2 differ is denoted by $\Delta(s_1;s_2)$. Formally,*

$$\Delta(s_1;s_2) \triangleq \{\mathcal{O}_p \in \mathcal{O}V^+_{s_1} \mid s_1(p) \neq s_2(p)\}.$$

The various definitions of history and dynamic consistency that are used by different authors [34,69,113] are equivalent. Notice that commutativity may not hold (i.e., generally it may be the case that $\Delta(s_1;s_2) \neq \Delta(s_2;s_1)$).

Example 3.5. *Consider the CSTN Γ_0 of Fig. 3.1a and the scenarios s_1,s_2 defined as follows: $s_1 \triangleq p \wedge q$; $s_2 \triangleq \neg p \wedge \neg q$.*
Then, $\Delta(s_1;s_2) = \{\mathcal{O}_p,\mathcal{O}_q\}$ and $\Delta(s_2;s_1) = \{\mathcal{O}_p\}$.

The next lemma will be useful later on in Section 4.3, basically it is due to [113]; here below we propose a full proof for the sake of completeness.

Lemma 3.1. *Let $s_1,s_2 \in \Sigma_P$ and $v \in V^+_{s_1,s_2}$. Let $\sigma \in \mathcal{S}_\Gamma$ be an execution strategy.*

Then, σ is dynamic if and only if the following implication holds for every $s_1,s_2 \in \Sigma_P$ and for every $u \in V^+_{s_1,s_2}$:

$$\Big(\bigwedge_{v\in\Delta(s_1;s_2)} [\sigma(s_1)]_u \leq [\sigma(s_1)]_v \Big) \Rightarrow [\sigma(s_1)]_u = [\sigma(s_2)]_u \tag{L3.1}$$

Proof. Notice that, by definition of $Con(\cdot,\cdot)$ and $Hst(\cdot,\cdot,\cdot)$, $Con(Hst(\tau,s_1,\sigma),s_2)$ (for $\tau \triangleq [\sigma(s_1)]_u$) holds if and only if there is no observation event $v \in \Delta(s_1;s_2)$ which is scheduled by $\sigma(s_1)$ strictly before τ. Therefore,

$$Con(Hst(\tau,s_1,\sigma),s_2) \iff \bigwedge_{v\in\Delta(s_1;s_2)} \tau \leq [\sigma(s_1)]_v.$$

At this point, substituting the $Con(Hst(\tau,s_1,\sigma),s_2)$ expression with the equivalent formula $\bigwedge_{v\in\Delta(s_1;s_2)}[\sigma(s_1)]_u \leq [\sigma(s_1)]_v$ inside the definition of dynamic execution strategy (see Definition 3.9), the thesis follows. □

3.4 Algorithmics of Dynamic Consistency

Firstly, let us offer the following coNP-hardness result for DC; notice that, since any CSTN is also a CHyTN, the same hardness result holds for CHyTNs.

Theorem 3.1. *DC is* coNP*-hard even if the input instances $\Gamma = \langle V,A,L,\mathcal{O},\mathcal{O}V,P\rangle$ are restricted to satisfy $w_A(\cdot) \in \{-1,0\}$ and $\ell \in \{p,\neg p \mid p \in P\} \cup \{\lambda\}$ for every $(u,v,w,\ell) \in A$.*

Proof. We reduce 3-SAT to the complement of DC. Let φ be a boolean formula in 3CNF. Let X be the set of variables and let $\mathcal{C} = \{C_0,\ldots,C_{m-1}\}$ be the set of clauses comprising $\varphi = \bigwedge_{j=0}^{m-1} C_j$.

(1) Let N^φ be the CSTN $\langle V^\varphi,A^\varphi,L^\varphi,\mathcal{O}^\varphi,\mathcal{O}V^\varphi,P^\varphi\rangle$, where: $V^\varphi \triangleq X \cup \mathcal{C}$, and all the nodes are given an empty label, i.e., $L^\varphi(v) = \lambda$ for every $v \in V^\varphi$; each variable in X becomes an observation event and each clause in $\mathcal{C}$ a non-observation, i.e., $P^\varphi \triangleq \mathcal{O}V^\varphi \triangleq X$, so, $\mathcal{O}^\varphi$ is the identity map; moreover, all observation events will be forced to be executed simultaneously before any of the non-observation events, thus for every $u,v \in \mathcal{O}V^\varphi$ we have $\langle u - v \leq 0,\lambda\rangle \in A^\varphi$, and for every $x \in X$ and $C \in \mathcal{C}$ we have $\langle x - C \leq -1,\lambda\rangle \in A^\varphi$; finally, there is a negative loop among all the $C \in \mathcal{C}$ which plays an important role in the rest of the proof, particularly, for each $j = 0,\ldots,m-1$ and for each literal $\ell \in C_j$, we have $\langle C_j - C_{(j+1)\bmod m} \leq -1,\ell\rangle \in \mathcal{A}_\varphi$. Notice that $|V^\varphi| = n + m$ and $|A^\varphi| = n^2 + nm + 3m$.

(2) We show that, if φ is satisfiable, there must be an unavoidable negative circuit among all the $C_j \in \mathcal{C}$. Assume that φ is satisfiable. Let ν be a satisfying truth-assignment of φ. In order to prove that N^φ is not dynamically-consistent, observe that the restriction of N^φ w.r.t. the scenario ν is an inconsistent STN. Indeed, if for every $j = 0,\ldots,m-1$ we pick a standard arc

$\langle C_j - C_{(j+1)\bmod m} \leq -1, \ell_j \rangle$ with ℓ_j being a literal in C_j such that $\nu(\ell_j) = \top$, then we obtain a negative circuit.

(3) We show that, if φ is unsatisfiable, there can't be a negative circuit among the $C_j \in \mathcal{C}$ because for each scenario, there will be at least one j such that all three labels, α_j, β_j and γ_j will be false. Assume that φ is unsatisfiable. In order to prove that N^φ is dynamically-consistent, we exhibit a viable and dynamic execution strategy σ for N^φ. Firstly, schedule every $x \in X$ at $\sigma(x) \triangleq 0$. Therefore, by time 1, the strategy has full knowledge of the observed scenario ν. Since φ is unsatisfiable, there exists an index j_ν such that $\nu(C_{j_\nu}) = \bot$. At this point, set $\sigma(C_{(j_\nu+k)\bmod m}) \triangleq k$ for each $k = 1, \ldots, m$. The reader can verify that σ is viable and dynamic for N^φ. □

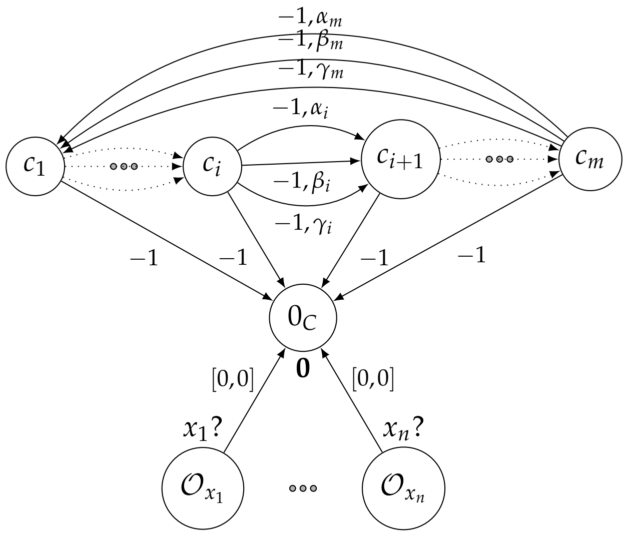


Figure 3.4: The CSTN N^φ where $\varphi(x_1, \ldots, x_n) = \bigwedge_{i=1}^m c_i$ for $c_i = (\alpha_i \vee \beta_i \vee \gamma_i)$.

An illustration of the CSTN N^φ, which was constructed in the proof of Theorem 3.1, is shown in Fig. 3.4; to ease the representation we have introduced an additional non-observation event 0_C in Fig. 3.4, which is executed at time $t = 0$, together with all of the observation events in X.

Next, we show that when the input CHyTN instances are allowed to have *both* multi-heads *and* multi-tail LTHCs then the DC-Checking problem becomes PSPACE-hard.

Theorem 3.2. *General-CHyTN-DC is* PSPACE*-hard, even if the input instances $\Gamma = \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P \rangle$ are restricted to satisfy the following two constraints:*

– $w_a(\cdot) \in [-n-1, n+1] \cap \mathbb{Z}$ and $\ell_a \in \{p, \neg p \mid p \in P\} \cup \{\lambda\}$ for every weight w_a and label ℓ_a appearing in any standard *LTHC $a \in \mathcal{A}$;*

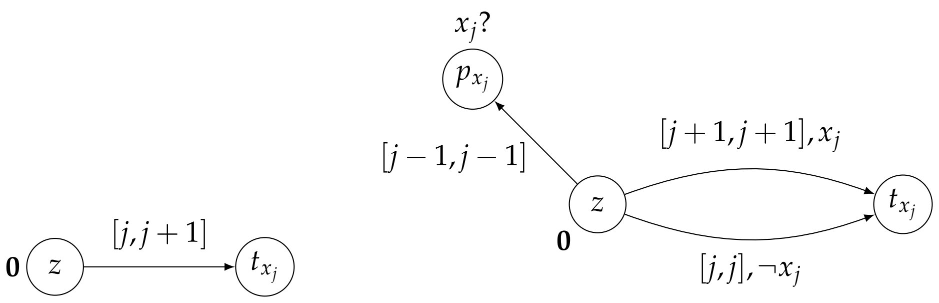
– $w_A(\cdot) \in \{-1, 0, 1\}$, $\ell_A = \lambda$ and $|A| \leq 2$ for every weight w_A and label ℓ_A appearing in any multi-tail/head *LTHC $A \in \mathcal{A}$.*

Proof. To show that General-CHyTN-DC is PSPACE-hard, we describe a reduction from the problem 3-CNF-TQBF (True Quantified Boolean Formula in 3-CNF).

Let us consider a 3-CNF quantified boolean formula with $n \geq 1$ variables and $m \geq 1$ clauses:

$$\varphi(x_1, \ldots, x_n) = Q_1 x_1 \ldots Q_n x_n \bigwedge_{i=1}^{m} (\alpha_i \vee \beta_i \vee \gamma_i),$$

where for every $j \in [n]$ the symbol Q_j is either $\exists$ or $\forall$, and where $\mathcal{C}_i = (\alpha_i \vee \beta_i \vee \gamma_i)$ is the i-th clause of φ and each $\alpha_i, \beta_i, \gamma_i \in \{x_j, \neg x_j \mid 1 \leq j \leq n\}$ is a positive or negative literal. We also say that $Q_1 x_1 \ldots Q_n x_n$ is the *prefix* of φ.



(a) Gadget for a 3-CNF-TQBF existentially quantified variable $\exists x_j$.

(b) Gadget for a 3-CNF-TQBF universally quantified variable $\forall x_j$.

Figure 3.5: Gadgets for quantified variables used in the reduction from 3-CNF-TQBF to General-CHyTN-DC.

Construction. We associate to φ a CHyTN $\Gamma_\varphi = \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{OV}, P \rangle$. In so doing, our first goal is to simulate the interaction between two players: Player-$\exists$ (corresponding to the Planner in CHyTNs) and Player-$\forall$ (corresponding to the Nature in CHyTNs), which corresponds directly to the chain of alternating quantifiers in the prefix of φ. Naturally, the Planner is going to control those variables that are quantified existentially in φ, whereas the Nature is going to control (by means of some observation events in $\mathcal{OV}$) those variables that are quantified universally in φ. Briefly, P contains one boolean variable for each universally quantified variable of φ, and V contains the following: two special events z and z' to be executed at time 0 and $n+1$, respectively; an observation event p_{x_j} for each universally quantified variable $\forall x_j$; a non-observation event t_{x_j} for each quantified variable x_j; two non-observation events l_{x_j} and $l_{\overline{x}_j}$ for each quantified variable x_j, these will play (respectively) the role of positive and negative literals of φ (i.e., the α, β and γ in each clause $\mathcal{C}_i$); finally, a non-observation event $\mathcal{C}_i$ for each clause.

Let us describe the low-level details of Γ_φ. Let us define:

$$P \triangleq \{x_j \mid \text{“}\forall x_j\text{” appears in the prefix of } \varphi\}.$$

Also, V contains a node z (i.e., the *zero* node to be executed at time $t = 0$).

Next, for each existential quantification $\exists x_j$ in the prefix of φ, V contains a node named t_{x_j} and $\mathcal{A}$ contains the following two standard LTHCs: $(z, t_{x_j}, j + 1, \lambda)$ and $(t_{x_j}, z, -j, \lambda)$; the underlying intuition being that, during execution, it will be the responsibility of the Planner to schedule t_{x_j} either at time j (and this means that the Planner chooses to set x_j to `false` in φ) or at time $j + 1$ (and this means that he chooses to set x_j to `true` in φ). See Fig. 3.5a for an illustration of the $\exists x_j$ gadget.

Moreover, for each universal quantification $\forall x_j$ in the prefix of φ (i.e., for each $x_j \in P$), V contains two nodes named p_{x_j} and t_{x_j}. Particularly, p_{x_j} is an observation event (i.e., $p_{x_j} \in \mathcal{O}V$) such that $\mathcal{O}(x_j) = p_{x_j}$; hence, $\mathcal{O}V \triangleq \{p_{x_j} \mid x_j \in P\}$. Also, for each $\forall x_j$ in φ's prefix (i.e., for each $x_j \in P$), $\mathcal{A}$ contains the following six standard LTHCs: $(z, p_{x_j}, j - 1, \lambda)$, $(p_{x_j}, z, -j + 1, \lambda)$, $(z, t_{x_j}, j + 1, x_j)$, $(t_{x_j}, z, -j - 1, x_j)$, $(z, t_{x_j}, j, \neg x_j)$ and $(t_{x_j}, z, -j, \neg x_j)$; the underlying intuition being that the Nature must choose whether to schedule t_{x_j} at time j (setting x_j to `false` in φ by controlling the observation event p_{x_j}) or at time $j + 1$ (setting x_j to `true` in φ again, by controlling the observation event p_{x_j}). Fig. 3.5b illustrates the gadget for universally quantified variables $\forall x_j$.

In both cases (existentially and universally quantified variables), the weights of the involved standard temporal constraints depend on j in such a way that their scheduling times and their corresponding propositional choices must occur one after the other in time. More precisely, for every $j \in [n]$, t_{x_j} is going to be scheduled either at time j (if x_j is `true` in φ) or at time $j + 1$ (when instead x_j is `false` in φ). In addition to this, when x_j is quantified universally in φ (i.e., when $x_j \in P$), the observation event that determines its propositional value (i.e., p_{x_j}) is always scheduled at time $j - 1$ (and this leaves enough space for the reaction time; actually, an entire unit of time between time $j - 1$ and time j).

This concludes the description of our gadgets for simulating the chain of alternating quantifiers in the prefix of φ.

At this point, we have an additional node in V, named z', which is always scheduled at time $n + 1$; for this, $\mathcal{A}$ contains the following two standard LTHCs: $(z, z', n + 1, \lambda)$ and $(z', z, -n - 1, \lambda)$. Next, we shall describe two additional gadgets (that make use of z') for simulating the 3-CNF formula $\bigwedge_{i=1}^{m}(\alpha_i \vee \beta_i \vee \gamma_i)$, one for the literals, and one for the clauses. We have a gadget for the positive (i.e., x_j) and the negative (i.e., $\neg x_j$) literals. It goes as follows: for each $j \in [n]$, V contains two nodes named l_{x_j} (i.e., positive literal) and $l_{\overline{x}_j}$ (i.e., negative literal). Moreover, $\mathcal{A}$ contains the following four standard LTHCs, $(z', l_{x_j}, 1, \lambda), (l_{x_j}, z', 0, \lambda)$ and $(z', l_{\overline{x}_j}, 1, \lambda), (l_{\overline{x}_j}, z', 0, \lambda)$, plus the following *multi-head* LTHC,

$$A^h(l_{x_j}, l_{\overline{x}_j}) \triangleq \Big(z', \{l_{x_j}, l_{\overline{x}_j}\}, \langle w(l_{x_j}), w(l_{\overline{x}_j})\rangle = \langle 0, 0\rangle, \langle L(l_{x_j}), L(l_{\overline{x}_j})\rangle = \langle \lambda, \lambda\rangle\Big),$$

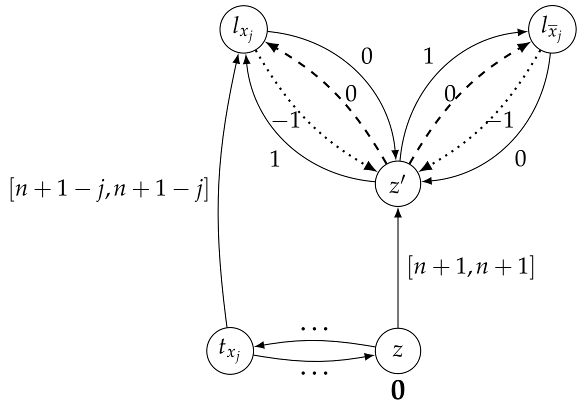
and the following *multi-tail* LTHC,

$$A^t(l_{x_j}, l_{\overline{x}_j}) \triangleq \Big(\{l_{x_j}, l_{\overline{x}_j}\}, z', \langle w(l_{x_j}), w(l_{\overline{x}_j})\rangle = \langle -1, -1\rangle, \langle L(l_{x_j}), L(l_{\overline{x}_j})\rangle = \langle \lambda, \lambda\rangle\Big).$$

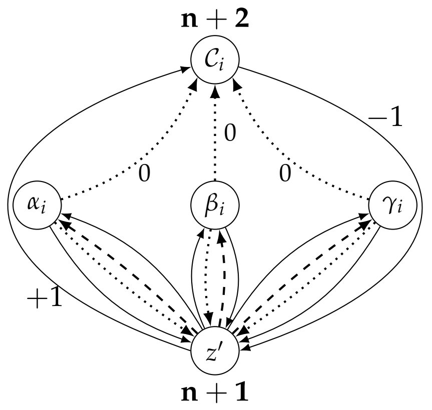
The idea here is that the standard LTHCs are going to force the scheduling times of both l_{x_j} and $l_{\overline{x}_j}$ to fall within the real interval $[n+1, n+2]$ (i.e., not before z' and at most 1 time unit after z'). Meanwhile, the multi-head constraint $A^h(l_{x_j}, l_{\overline{x}_j})$ forces that at least one between l_{x_j} and $l_{\overline{x}_j}$ happen not later than time $n+1$ (i.e., not later than the scheduling time of z'); similarly, the multi-tail constraint $A^t(l_{x_j}, l_{\overline{x}_j})$ is going to force that at least one between l_{x_j} and $l_{\overline{x}_j}$ happen not before time $n+2$ (i.e., not before the scheduling time of z' plus 1). Therefore, exactly one between l_{x_j} and $l_{\overline{x}_j}$ will be forced to happen at time $n+1$, and the other one at time $n+2$.

Up to this point, the key idea is that, for every $j \in [n]$, we can force the scheduling time of each node l_{x_j} and $l_{\overline{x}_j}$ to be uniquely determined, according to a suitable translation of the scheduling time of t_{x_j}. Particularly, we want to schedule at time $n+1$ (i.e., at the same scheduling time of z') the one node between l_{x_j} and $l_{\overline{x}_j}$ whose corresponding literal was chosen to be `false` in φ (that is l_{x_j} if t_{x_j} was scheduled at time j, and $l_{\overline{x}_j}$ if t_{x_j} was scheduled at time $j+1$); similarly, we want to schedule at time $n+2$ (i.e., at the same scheduling time of z' *plus* 1 time unit) the one node between l_{x_j} and $l_{\overline{x}_j}$ whose corresponding literal was chosen to be `true` (that is l_{x_j} if t_{x_j} was scheduled at time $j+1$, and $l_{\overline{x}_j}$ if t_{x_j} was scheduled at time j). In order to achieve this, for each $j \in [n]$, $\mathcal{A}$ contains the following two standard LTHCs: $(t_{x_j}, l_{x_j}, n+1-j, \lambda)$ and $(l_{x_j}, t_{x_j}, -n-1+j, \lambda)$ (in Fig. 3.6a they are depicted with a unique arc $t_{x_j} \xrightarrow{[k,k],\lambda} l_{x_j}$ where $k = n+1-j$); in this way, l_{x_j} is forced to happen at the same time of t_{x_j} *plus* $n+1-j$ units of time. Therefore, if t_{x_j} was scheduled at time j (i.e., x_j is `false` in φ), then node l_{x_j} is scheduled at time $j+n+1-j = n+1$; otherwise, if t_{x_j} was scheduled at time $j+1$ (i.e., x_j is `true` in φ), then node l_{x_j} is scheduled at time $j+1+n+1-j = n+2$. At this point, the scheduling time of the node $l_{\overline{x}_j}$ is determined uniquely thanks to the hyperarcs $A^h(l_{x_j}, l_{\overline{x}_j}), A^t(l_{x_j}, l_{\overline{x}_j})$ and the standard constraints $(z', l_{\overline{x}_j}, 1, \lambda)$, $(l_{\overline{x}_j}, z', 0, \lambda)$: if the node l_{x_j} is scheduled at time $n+1$ (i.e., if x_j is `false` in φ), then $l_{\overline{x}_j}$ must be scheduled at time $n+1+1 = n+2$ (i.e., if $\overline{x}_j$ is `true` in φ) so that to satisfy $A^t(l_{x_j}, l_{\overline{x}_j})$ and $(z', l_{\overline{x}_j}, 1, \lambda)$; otherwise, if l_{x_j} is scheduled at time $n+2$ (i.e., if x_j is `true` in φ), then $l_{\overline{x}_j}$ must be scheduled at time $n+1+0 = n+1$ (i.e., if $\overline{x}_j$ is `false` in φ) so that to satisfy $A^h(l_{x_j}, l_{\overline{x}_j})$ and $(l_{\overline{x}_j}, z', 0, \lambda)$. Notice that the literals $\alpha_i, \beta_i, \gamma_i$ of φ are thus instances of the nodes l_{x_i} or $l_{\overline{x}_i}$ described in Fig. 3.6a.

Finally, we describe the gadget for the clauses: for each $i \in [m]$, the CHyTN Γ_φ contains a node C_i for each clause $C_i = (\alpha_i \vee \beta_i \vee \gamma_i)$ of φ; also, each node C_i



(a) Gadget for 3-CNF-TQBF positive x_j and negative $\neg x_j$ literal.



(b) Gadget for 3-CNF-TQBF clause $\mathcal{C}_i = (\alpha_i \vee \beta_i \vee \gamma_i)$ where each $\alpha_i, \beta_i, \gamma_i$ is a positive or negative literal.

Figure 3.6: Gadgets used for variables and clauses in the reduction from 3-CNF-TQBF to General-CHyTN-DC.

is connected by:

– a multi-tail hyperarc with head in $\mathcal{C}_i$ and tails over the literals $\alpha_i, \beta_i, \gamma_i$ occurring in $\mathcal{C}_i$ and having weight 0 and label λ, i.e., by a multi-tail LTHC:

$$A^c(\alpha_i, \beta_i, \gamma_i) \triangleq \Big(\{\alpha_i, \beta_i, \gamma_i\}, \mathcal{C}_i, \langle w(\alpha_i), w(\beta_i), \\ w(\gamma)_i\rangle = \langle 0,0,0\rangle, \langle L(\alpha_i), L(\beta_i), L(\gamma)_i\rangle = \langle \lambda, \lambda, \lambda\rangle\Big),$$

for some literals $\alpha_i, \beta_i, \gamma_i \in \{l_{x_j}, l_{\overline{x}_j} \mid 1 \le j \le n\}$.

– two standard and opposite LTHCs, $(z', \mathcal{C}_i, 1, \lambda)$ and $(\mathcal{C}_i, z', -1, \lambda)$, with node z'.

See Fig. 3.6b for an illustration of the clauses' gadget; the dashed arrows form the multi-head LTHCs and the dotted arrows form the multi-tail LTHCs.

Every node of Γ_φ has an empty label, i.e., $L(v) = \lambda$ for every $v \in V$. The rationale of the clauses' gadget is that, for each i, at least one of the $\alpha_i, \beta_i, \gamma_i$ must occur at the same time instant of C_i (i.e., at least one must occur at time $n+2$, because one of the literals must be true)

This concludes our description of Γ_φ.

More formally and succinctly, the CHyTN $\Gamma_\varphi = \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P\rangle$ is defined as follows:

- $P \triangleq \{x_j \mid \text{“}\forall x_j\text{” appears in the prefix of } \varphi\}$;
- – $V \triangleq \{z, z'\} \cup \{t_{x_j} \mid 1 \leq j \leq n\} \cup \{p_{x_j} \mid x_j \in P\} \cup$
  $\cup \{l_{x_j} \mid 1 \leq j \leq n\} \cup \{l_{\overline{x}_j} \mid 1 \leq j \leq n\} \cup \{\mathcal{C}_i \mid 1 \leq i \leq m\}$;
  - $\mathcal{O}V \triangleq \{p_{x_j} \mid x_j \in P\}$ and $\mathcal{O}(x_j) = p_{x_j}$ for every $x_j \in P$;
  - $L(v) = \lambda$ for every $v \in V$;
- $$\mathcal{A} \triangleq \bigcup_{j:\text{“}\exists x_j\text{”}\in\varphi} \exists\text{-Qnt}_j \cup \bigcup_{j:\text{“}\forall x_j\text{”}\in\varphi} \forall\text{-Qnt}_j \cup$$
  $$\cup \bigcup_{j=1}^{n} \text{Var}_j \cup \bigcup_{i=1}^{m} \text{Cla}_i \cup \big\{(z, z', n+1, \lambda), (z', z, -n-1, \lambda)\big\},$$

  where:

  - $\exists\text{-Qnt}_j \triangleq \Big\{(z, t_{x_j}, j+1, \lambda), (t_{x_j}, z, -j, \lambda)\Big\}$;
    This defines the existential quantifier gadget as depicted in Fig. 3.5a;
  - $\forall\text{-Qnt}_j \triangleq \Big\{(z, p_{x_j}, j-1, \lambda), (p_{x_j}, z, -j+1, \lambda),$
    $(z, t_{x_j}, j+1, x_j), (t_{x_j}, z, -j-1, x_j), (z, t_{x_j}, j, \neg x_j), (t_{x_j}, z, -j, \neg x_j)\Big\}$;
    This defines the universal quantifier gadget as depicted in Fig. 3.5b;
  - $\text{Var}_j = \Big\{(z', l_{x_j}, 1, \lambda), (l_{x_j}, z', 0, \lambda), (z', l_{\overline{x}_j}, 1, \lambda), (l_{\overline{x}_j}, z', 0, \lambda),$
    $A^t_j \triangleq \Big(\{l_{x_j}, l_{\overline{x}_j}\}, z', \langle w_{A^t_j}(l_{x_j}), w_{A^t_j}(l_{\overline{x}_j})\rangle = \langle -1, -1\rangle,$
    $\langle L_{A^t_j}(l_{x_j}), L_{A^t_j}(l_{\overline{x}_j})\rangle = \langle \lambda, \lambda\rangle\Big),$
    $A^h_j \triangleq \Big(z', \{l_{x_j}, l_{\overline{x}_j}\}, \langle w_{A^h_j}(l_{x_j}), w_{A^h_j}(l_{\overline{x}_j})\rangle = \langle 0, 0\rangle,$
    $\langle L_{A^h_j}(l_{x_j}), L_{A^h_j}(l_{\overline{x}_j})\rangle = \langle \lambda, \lambda\rangle\Big),$
    $(t_{x_j}, l_{x_j}, n+1-j, \lambda), (l_{x_j}, t_{x_j}, -n-1+j, \lambda)\Big\}.$
    This defines the variable gadget for x_j as depicted in Fig. 3.6a;
  - $\text{Cla}_i = \Big\{(z', \mathcal{C}_j, 1), (\mathcal{C}_j, z', -1),$
    $A^c_i \triangleq \Big(\{\alpha_j, \beta_j, \gamma_j\}, \mathcal{C}_j, \langle w_{A^c_i}(\alpha_j), w_{A^c_i}(\beta_j), w_{A^c_i}(\gamma_j)\rangle = \langle 0, 0, 0\rangle,$
    $\langle L_{A^c_i}(\alpha_j), L_{A^c_i}(\beta_j), L_{A^c_i}(\gamma_j)\rangle = \langle \lambda, \lambda, \lambda\rangle\Big)\Big\}.$
    This defines the clause gadget for clause $\mathcal{C}_j = (\alpha_i \vee \beta_i \vee \gamma_i)$ as depicted in Fig. 3.6b.

Notice that $|V| \leq 1 + 4n + m = O(m+n)$ and $m_{\mathcal{A}} \leq 16n + 5m = O(m+n)$; the transformation is thus linear.

Correctness. Let us show that φ is `true` if and only if Γ_φ is dynamically-consistent.

($\Rightarrow$) Assume φ is `true`, so Player-$\exists$ has a strategy to satisfy $\bigwedge_{i=1}^{m}(\alpha_i \vee \beta_i \vee \gamma_i)$ no matter how Player-$\forall$ decides to assign the universally quantified variables of φ. Suppose that Player-$\exists$ and Player-$\forall$ alternate their choices by assigning a truth value to the variables of φ; we can construct a dynamic and viable execution strategy $\sigma \in \mathcal{S}_{\Gamma_\varphi}$ for Γ_φ by reflecting these choices, as follows. The nodes z and z' are scheduled at time 0 and $n+1$ (respectively) under all possible scenarios. For each $j = 1, \ldots, n$, the node t_{x_j} is scheduled at time j if x_j is set to `true` in φ, either by Player-$\exists$ or Player-$\forall$, otherwise at time $j+1$; and, when x_j is quantified universally in φ, the node p_{x_j} is scheduled at time $j-1$ under all possible scenarios; also, the node l_{x_j} is scheduled at time $n+2$ if x_j is set to `true` in φ, either by Player-$\exists$ or Player-$\forall$, otherwise at time $n+1$; symmetrically, $l_{\overline{x}_j}$ is scheduled at time $n+1$ if x_j is `true` in φ, otherwise at time $n+2$. Finally, for each $i = 1, \ldots, m$, the node C_i is scheduled at time $n+2$ under all possible scenarios. It is easy to check that all LTHCs of Γ_φ are satisfied by σ under all possible scenarios, so σ is viable for Γ_φ; moreover, since σ reflects the alternating choices of Player-$\exists$ and Player-$\forall$, then σ is also dynamic. Therefore, Γ_φ is dynamically-consistent.

($\Leftarrow$) Vice versa, assume that Γ_φ is dynamically-consistent. Let $\sigma \in \mathcal{S}_{\Gamma_\varphi}$ be a viable and dynamic execution strategy for Γ_φ. Firstly, we argue that σ is integer valued, i.e., that $[\sigma(s)]_v \in \mathbb{Z}$ for every $v \in V$ and $s \in \Sigma_{\Gamma_\varphi}$. Indeed, since σ is viable, it is easy to check that the scheduling time of z, z', C_i (for every $i = 1, \ldots, m$) and p_{x_j} (for every universally quantified variable x_j in φ) is forced to be 0, $n+1$, $n+2$ and $j-1$ (respectively); also, for each universally quantified variable x_j in φ, the scheduling time of p_{x_j} is forced to be $j-1$, and that of t_{x_j} is forced to be either j or $j+1$ according to whether x_j is `true` or `false` in the current scenario. Still, for each existentially quantified variable x_j in φ, the two standard LTHCs $(z, t_{x_j}, j+1, \lambda)$ and $(t_{x_j}, z, -j, \lambda)$ allow t_{x_j} to be scheduled anywhere within $[j, j+1]$, i.e., even at non-integer values. However, on one side, the scheduling time of l_{x_j} is forced to be that of t_{x_j} *plus* $n+1-j$, on the other side, l_{x_j} must be scheduled either at time $n+1$ or $n+2$ because of the multi-head $A^h(l_{x_j}, l_{\overline{x}_j})$ and multi-tail $A^t(l_{x_j}, l_{\overline{x}_j})$ LTHCs (respectively). Thus, for σ to be viable, t_{x_j} must be scheduled either at time j or $j+1$. Therefore, σ is integer valued. Now, suppose to execute σ step-by-step over the integer line; we can construct a strategy for Player-$\exists$ by reflecting the integer choices that the Planner makes to schedule the nodes of Γ_φ, as follows. For each existentially quantified variable x_j in φ, Player-$\exists$ sets x_j to `true` if the Planner schedules t_{x_j} at time $j+1$ (i.e., if l_{x_j} is scheduled at time $n+2$, and $l_{\overline{x}_j}$ at time $n+1$), and to `false` otherwise (i.e., if t_{x_j} is at time j, l_{x_j} at time $n+1$ and $l_{\overline{x}_j}$ at time $n+2$). Then, since σ is viable, for each clause C_i of φ, at least one of the literals $\alpha_i, \beta_i, \gamma_i$ must be `true`, thanks to the multi-tail LTHC $A^c(\alpha_i, \beta_i, \gamma_i)$;

and since σ is also dynamic, then Player-$\exists$ wins, so φ is `true`.

To conclude, notice that any LTHC $A \in \mathcal{A}$ of Γ_φ has weights $w_A(\cdot) \in \{-1,0,1\}$ and size $|A| \leq 3$. Since any hyperarc with three heads (tails) can be replaced by two hyperarcs each having at most two heads (tails), then General-CHyTN-DC remains PSPACE-hard even if $w_A(\cdot) \in \{-1,0,1\}$ and $|A| \leq 2$ for every multi-tail/head LTHC $A \in \mathcal{A}$. Also notice that $w_a(\cdot) \in [-n-1, n+1] \cap \mathbb{Z}$ and $\ell_a \in \{p, \neg p \mid p \in P\} \cup \{\lambda\}$ holds for every weight w_a and label ℓ_a appearing in any *standard* LTHC $a \in \mathcal{A}$. This concludes the proof. □

Theorem 3.2 motivates the study of consistency problems on CHyTNs having either only multi-head or only multi-tail hyperarcs. Since we are interested in dynamic consistency, where time moves only forward of course, and the execution strategy depends only on past observations, from now on we shall consider only multi-head CHyTNs.

3.4.1 ϵ-Dynamic Consistency

In CHyTNs, decisions about the precise timing of actions are postponed until execution time, when information gathered from the execution of the observation events can be taken into account. However, the Planner is allowed to factor in an observation, and modify its strategy in response to it, only strictly after the observation has been made (whence the strict inequality in Definition 3.7). Notice that this definition does not take into account the actual reaction time, which, in most applications, is non-negligible. In order to deliver algorithms that can also deal with the *reaction time* ϵ of the Planner we now introduce ϵ-dynamic consistency, a refined notion of dynamic consistency. The intuition underlying Definition 4.1 is that to model a specific kind of disjunctive constraint: given a small real number $\epsilon > 0$, for any two scenarios $s_1, s_2 \in \Sigma_P$ and any event $u \in V^+_{s_1,s_2}$, the scheduling time of u under s_1 must be greater or equal to either that of u under s_2 or that of v under s_2 plus ϵ for at least one $v \in \Delta(s_1; s_2)$. Let us remind the fact that, from now on, our CHyTNs admit only multi-head hyperarcs. The definition of ϵ-dynamic consistency follows below.

Definition 3.12 (ϵ-dynamic consistency). *Given any CHyTN $\langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P\rangle$ and any real number $\epsilon \in (0, \infty)$, an execution strategy $\sigma \in \mathcal{S}_\Gamma$ is* ϵ-dynamic *if it satisfies all the H_ϵ-constraints, namely, for any two scenarios $s_1, s_2 \in \Sigma_P$ and any event $u \in V^+_{s_1,s_2}$, the execution strategy σ satisfies the following constraint $H_\epsilon(s_1; s_2; u)$:*

$$[\sigma(s_1)]_u \geq \min\Big(\big\{[\sigma(s_2)]_u\big\} \cup \big\{[\sigma(s_1)]_v + \epsilon \mid v \in \Delta(s_1; s_2)\big\}\Big)$$

We say that a CHyTN Γ is ϵ-dynamically-consistent *if it admits $\sigma \in \mathcal{S}_\Gamma$ which is both viable and ϵ-dynamic.*

The problem of checking whether a given CHyTN is ϵ-dynamically-consistent is named CHyTN-ϵ-DC.

It follows directly from Definition 4.1 that, whenever $\sigma \in S_\Gamma$ satisfies some $H_\epsilon(s_1;s_2;u)$, then σ satisfies $H_{\epsilon'}(s_1;s_2;u)$ for every $\epsilon' \in (0,\epsilon]$ as well. This proves the following lemma.

Lemma 3.2. *Let Γ be a CHyTN. If Γ is ϵ-dynamically-consistent for some real $\epsilon > 0$, then Γ is ϵ'-dynamically-consistent for every $\epsilon' \in (0,\epsilon]$.*

Given any dynamically-consistent CHyTN, we may ask for the maximum reaction time ϵ of the Planner beyond which the network is no longer dynamically-consistent.

Definition 3.13 (Reaction time $\hat{\epsilon}$). *Let Γ be a CHyTN. Let $\hat{\epsilon} \triangleq \hat{\epsilon}(\Gamma)$ be the least upper bound of the set of all real numbers $\epsilon > 0$ such that Γ is ϵ-dynamically-consistent, i.e.,*

$$\hat{\epsilon} \triangleq \hat{\epsilon}(\Gamma) \triangleq \sup\{\epsilon > 0 \mid \Gamma \textit{ is } \epsilon\textit{-dynamically-consistent}\}.$$

Let us consider the (affinely) extended real numbers $\overline{\mathbb{R}} \triangleq \mathbb{R} \cup \{-\infty,\infty\}$, where *every* subset S of $\overline{\mathbb{R}}$ has an infimum and a supremum. Particularly, recall that $\sup\emptyset = -\infty$ and, if S is unbounded above, then $\sup S = \infty$.

If Γ is dynamically-consistent, then $\hat{\epsilon}(\Gamma)$ exists and $\hat{\epsilon}(\Gamma) \neq -\infty$ (i.e., the set on which we have taken the supremum in Definition 3.13 is non-empty), as it is now proved in Lemma 3.3.

Lemma 3.3. *Let σ be a dynamic execution strategy for the CHyTN Γ. Then, there exists a sufficiently small real number $\epsilon \in (0,\infty)$ such that σ is ϵ-dynamic.*

Proof. Let $s_1,s_2 \in \Sigma_P$ be two scenarios and let us consider any event $u \in V^+_{s_1,s_2}$. Since σ is dynamic, then by Lemma 3.1 the following implication necessarily holds:

$$\Big(\bigwedge_{v\in\Delta(s_1;s_2)} [\sigma(s_1)]_u \leq [\sigma(s_1)]_v\Big) \Rightarrow [\sigma(s_1)]_u \geq [\sigma(s_2)]_u \tag{*}$$

Notice that, w.r.t. Lemma 3.1, we have relaxed the equality $[\sigma(s_1)]_u = [\sigma(s_2)]_u$ in the implicand of (L3.1) by introducing the inequality $[\sigma(s_1)]_u \geq [\sigma(s_2)]_u$. At this point, we convert $(*)$ from implicative to disjunctive form, first by applying the rule of material implication[2], and then De Morgan's law[3], resulting in the following equivalent expression:

$$\Big([\sigma(s_1)]_u \geq [\sigma(s_2)]_u\Big) \vee \Big(\bigvee_{v\in\Delta(s_1;s_2)} [\sigma(s_1)]_u > [\sigma(s_1)]_v\Big) \tag{**}$$

Then, we argue that there exists a real number $\epsilon \in (0,\infty)$ such that the following disjunction holds as well:

$$\Big([\sigma(s_1)]_u \geq [\sigma(s_2)]_u\Big) \vee \Big(\bigvee_{v\in\Delta(s_1;s_2)} [\sigma(s_1)]_u \geq [\sigma(s_1)]_v + \epsilon\Big).$$

[2]The rule of material implication: $\models p \Rightarrow q \iff \neg p \vee q$.

[3]De Morgan's law: $\models \neg(p \wedge q) \iff \neg p \vee \neg q$.

In fact, since the disjunction ($**$) necessarily holds, then one can pick the following real number $\epsilon > 0$:

$$\epsilon \triangleq \min_{\langle s_1,s_2,u\rangle \in \Sigma_P \times \Sigma_P \times V^+_{s_1,s_2}} \epsilon(s_1;s_2;u),$$

where the values $\epsilon(s_1;s_2;u) \in (0,\infty)$ are defined as follows, for every $\langle s_1,s_2,u\rangle \in \Sigma_P \times \Sigma_P \times V^+_{s_1,s_2}$:

$$\epsilon(s_1;s_2;u) \triangleq \begin{cases} 1, \text{ if } [\sigma(s_1)]_u \geq [\sigma(s_2)]_u; \\ \min\Big\{[\sigma(s_1)]_u - [\sigma(s_1)]_v \mid \\ \qquad v \in \Delta(s_1;s_2),\ [\sigma(s_1)]_u > [\sigma(s_1)]_v\Big\}, \text{ otherwise.} \end{cases}$$

This implies that σ satisfies every H_ϵ-constraint of Γ, thus σ is ϵ-dynamic. □

Next, we prove a converse formulation of Lemma 3.3.

Lemma 3.4. *Let σ be an ϵ-dynamic execution strategy for a CHyTN Γ, for some real number $\epsilon \in (0,\infty)$.*
Then, σ is dynamic.

Proof. For the sake of contradiction, let us suppose that σ is not dynamic. Let F be the set of all the triplets $\langle u,s_1,s_2\rangle \in V^+_{s_1,s_2} \times \Sigma_P \times \Sigma_P$, for which the implication (L3.1) given in Lemma 3.1 does not hold. Notice, $F \neq \emptyset$; indeed, since σ is not dynamic, by Lemma 3.1 there exists at least one $\langle u,s_1,s_2\rangle$ for which (L3.1) doesn't hold. So, it holds that $\langle u,s_1,s_2\rangle \in F$ if and only if the following two properties hold:

1. $[\sigma(s_1)]_u \leq [\sigma(s_1)]_v$, for every $v \in \Delta(s_1;s_2)$;
2. $[\sigma(s_1)]_u \neq [\sigma(s_2)]_u$.

Let $\langle \hat{u},\hat{s_1}\rangle$ be an event whose scheduling time $[\sigma(\hat{s_1})]_{\hat{u}}$ is minimum and for which (1) and (2) hold, namely, let:

$$\langle \hat{u},\hat{s_1}\rangle \triangleq \arg\min\Big\{[\sigma(s_1)]_u \mid \exists s_2\, \langle u,s_1,s_2\rangle \in F\Big\}.$$

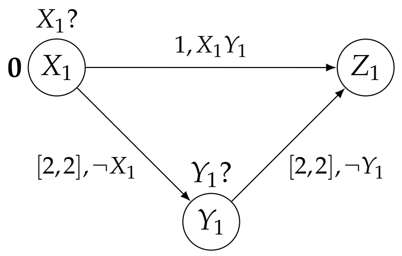
Since $\langle \hat{u},\hat{s_1}\rangle$ is minimum in $[\sigma(\hat{s_1})]_{\hat{u}}$, then $[\sigma(\hat{s_1})]_{\hat{u}} \leq [\sigma(s_2)]_{\hat{u}}$ for every $s_2 \in \Sigma_P$ such that $\langle \hat{u},\hat{s_1},s_2\rangle \in F$; moreover, since $\langle \hat{u},\hat{s_1},s_2\rangle \in F$, then $[\sigma(\hat{s_1})]_{\hat{u}} \neq [\sigma(s_2)]_{\hat{u}}$ holds by (2), so that $[\sigma(\hat{s_1})]_{\hat{u}} < [\sigma(s_2)]_{\hat{u}}$. At this point, recall that σ is ϵ-dynamic by hypothesis, hence $[\sigma(\hat{s_1})]_{\hat{u}} < [\sigma(s_2)]_{\hat{u}}$ implies that there exists $v \in \Delta(\hat{s_1};s_2)$ such that:

$$[\sigma(\hat{s_1})]_{\hat{u}} \geq [\sigma(\hat{s_1})]_v + \epsilon > [\sigma(\hat{s_1})]_v,$$

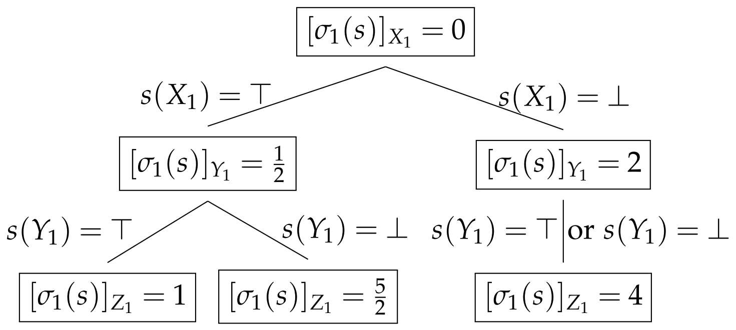
but this inequality contradicts item (1) above. Indeed, $F = \emptyset$ and σ is thus dynamic. □

In Section 3.5, the following theorem is proved.

Theorem 3.3. *For any dynamically-consistent CHyTN Γ, where V is the set of events and Σ_P is the set of scenarios, it holds that $\hat{\epsilon}(\Gamma) \geq |\Sigma_P|^{-1}|V|^{-1}$.*



(a) The CSTN $\Gamma_{\frac{1}{2}}$.



$[\sigma_1(s)]_{X_1} = 0$
$s(X_1) = \top$
$s(X_1) = \bot$
$[\sigma_1(s)]_{Y_1} = \frac{1}{2}$
$[\sigma_1(s)]_{Y_1} = 2$
$s(Y_1) = \top$
$s(Y_1) = \bot$
$s(Y_1) = \top$ or $s(Y_1) = \bot$
$[\sigma_1(s)]_{Z_1} = 1$
$[\sigma_1(s)]_{Z_1} = \frac{5}{2}$
$[\sigma_1(s)]_{Z_1} = 4$

(b) A viable and ϵ-dynamic execution strategy for $\Gamma_{\frac{1}{2}}$.

Figure 3.7: A dynamically-consistent CSTN whose viable and dynamic execution strategies are fractional.

Notice that one really needs to consider rational values for $\hat{\epsilon}$, as it is shown in the following example.

Example 3.6. *Consider the CSTN $\Gamma_{\frac{1}{2}}$ shown in Fig. 3.7a. The Planner needs to schedule and to observe X_1 at time 0 under all possible scenarios. But it is not viable to schedule Y_1 or Z_1 at time 0, because X_1 and Y_1 may turn out to be $\bot$; so Y_1 and Z_1 both need to be scheduled strictly after 0. Next, assume that X_1 turns out to be $\top$ at time 0. Then, it is not viable to schedule Y_1 at time 1, because Z_1 needs to be scheduled within time 1 if Y_1 is $\top$ and strictly after otherwise, and the Planner can't react instantaneously to the observation made at Y_1. Thus, if X_1 is $\top$ at time 0, then Y_1 needs to be scheduled at time $t \in (0,1)$, e.g., $t = \frac{1}{2}$. The corresponding execution strategy is shown in Fig. 3.7b.*

Also notice that, in Definition 3.9, dynamic consistency was defined by strict-inequality and equality constraints. However, by Theorem 4.1, dynamic consistency can also be defined in terms of H_ϵ-constraints only (i.e., no strict-inequalities are required).

Theorem 3.4. *Let Γ be a CHyTN. Let $\epsilon \triangleq |\Sigma_P|^{-1}|V|^{-1}$. Then, Γ is dynamically-consistent if and only if Γ is ϵ-dynamically-consistent.*

By Theorem 4.1, any algorithm for checking ϵ-dynamic consistency can be used to check dynamic consistency.

3.4.2 A (pseudo) Singly-Exponential Time Algorithm for DC and CHyTN-DC

In this section, we present a (pseudo) singly-exponential time algorithm for solving DC and CHyTN-DC, also producing a dynamic execution strategy whenever the input CHyTN is dynamically-consistent.

The main result of this chapter is summarized in the following theorem, which is proven in the next Section 3.5.

Theorem 3.5. *The following two algorithmic results hold for CHyTNs.*

1. *There exists an*

$$O\big(|\Sigma_P|^2|\mathcal{A}|m_{\mathcal{A}} + |\Sigma_P|^3|V||\mathcal{A}||P| + |\Sigma_P|^3|V|m_{\mathcal{A}} + |\Sigma_P|^4|V|^2|P|\big)WD$$

   time deterministic algorithm for deciding CHyTN-ϵ-DC on input $\langle\Gamma,\epsilon\rangle$, for any CHyTN $\Gamma = \langle V,\mathcal{A},L,\mathcal{O},\mathcal{O}V,P\rangle$ and any rational number $\epsilon = N/D$ where $N,D \in \mathbb{N}_+$. Particularly, given any ϵ-dynamically-consistent CHyTN Γ, the algorithm returns as output a viable and ϵ-dynamic execution strategy $\sigma \in \mathcal{S}_\Gamma$.

2. *There exists an*

$$O\big(|\Sigma_P|^3|V||\mathcal{A}|m_{\mathcal{A}} + |\Sigma_P|^4|V|^2|\mathcal{A}||P| + |\Sigma_P|^4|V|^2m_{\mathcal{A}} + |\Sigma_P|^5|V|^3|P|\big)W$$

   time deterministic algorithm for checking CHyTN-DC on any input Γ. Particularly, given any dynamically-consistent CHyTN Γ, it returns as output a viable and dynamic execution strategy $\sigma \in \mathcal{S}_\Gamma$.

Here, $W \triangleq \max_{a\in A}|w_a|$.

Since every CSTN is also a CHyTN, Theorem 4.2 holds for CSTNs as well.

We now present the reduction from CHyTN-DC to Head-HyTN-Consistency. Again, since any CSTN is a CHyTN, the same argument reduces DC to Head-HyTN-Consistency. Firstly, we argue that any CHyTN can be viewed as a succinct representation which can be expanded into an exponentially sized HyTN.

The *Expansion* of CSTNs is introduced below.

Definition 3.14 (Expansion $\langle V_\Gamma^{\text{Ex}},\Lambda_\Gamma^{\text{Ex}}\rangle$)**.** *Let $\Gamma = \langle V,\mathcal{A},L,\mathcal{O},\mathcal{O}V,P\rangle$ be a CHyTN. Consider the family of distinct and disjoint HyTNs $\langle V_s,\mathcal{A}_s\rangle$, one for each scenario*

$s \in \Sigma_P$, which is defined as follows (where $v_s \triangleq (v,s)$ for every $v \in V$ and $s \in \Sigma_P$):

$$\begin{aligned}
&V_s \triangleq \{v_s \mid v \in V_s^+\},\\
&\mathcal{A}_s \triangleq \Big\{ \big(t_s, \underbrace{\{h_s^{(1)},\ldots,h_s^{(k)}\}}_{\text{heads labeled with } s}, \underbrace{\langle w(h_s^{(1)}),\ldots,w(h_s^{(k)})\rangle}_{\text{corresponding weights}}\big) \ \Big| \\
&\qquad \big(\underbrace{t}_{\text{tail}}, \underbrace{\{h^{(1)},\ldots,h^{(k)}\}}_{\text{heads}}, \underbrace{\langle w(h^{(1)}),\ldots,w(h^{(k)})\rangle}_{\text{corresponding weights}}\big) \in \mathcal{A}_s^+ \Big\}.
\end{aligned}$$

(Of course, in the above notation, $k = 1$ when Γ is a CSTN, whereas $k \in \mathbb{N}_+$ when Γ is a CHyTN.)

Next, we define the expansion *$\langle V_\Gamma^{Ex}, \Lambda_\Gamma^{Ex}\rangle$ of Γ as follows:*

$$\langle V_\Gamma^{Ex}, \Lambda_\Gamma^{Ex}\rangle \triangleq \Big(\bigcup_{s\in\Sigma_P} V_s, \bigcup_{s\in\Sigma_P} \mathcal{A}_s \Big).$$

Notice that $V_{s_1} \cap V_{s_2} = \emptyset$ whenever $s_1 \neq s_2$ and that $\langle V_\Gamma^{\text{Ex}}, \Lambda_\Gamma^{\text{Ex}}\rangle$ is an STN/HyTN with at most $|V_\Gamma^{\text{Ex}}| \leq |\Sigma_P| \cdot |V|$ nodes and size at most $|\Lambda_\Gamma^{\text{Ex}}| \leq |\Sigma_P| \cdot |\mathcal{A}|$.

We now show that the expansion of a CHyTN can be enriched with some (extra) multi-head hyperarcs in order to model ϵ-dynamic consistency, by means of a particular HyTN which is denoted by $\mathcal{H}_\epsilon(\Gamma)$.

Definition 3.15 (HyTN $\mathcal{H}_\epsilon(\Gamma)$). *Let $\Gamma = \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P\rangle$ be a CHyTN. Given any real number $\epsilon \in (0, \infty)$, the HyTN $\mathcal{H}_\epsilon(\Gamma)$ is defined as follows:*

- *For every two scenarios $s_1, s_2 \in \Sigma_P$ and for every event node $u \in V_{s_1,s_2}^+$, define a hyperarc $\alpha \triangleq \alpha_\epsilon(s_1;s_2;u)$ as follows (with the intention to model $H_\epsilon(s_1;s_2;u)$ from Def. 4.1):*

  $$\alpha_\epsilon(s_1;s_2;u) \triangleq \big(t_\alpha, H_\alpha, w_\alpha\big), \ \forall s_1, s_2 \in \Sigma_P \text{ and } u \in V_{s_1,s_2}^+.$$

  where:

  - *$t_\alpha \triangleq u_{s_1}$ is the tail of the (multi-head) hyperarc $\alpha_\epsilon(s_1;s_2;u)$;*
  - *$H_\alpha \triangleq \{u_{s_2}\} \cup \Delta(s_1;s_2)$ is the set of the heads of $\alpha_\epsilon(s_1;s_2;u)$;*
  - *$w_\alpha(u_{s_2}) \triangleq 0$, and $w_\alpha(v) \triangleq -\epsilon$ for each $v \in \Delta(s_1;s_2)$.*

- *Consider the expansion $\langle V_\Gamma^{Ex}, \Lambda_\Gamma^{Ex}\rangle$ of Γ. Then, $\mathcal{H}_\epsilon(\Gamma)$ is defined as $\mathcal{H}_\epsilon(\Gamma) \triangleq \big(V_\Gamma^{Ex}, \mathcal{A}_{H_\epsilon}\big)$, where,*

  $$\mathcal{A}_{H_\epsilon} \triangleq \Lambda_\Gamma^{Ex} \cup \bigcup_{\substack{s_1,s_2\in\Sigma_P\\ u\in V_{s_1,s_2}^+}} \alpha_\epsilon(s_1;s_2;u).$$

Notice that each $\alpha_\epsilon(s_1;s_2;u)$ has size $|\alpha_\epsilon(s_1;s_2;u)| = 1 + \Delta(s_1;s_2) \leq 1 + |P|$. Here below, Algorithm 9 provides a pseudocode for constructing $\mathcal{H}_\epsilon(\Gamma)$.

Algorithm 6: `construct_`$\mathcal{H}(\Gamma, \epsilon)$

```
   Input: a CHyTN $\Gamma \triangleq \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P \rangle$, a rational number $\epsilon > 0$
1  foreach ($s \in \Sigma_P$) do
2      $V_s \leftarrow \{v_s \mid v \in V_s^+\}$; // see Def. 4.7
3      $A_s \leftarrow \{a_s \mid a \in A_s^+\}$; // see Def. 4.7
4  $V_\Gamma^{\text{Ex}} \leftarrow \cup_{s \in \Sigma_P} V_s$;
5  $\Lambda_\Gamma^{\text{Ex}} \leftarrow \cup_{s \in \Sigma_P} A_s$;
6  foreach ($s_1, s_2 \in \Sigma_P$ and $u \in V_{s_1,s_2}^+$) do
7      $t_\alpha \leftarrow u_{s_1}$;
8      $H_\alpha \leftarrow \{u_{s_2}\} \cup \Delta(s_1; s_2)$;
9      $w_\alpha(u_{s_2}) \leftarrow 0$;
10     foreach $v \in \Delta(s_1; s_2)$ do
11         $w_\alpha(v_{s_1}) \leftarrow -\epsilon$;
12     $\alpha_\epsilon(s_1; s_2; u) \leftarrow (t_\alpha, H_\alpha, w_\alpha)$;
13 $\mathcal{A}_{\mathcal{H}_\epsilon} \leftarrow \Lambda_\Gamma^{\text{Ex}} \cup \bigcup_{\substack{s_1,s_2 \in \Sigma_P \\ u \in V_{s_1,s_2}^+}} \alpha_\epsilon(s_1; s_2; u)$;
14 $\mathcal{H}_\epsilon(\Gamma) \leftarrow (V_\Gamma^{\text{Ex}}, \mathcal{A}_{H_\epsilon})$;
15 return $H_\epsilon(\Gamma)$;
```

Algorithm 6: Constructing $\mathcal{H}_\epsilon(\Gamma)$.

Example 3.7. *An excerpt of the HyTN $\mathcal{H}_\epsilon(\Gamma_0)$ corresponding to the CSTN Γ_0 of Fig. 3.1a is depicted in Fig. 3.8; here, two scenarios $s_1 \triangleq p \wedge q$ and $s_4 \triangleq \neg p \wedge \neg q$ are considered, on top we have ${\Gamma_0}_{s_1}^+$, whereas ${\Gamma_0}_{s_4}^+$ is below, finally, the corresponding hyperconstraints $H_\epsilon(s_1; s_4; u)$ and $H_\epsilon(s_4; s_1; u)$ are depicted as dashed hyperarcs.*

The following establishes the connection between dynamic consistency of CHyTNs and consistency of HyTNs.

Theorem 3.6. *Given any CHyTN $\Gamma = \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P \rangle$, there exists a sufficiently small real number $\epsilon \in (0, \infty)$ such that the CHyTN Γ is dynamically-consistent if and only if the HyTN $\mathcal{H}_\epsilon(\Gamma)$ is consistent.*

Moreover, the HyTN $\mathcal{H}_\epsilon(\Gamma)$ has at most so many nodes:

$$|V_{\mathcal{H}_\epsilon}| \leq |\Sigma_P| \cdot |V|,$$

so many hyperarcs:

$$|\mathcal{A}_{\mathcal{H}_\epsilon}| = O(|\Sigma_P| \cdot |\mathcal{A}| + |\Sigma_P|^2 |V|),$$

and it has size:

$$m_{\mathcal{A}_{\mathcal{H}_\epsilon}} = O(|\Sigma_P| \cdot m_{\mathcal{A}} + |\Sigma_P|^2 |V|\, |P|).$$

Proof. For any real number $\epsilon \in (0, \infty)$, let $\mathcal{H}_\epsilon(\Gamma) = \langle V_\Gamma^{\text{Ex}}, \mathcal{A}_{H_\epsilon} \rangle$ be the HyTN of Definition 4.8.

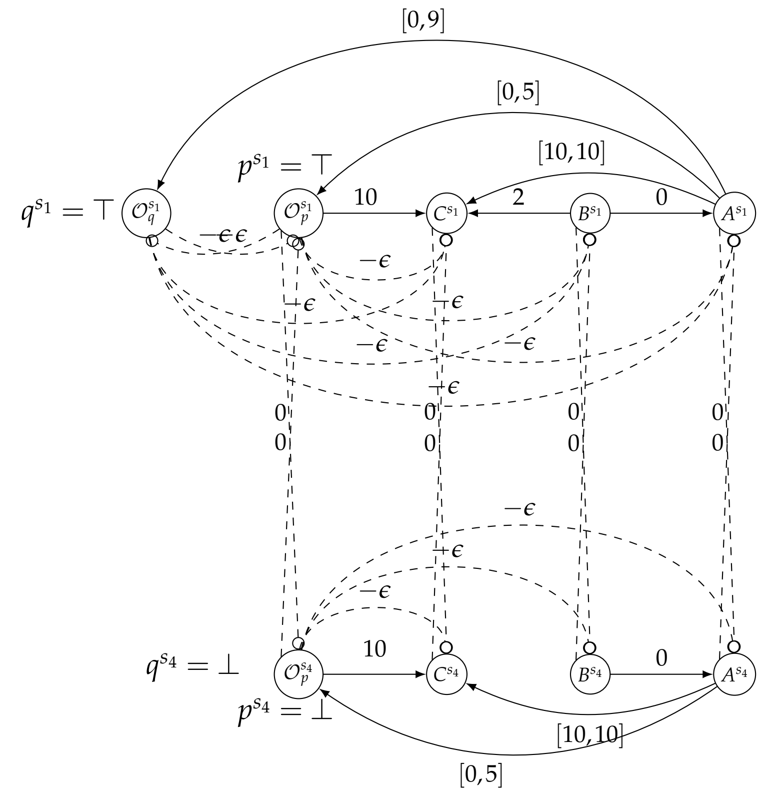


Figure 3.8: An excerpt of the HyTN $\mathcal{H}_\epsilon(\Gamma_0)$ corresponding to the CSTN Γ_0 of Fig. 3.1a, in which two scenarios, s_1 and s_4, are considered.

(1) By Definitions 4.7 and 4.8,

$$|V_{\mathcal{H}_\epsilon}| = |V_\Gamma^{\text{Ex}}| \le |\Sigma_P| \cdot |V|;$$

also,

$$|\mathcal{A}_{\mathcal{H}_\epsilon}| = |\Lambda_\Gamma^{\text{Ex}}| + \Big| \bigcup_{s_1,s_2\in\Sigma_P; u\in V^+_{s_1,s_2}} \alpha_\epsilon(s_1;s_2;u)\Big| = O(|\Sigma_P||\mathcal{A}| + |\Sigma_P|^2|V|),$$

and since $\alpha_\epsilon(s_1;s_2;u)$ has at most P heads, then $m_{\mathcal{A}_{\mathcal{H}_\epsilon}} = O(|\Sigma_P| m_{\mathcal{A}} + |\Sigma_P|^2|V|\,|P|)$.

(2) We claim that, for any $\epsilon > 0$, $\mathcal{H}_\epsilon(\Gamma)$ is consistent iff Γ is ϵ-dynamically-consistent.

($\Rightarrow$) Given any feasible schedule $\phi : V_\Gamma^{\text{Ex}} \to \mathbb{R}$ for the HyTN $\mathcal{H}_\epsilon(\Gamma)$, let $\sigma_\phi(s) \in \mathcal{S}_\Gamma$ be the execution strategy defined as follows:

$$[\sigma_\phi(s)]_v \triangleq \phi(v_s), \text{ for every } v_s \in V_\Gamma^{\text{E}}, \text{ where } v \in V \text{ and } s \in \Sigma_P.$$

Notice that each hyperarc $\alpha_\epsilon(s_1;s_2;u)$ is satisfied by ϕ if and only if the corresponding H_ϵ-constraint $H_\epsilon(s_1;s_2;u)$ is satisfied by σ_ϕ; moreover, recall that

$\Lambda_\Gamma^{\text{Ex}} \subseteq \mathcal{A}_{H_\epsilon}$, and that $\Lambda_\Gamma^{\text{Ex}}$ contains all the *original* standard/hyper difference constraints of Γ (i.e., those induced by $\mathcal{A}$, by means of Def. 4.7). At this point, since ϕ is feasible for the HyTN $\mathcal{H}_\epsilon(\Gamma)$, then σ_ϕ must be viable and ϵ-dynamic for Γ (because it satisfies all the required constraints).

Therefore, Γ is ϵ-dynamically-consistent.

($\Leftarrow$) Given any viable and ϵ-dynamic execution strategy $\sigma \in \mathcal{S}_\Gamma$, for some real number $\epsilon \in (0,\infty)$, let $\phi_\sigma : V_\Gamma^{\text{Ex}} \to \mathbb{R}$ be the schedule of the HyTN $\mathcal{H}_\epsilon(\Gamma)$ defined as follows:

$$\phi_\sigma(v_s) \triangleq [\sigma(s)]_v \text{ for every } v_s \in V_\Gamma^{\text{Ex}}, \text{ where } v \in V \text{ and } s \in \Sigma_P.$$

Also in this case, we have that $\Lambda_\Gamma^{\text{Ex}} \subseteq \mathcal{A}_{H_\epsilon}$, and a moment's reflection reveals that each hyperarc $\alpha_\epsilon(s_1;s_2;u)$ is satisfied by ϕ_σ if and only if $H_\epsilon(s_1;s_2;u)$ is satisfied by σ. At this point, since σ is viable and ϵ-dynamic for the CHyTN Γ, then ϕ_σ must be feasible for $\mathcal{H}_\epsilon(\Gamma)$. Therefore, $\mathcal{H}_\epsilon(\Gamma)$ is consistent.

This proves that, for any $\epsilon \in (0,\infty)$, $\mathcal{H}_\epsilon(\Gamma)$ is consistent if and only if Γ is ϵ-dynamically-consistent.

(3) At this point, by composition with (1), Lemma 3.3 implies that there exists a sufficiently small real number $\epsilon \in (0,\infty)$ such that Γ is dynamically-consistent if and only if $\mathcal{H}_\epsilon(\Gamma)$ is consistent. □

At this point, we are in the position to show the pseudocode for checking CHyTN-ϵ-DC, it is given in Algorithm 7:

Algorithm 7: check_CHyTN-ϵ-DC(Γ,ϵ)

```
Input: a CHyTN $\Gamma \triangleq \langle V,\mathcal{A},L,\mathcal{O},\mathcal{O}V,P\rangle$, a rational number $\epsilon \triangleq N/D$, for
       $N,D \in \mathbb{N}_+$
1 $\mathcal{H}_\epsilon(\Gamma) \leftarrow$ construct_$\mathcal{H}(\Gamma,\epsilon)$; // ref. Algorithm 9
2 foreach ($A = \langle t_A,H_A,w_A\rangle \in \mathcal{A}_{\mathcal{H}_\epsilon(\Gamma)}$ and $h \in H_A$) do
3     $w_A(h) \leftarrow w_A(h) \cdot D$; // scale all weights of $\mathcal{H}_\epsilon(\Gamma)$, from Q to $\mathbb{Z}$
4 $\phi \leftarrow$ check_HEAD-HYTN-CONSISTENCY($\mathcal{H}_\epsilon(\Gamma)$); // ref. Thm 2.7
5 if ($\phi$ is a feasible schedule of $\mathcal{H}_\epsilon(\Gamma)$) then
6     foreach (event node $v \in V_{\mathcal{H}_\epsilon(\Gamma)}$) do
7         $\phi(v) \leftarrow \phi(v)/D$; // re-scale back to size the scheduling time, from $\mathbb{Z}$ to Q,
          w.r.t. $\epsilon$
8     return $\langle YES,\phi\rangle$;
9 else return NO ;
```

Algorithm 7: Checking CHyTN-ϵ-DC on input (Γ,ϵ).

whereas, the pseudocode for checking CHyTN-DC is provided in Algorithm 8, here below:

Notice that the latter (Algorithm 8) invokes the former (Algorithm 7); more details follow.

Algorithm 8: `check_DC/CHyTN-DC`(Γ)

```
Input: a CHyTN $\Gamma \triangleq \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P \rangle$
1 $\hat{\epsilon} \leftarrow |\Sigma_P|^{-1}|V|^{-1}$; // ref. Thm. 4.1
2 return check_CHyTN-$\epsilon$-DC$(\Gamma, \hat{\epsilon})$;
```

Algorithm 8: Checking CHyTN-DC on input Γ.

Description of Algorithm 8 Firstly, Algorithm 8 computes a sufficiently small *rational* number $\epsilon \in (0, \infty) \cap \mathbf{Q}$, by relying on Theorem 4.1, i.e., it is set $\hat{\epsilon} \triangleq |\Sigma_P|^{-1}|V|^{-1}$ (line 1). Secondly, Algorithm 7 is invoked on input $(\Gamma, \hat{\epsilon})$. At this point, Algorithm 7 firstly constructs $\mathcal{H}_{\hat{\epsilon}}(\Gamma)$ (line 1 of Algorithm 7) by invoking Algorithm 9, and then it scales *every* hyperarc's weight, appearing in $\mathcal{H}_{\hat{\epsilon}}(\Gamma)$, from $\mathbf{Q}$ to $\mathbb{Z}$ (at lines 2-3). This is done by multiplying each weight by a factor D (line 3), where $D \in \mathbb{N}_+$ is the denominator of $\hat{\epsilon}$ (i.e., $D = |\Sigma_P| \cdot |V|$). Thirdly, $\mathcal{H}_{\hat{\epsilon}}(\Gamma)$ is solved with the Head-HyTN-Consistency-Checking algorithm underlying Theorem 2.7 (at line 4), i.e., within the underlying algorithmic engine, an instance of the Head-HyTN-Consistency problem is solved by reducing it to the problem of determining winning regions in a carefully constructed MPG (see [32, 33] for the details of such a reduction). At this point, if the Head-HyTN-Consistency algorithm outputs `YES`, together with a feasible schedule ϕ of $\mathcal{H}_{\hat{\epsilon}}(\Gamma)$, then the time values of ϕ are scaled back to size w.r.t. $\hat{\epsilon}$, and then $\langle \texttt{YES}, \phi \rangle$ is returned as output (lines 5-8); otherwise, the output is simply `NO` (at line 9). Still, notice that, thanks to Item 3 of Theorem 2.7, we could also return a negative certificate, because negative instances are well characterized in terms of generalized negative cycles (see Definition 2.5).

Remark 3.1. *The same algorithm, with essentially the same upper bound on its running time and space, works also in case we allow for arbitrary boolean formulae as labels, rather than just conjunctions.*

Remark 3.2. *We remark that the HyTN/MPG algorithm that is at the heart of our approach requires integer weights (i.e., it requires that $w(u, v) \in \mathbb{Z}$ for every $(u, v) \in A$); somehow, we could not play it differently (see [32,33] for a discussion). Moreover, the algorithm always computes an integer solution to HyTNs/MPGs and, therefore, it always computes* rational *feasible schedules for the CHyTNs given as input. As such, it seems to us that this "requirement" actually turns out to be a plus in practice. It is actually the integer assumption that allows us to analyze the algorithm quantitatively, also presenting a sharp lower bounding analysis on the critical value of the reaction time $\hat{\epsilon}$, where the CHyTN transits from being, to not being, dynamically-consistent. We believe that these issues deserve much attention, and going into them required a "discrete" approach to the notion of numbers.*

The correctness and the time complexity of Algorithms 7 and 8 is analyzed in Section 3.5.

3.5 Bounding Analysis on the Reaction Time $\hat{\epsilon}$

In this section we present an asymptotically sharp lower bound for $\hat{\epsilon}(\Gamma)$, that is the critical value of reaction time where the CHyTN transits from being, to not being, dynamically-consistent. The proof technique introduced in this analysis is applicable more generally, when dealing with linear difference constraints which include strict inequalities. This bound implies that Algorithm 8 is a (pseudo) singly-exponential time algorithm for solving CHyTN-DC.

To begin, we are going to provide a proof of Theorem 3.3; for this, let us firstly introduce some further notation.

Let $\Gamma \triangleq \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P\rangle$ be a dynamically-consistent CHyTN. By Theorem 4.3, there exists $\epsilon > 0$ such that the HyTN $\mathcal{H}_\epsilon(\Gamma)$ is consistent. Then, let $\phi : V_\Gamma^{\text{Ex}} \to \mathbb{R}$ be a feasible schedule for $\mathcal{H}_\epsilon(\Gamma)$. For any hyperarc $A = \langle t_A, H_A, w_A\rangle \in \mathcal{A}_{\mathcal{H}_\epsilon}$, define a standard arc a_A as follows:

$$a_A \triangleq \langle t_A, \hat{h}, w_A(\hat{h})\rangle, \text{ where } \hat{h} \triangleq \arg\min_{h \in H_A} \big(\phi(h) - w_A(h)\big).$$

Then, notice that the network $T_\epsilon^\phi(\Gamma) \triangleq \langle V_\Gamma^{\text{Ex}}, \bigcup_{A \in \mathcal{A}_{\mathcal{H}_\epsilon}} a_A\rangle$ is always an STN. Moreover, a moment's reflection reveals that, by definition of $\hat{h}$ as above, then ϕ is a feasible schedule for the STN $T_\epsilon^\phi(\Gamma)$.

At this point, assuming $v \in V_\Gamma^{\text{Ex}}$, let us consider the *fractional part* r_v of ϕ_v, i.e.,

$$r_v \triangleq \phi_v - \lfloor \phi_v \rfloor.$$

Then, let $R \triangleq \{r_v\}_{v \in V_\Gamma^{\text{Ex}}}$ be the set of all the fractional parts induced by V_Γ^{Ex}. Sort R by the common ordering on $\mathbb{R}$ and assume that $S \triangleq \{r_1, \ldots, r_k\}$ is the resulting ordered set (without repetitions), i.e., $|S| = k$, $S = R$, $r_1 < \ldots < r_k$. Now, let $\texttt{pos}(v) \in [k]$ be the (unique) index position such that:

$$r_{\texttt{pos}(v)} = r_v.$$

Then, we define a new fractional part r'_v as follows:

$$r'_v \triangleq \frac{\texttt{pos}(v) - 1}{|\Sigma_P| \cdot |V|}, \tag{NFP}$$

and a new schedule function as follows:

$$\phi'_v \triangleq \lfloor \phi_v \rfloor + r'_v. \tag{NSF}$$

Then the following holds.

Remark 3.3. *Notice that (NFP) doesn't alter the ordering relation among the fractional parts, i.e.,*

$$r'_u < r'_v \iff r_u < r_v, \textit{ for any } u, v \in V_\Gamma^{Ex}.$$

Moreover, since $\big(\texttt{pos}(v) - 1\big) < |\Sigma_P| \cdot |V|$, observe that (NSF) doesn't change the *value of any integer part, i.e.,*

$$\lfloor \phi'_u \rfloor = \lfloor \phi_u \rfloor, \textit{ for any } u \in V_\Gamma^{Ex}.$$

We are now in the position to prove Theorem 3.3.

Proof of Theorem 3.3. Let $\Gamma \triangleq \langle V, \mathcal{A}, L, \mathcal{O}, \mathcal{O}V, P \rangle$ be dynamically-consistent. By Theorem 4.3 there exists $\epsilon' > 0$ such that $\mathcal{H}_{\epsilon'}(\Gamma)$ is consistent and it admits some *feasible* schedule $\phi : V_\Gamma^{\text{Ex}} \to \mathbb{R}$. As mentioned, ϕ is feasible for the STN $T_{\epsilon'}^{\phi}(\Gamma)$. Now, let $\hat{\epsilon} \triangleq |\Sigma_P|^{-1}|V|^{-1}$. Moreover, let $T_{\hat{\epsilon}}^{\phi}(\Gamma)$ be the STN obtained from $T_{\epsilon'}^{\phi}(\Gamma)$ simply by replacing, in the weights of the arcs, each weight $-\epsilon'$ with $-\hat{\epsilon}$. We argue that ϕ' (as defined in (NSF) w.r.t. ϕ, V, Σ_P), is a feasible schedule for the STN $T_{\hat{\epsilon}}^{\phi}(\Gamma)$. Indeed, every constraint of $T_{\hat{\epsilon}}^{\phi}(\Gamma)$ has form $\phi_v - \phi_u \leq w$, for some $w \in \mathbb{Z}$ or $w = -\hat{\epsilon}$.

- Consider the case $w \in \mathbb{Z}$. Notice that $\phi_v - \phi_u \leq w$ holds because ϕ is feasible for the STN $T_{\epsilon'}^{\phi}(\Gamma)$. Then, it is not difficult to see that $\phi'_v - \phi'_u \leq w$ holds as well, because of Remark 3.3.

- Consider the case $w = -\hat{\epsilon}$. Notice that $\phi_v - \phi_u \leq -\epsilon'$ holds because ϕ is feasible for the STN $T_{\epsilon'}^{\phi}(\Gamma)$.

  Then, notice that the following implication always holds,

  $$\phi_v - \phi_u \leq -\epsilon' \implies \phi_v \neq \phi_u.$$

  Hence, again by Remark 3.3, we can conclude that $\phi'_v \neq \phi'_u$. At this point, we observe that the temporal distance between ϕ'_u and ϕ'_v is, therefore, at least $\hat{\epsilon}$ by definition of (NSF) and (NFP), i.e.,

  $$\phi'_u - \phi'_v \geq |\Sigma_P|^{-1}|V|^{-1} = \hat{\epsilon}.$$

  That is to say, $\phi'_v - \phi'_u \leq -\hat{\epsilon}$.

This proves that ϕ' is a feasible schedule also for the STN $T_{\hat{\epsilon}}^{\phi}(\Gamma)$. Since $T_{\hat{\epsilon}}^{\phi}(\Gamma)$ is thus consistent, then, a moment's reflection reveals that $\mathcal{H}_{\hat{\epsilon}}(\Gamma)$ is consistent as well thanks to the same schedule ϕ'.

Therefore, by Theorem 4.3, the CHyTN Γ is $\hat{\epsilon}$-dynamically-consistent, provided that $\hat{\epsilon} \triangleq |\Sigma_P|^{-1}|V|^{-1}$. □

The correctness proof and the time complexity of Algorithm 8 is given next.

Proof of Theorem 4.2. To begin, notice that some of the temporal constraints introduced during the reduction step depend on a sufficiently small parameter $\hat{\epsilon} \in (0, \infty) \cap \mathbf{Q}$, whose magnitude turns out to depend on the size of the input CHyTN. It is proved below that the time complexity of the algorithm depends multiplicatively on D, where $\hat{\epsilon} = N/D$ for some $N, D \in \mathbb{N}_+$. By Theorem 3.3,

$\hat{\epsilon}(\Gamma) \geq |\Sigma_P|^{-1}|V|^{-1}$; so line 1 of Algorithm 8 is correct. Therefore, as a corollary of Theorem 4.3, we obtain that Algorithm 8 correctly decides DC.

Concerning its time complexity, the most time-expensive step of the algorithm is clearly line 4 of Algorithm 7, which relies on Theorem 2.7 in order to solve an instance of Head-HyTN-Consistency on input $\mathcal{H}_\epsilon(\Gamma)$. From Theorem 4.3 we have an upper bound on the size of $\mathcal{H}_\epsilon(\Gamma)$, while Theorem 2.7 gives us a pseudo-polynomial upper bound for the computation time. Also, recall that we scale weights by a factor D at lines 2-3 of Algorithm 7, where $\hat{\epsilon} = N/D$ for some $N, D \in \mathbb{N}_+$. Thus, by composition, Algorithm 8 decides CHyTN-DC in a time T_Γ^{Algo3} which is bounded as follows, where $W \triangleq \max_{a \in A} |w_a|$ and $D \in \mathbb{N}_+$:

$$T_\Gamma^{\text{Algo3}} = O\Big((|V_{\mathcal{H}_\epsilon(\Gamma)}| + |\mathcal{A}_{\mathcal{H}_\epsilon(\Gamma)}|) m_{\mathcal{A}_{\mathcal{H}_\epsilon(\Gamma)}} \Big) WD.$$

Whence, taking into account the upper bound on the size of $\mathcal{H}_\epsilon(\Gamma)$ give by Theorem 4.3, the following holds:

$$\begin{aligned}
T_\Gamma^{\text{Algo3}} &= O\Big((|\Sigma_P||V| + |\Sigma_P||\mathcal{A}| + |\Sigma_P|^2|V|)(|\Sigma_P| m_{\mathcal{A}} + |\Sigma_P|^2|V|\,|P|) \Big) WD \\
&= O\Big(\cancel{|\Sigma_P|^2|V| m_{\mathcal{A}}} + \cancel{|\Sigma_P|^3|V|^2|P|} + |\Sigma_P|^2|\mathcal{A}| m_{\mathcal{A}} + |\Sigma_P|^3|\mathcal{A}||V||P| \\
&\qquad\qquad + |\Sigma_P|^3|V| m_{\mathcal{A}} + |\Sigma_P|^4|V|^2|P| \Big) WD \\
&= O\Big(|\Sigma_P|^2|\mathcal{A}| m_{\mathcal{A}} + |\Sigma_P|^3|\mathcal{A}||V||P| + |\Sigma_P|^3|V| m_{\mathcal{A}} + |\Sigma_P|^4|V|^2|P| \Big) WD.
\end{aligned}$$

By Theorem 3.3, it is sufficient to check ϵ-dynamic consistency for $\hat{\epsilon} = |\Sigma_P|^{-1}|V|^{-1}$. Therefore, the following worst-case time bound holds on Algorithm 8:

$$T_\Gamma^{\text{Algo3}} = O\Big(|\Sigma_P|^3|V||\mathcal{A}| m_{\mathcal{A}} + |\Sigma_P|^4|\mathcal{A}||V|^2|P| + |\Sigma_P|^4|V|^2 m_{\mathcal{A}} + |\Sigma_P|^5|V|^3|P| \Big) W.$$

Since $|\Sigma_P| \leq 2^{|P|}$, the (pseudo) singly-exponential time bound follows. $\square$

At this point, a natural question is whether the lower bound given by Theorem 3.3 can be improved up to $\hat{\epsilon}(\Gamma) = \Omega(|V|^{-1})$. In turn, this would improve the time complexity of Algorithm 8 by a factor $|\Sigma_P|$. However, the following theorem shows that this is not the case, by exhibiting a CSTN for which $\hat{\epsilon}(\Gamma) = 2^{-\Omega(|P|)}$. This proves that the lower bound given by Theorem 3.3 is (almost) asymptotically sharp.

Theorem 3.7. *For each $n \in \mathbb{N}_+$ there exists a CSTN Γ^n such that:*

$$\hat{\epsilon}(\Gamma^n) < 2^{-n+1} = 2^{-|P^n|/3+1},$$

where P^n is the set of boolean variables of Γ^n.

Proof. For each $n \in \mathbb{N}_+$, we define a CSTN $\Gamma^n \triangleq \langle V^n, A^n, L^n, \mathcal{O}^n, \mathcal{O}V^n, P^n \rangle$ as follows. See Fig. 3.9 for a clarifying illustration.

- $V^n \triangleq \{X_i, Y_i, Z_i \mid 1 \le i \le n\}$;
- $A^n \triangleq B \cup \bigcup_{i=1}^{n} C_i \cup \bigcup_{i=1}^{n-1} D_i$
  where:
  - $B \triangleq \{\langle X_1 - v \le 0, \lambda\rangle \mid v \in V^n\} \cup \{\langle Z_1 - X_1 \le 1, X_1 \wedge Y_1\rangle\}$;
  - $C_i \triangleq \{\langle Y_i - X_i \le 2, \neg X_i\rangle, \langle X_i - Y_i \le -2, \neg X_i\rangle, \langle Z_i - Y_i \le 2, \neg Y_i\rangle, \langle Y_i - Z_i \le -2, \neg Y_i\rangle\}$;
  - $D_i \triangleq \{\langle X_{i+1} - X_i \le 5, Z_i\rangle, \langle X_i - X_{i+1} \le -5, Z_i\rangle, \langle X_{i+1} - Y_i\rangle \le 5, \neg Z_i\rangle, \langle Y_i - X_{i+1} \le -5, \neg Z_i\rangle, \langle Z_{i+1} - Y_i \le 5, Z_i \wedge X_{i+1} \wedge Y_{i+1}\rangle, \langle Y_i - Z_{i+1} \le -5, Z_i \wedge X_{i+1} \wedge Y_{i+1}\rangle, \langle Z_{i+1} - Z_i \le 5, \neg Z_i \wedge X_{i+1} \wedge Y_{i+1}\rangle, \langle Z_i - Z_{i+1} \le -5, \neg Z_i \wedge X_{i+1} \wedge Y_{i+1}\rangle\}$;
- $L^n(v) \triangleq \lambda$ for every $v \in V^n$; $\mathcal{O}V^n \triangleq V^n$; $\mathcal{O}^n(v) \triangleq v$ for every $v \in \mathcal{O}V^n$; $P^n \triangleq V^n$.

We exhibit an execution strategy $\sigma_n : \Sigma_{P^n} \to \Phi_{V^n}$, which we will show is dynamic and viable for Γ^n.

Let $\{\delta_i\}_{i=1}^{n}$ and $\{\Delta_i\}_{i=1}^{n}$ be two real valued sequences such that:

$$(1)\ \Delta_1 \triangleq 1; (2)\ 0 < \delta_i < \Delta_i; (3)\ \Delta_i \triangleq \min(\delta_{i-1}, \Delta_{i-1} - \delta_{i-1}).$$

Then, the following also holds for every $1 \le i \le n$:

$$(4)\ 0 < \Delta_i \le 2^{-i+1},$$

where the equality holds if and only if $\delta_i = \Delta_i / 2$.

Hereafter, provided that $s \in \Sigma_P$ and $\ell \in P^*$, we will denote $\mathbb{1}_{s(\ell)} \triangleq 1$ if $s(\ell) = \top$ and $\mathbb{1}_{s(\ell)} \triangleq 0$ if $s(\ell) = \bot$.

We are in the position to define $\sigma_n(s)$ for any $s \in \Sigma_P$:

- $[\sigma_n(s)]_{X_1} \triangleq 0$;
- $[\sigma_n(s)]_{Y_1} \triangleq \delta_1 \mathbb{1}_{s(X_1)} + 2\mathbb{1}_{s(\neg X_1)}$;
- $[\sigma_n(s)]_{Z_1} \triangleq \mathbb{1}_{s(X_1 \wedge Y_1)} + (2 + [\sigma_n(s)]_{Y_1})\mathbb{1}_{s(\neg X_1 \vee \neg Y_1)}$;
- $[\sigma_n(s)]_{X_i} \triangleq 5 + [\sigma_n(s)]_{X_{i-1}} \mathbb{1}_{s(Z_{i-1})} +$
  $+ [\sigma_n(s)]_{Y_{i-1}} \mathbb{1}_{s(\neg Z_{i-1})}$, for any $2 \le i \le n$;
- $[\sigma_n(s)]_{Y_i} \triangleq [\sigma_n(s)]_{X_i} + \delta_i \mathbb{1}_{s(X_i)} + 2\mathbb{1}_{s(\neg X_i)}$, for any $2 \le i \le n$;
- $[\sigma_n(s)]_{Z_i} \triangleq \big(5 + [\sigma_n(s)]_{Y_{i-1}} \mathbb{1}_{s(Z_{i-1})} +$
  $+ [\sigma_n(s)]_{Z_{i-1}} \mathbb{1}_{s(\neg Z_{i-1})}\big) \mathbb{1}_{s(X_i \wedge Y_i)} +$
  $+ (2 + [\sigma_n(s)]_{Y_i}) \mathbb{1}_{s(\neg X_i \vee \neg Y_i)}$, for any $2 \le i \le n$;

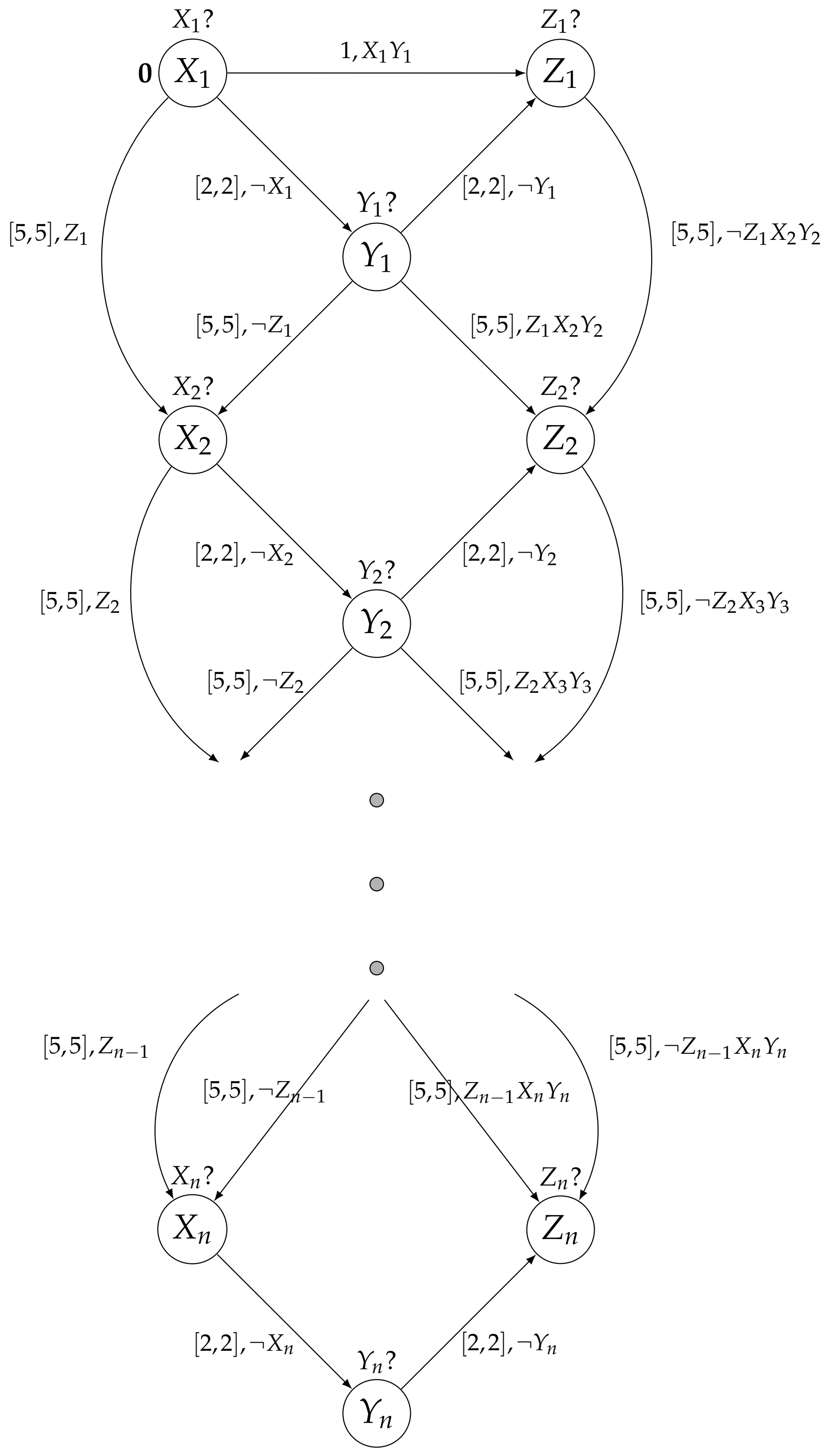


Figure 3.9: A CSTN Γ^n such that $\hat{\epsilon}(\Gamma^n) = 2^{-\Omega(|P^n|)}$.

Let us prove, by induction on $n \geq 1$, that σ_n is viable and dynamic for Γ^n.

- *Base case*. Let $n = 1$. Notice that Γ^1 almost coincides with the CSTN $\Gamma_{\frac{1}{2}}$ described in Example 3.6; so, it is really needed that $0 < \delta_1 < 1$. Then, by construction, σ_1 leads to the schedule depicted in Figure 3.10. This shows that σ_1 is viable and dynamic for Γ^1.

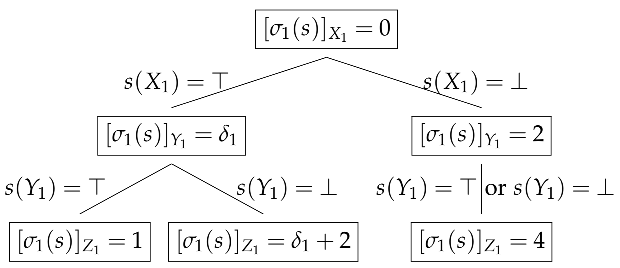

$[\sigma_1(s)]_{X_1} = 0$
$s(X_1) = \top$
$s(X_1) = \bot$
$[\sigma_1(s)]_{Y_1} = \delta_1$
$[\sigma_1(s)]_{Y_1} = 2$
$s(Y_1) = \top$
$s(Y_1) = \bot$
$s(Y_1) = \top$ or $s(Y_1) = \bot$
$[\sigma_1(s)]_{Z_1} = 1$
$[\sigma_1(s)]_{Z_1} = \delta_1 + 2$
$[\sigma_1(s)]_{Z_1} = 4$

Figure 3.10: A viable and dynamic execution strategy for the base case $n = 1$.

- *Inductive step*. Let us assume that σ_{n-1} is viable and dynamic for Γ^{n-1}. Then, by construction, $[\sigma_n(s)]_v = [\sigma_{n-1}(s)]_v$ for every $s \in \Sigma_P$ and $v \in V_{n-1}$. Hence, by induction hypothesis, σ_n is viable and dynamic on V_{n-1}. Moreover, by construction, σ_n leads to the schedule depicted in Figure 3.11 and Figure 3.12. This shows that σ_n is viable and dynamic even on $V_n \setminus V_{n-1}$. Thus, σ_n is viable and dynamic for Γ^n, i.e., Γ^n is dynamically-consistent.

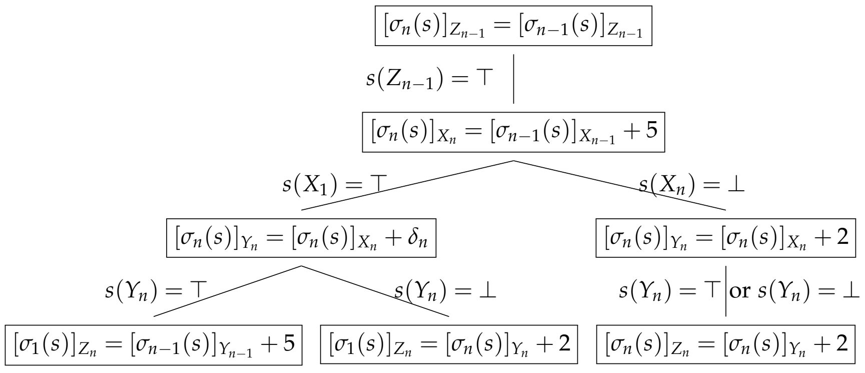
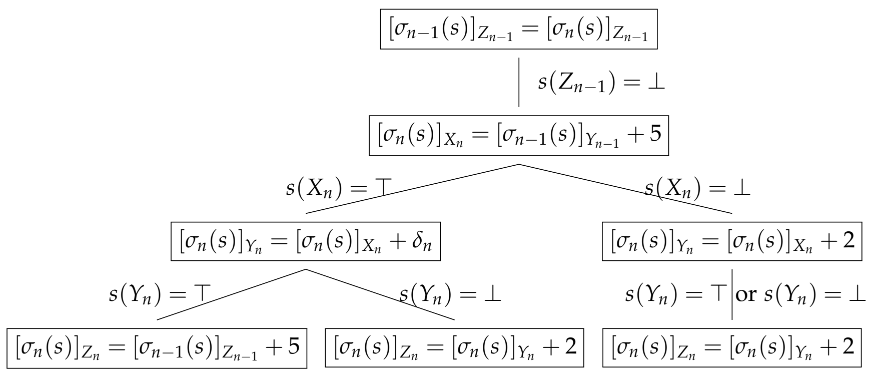

$[\sigma_n(s)]_{Z_{n-1}} = [\sigma_{n-1}(s)]_{Z_{n-1}}$
$s(Z_{n-1}) = \top$
$[\sigma_n(s)]_{X_n} = [\sigma_{n-1}(s)]_{X_{n-1}} + 5$
$s(X_1) = \top$
$s(X_n) = \bot$
$[\sigma_n(s)]_{Y_n} = [\sigma_n(s)]_{X_n} + \delta_n$
$[\sigma_n(s)]_{Y_n} = [\sigma_n(s)]_{X_n} + 2$
$s(Y_n) = \top$
$s(Y_n) = \bot$
$s(Y_n) = \top$ or $s(Y_n) = \bot$
$[\sigma_1(s)]_{Z_n} = [\sigma_{n-1}(s)]_{Y_{n-1}} + 5$
$[\sigma_1(s)]_{Z_n} = [\sigma_n(s)]_{Y_n} + 2$
$[\sigma_n(s)]_{Z_n} = [\sigma_n(s)]_{Y_n} + 2$

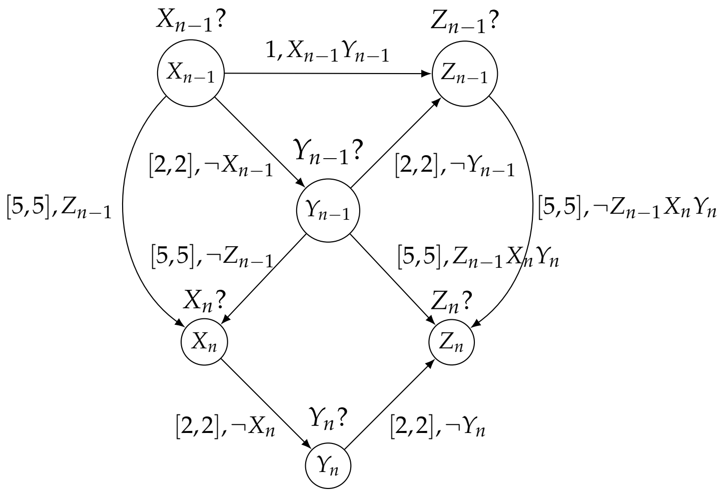
Figure 3.11: A viable and dynamic execution strategy for the inductive step $n - 1 \rightsquigarrow n$ when $s(Z_{n-1}) = \top$.

We claim that $\hat{e}(\Gamma^n) < 2^{-n+1} = 2^{-|P^n|/3+1}$ for every $n \geq 1$. Consider the following scenario $\hat{s}$ for $1 \leq i \leq n$:

$$\hat{s}(X_i) \triangleq \hat{s}(Y_i) \triangleq \top; \;\; \hat{s}(Z_i) \triangleq \begin{cases} \top, & \text{if } \delta_i \leq \Delta_i/2 \\ \bot, & \text{if } \delta_i > \Delta_i/2 \end{cases}$$


$[\sigma_{n-1}(s)]_{Z_{n-1}} = [\sigma_n(s)]_{Z_{n-1}}$
$s(Z_{n-1}) = \bot$
$[\sigma_n(s)]_{X_n} = [\sigma_{n-1}(s)]_{Y_{n-1}} + 5$
$s(X_n) = \top$
$s(X_n) = \bot$
$[\sigma_n(s)]_{Y_n} = [\sigma_n(s)]_{X_n} + \delta_n$
$[\sigma_n(s)]_{Y_n} = [\sigma_n(s)]_{X_n} + 2$
$s(Y_n) = \top$
$s(Y_n) = \bot$
$s(Y_n) = \top$ or $s(Y_n) = \bot$
$[\sigma_n(s)]_{Z_n} = [\sigma_{n-1}(s)]_{Z_{n-1}} + 5$
$[\sigma_n(s)]_{Z_n} = [\sigma_n(s)]_{Y_n} + 2$
$[\sigma_n(s)]_{Z_n} = [\sigma_n(s)]_{Y_n} + 2$

(a) An execution strategy for the inductive step $n-1 \rightsquigarrow n$ when $s(Z_{n-1}) = \bot$



(b) An excerpt of Γ^n relevant to the inductive step $n-1 \rightsquigarrow n$.

Figure 3.12: The inductive step $n-1 \rightsquigarrow n$ when $s(Z_{n-1}) = \bot$.

We shall assume that σ is an execution strategy for Γ^n and study necessary conditions to ensure that σ is viable and dynamic, provided that the observations follow the scenario $\hat{s}$. First, σ must schedule X_1 at time $[\sigma(\hat{s})]_{X_1} = 0$. Then, since $\hat{s}(X_1) = \top$, we must have $0 < [\sigma(\hat{s})]_{Y_1} < 1$, because of the constraint $(Z_1 - X_1 \leq 1, X_1 \wedge Y_1)$. Stated otherwise, it is necessary that:

$$0 < [\sigma(\hat{s})]_{Y_1} - [\sigma(\hat{s})]_{X_1} < \Delta_1.$$

After that, since $\hat{s}(Y_1) = \top$, then σ must schedule Z_1 at time $[\sigma(\hat{s})]_{Z_1} = 1 = \Delta_1$. A moment's reflection reveals that almost identical necessary conditions now recur for X_2, Y_2, Z_2, with the crucial variation that it will be necessary to require: $0 < [\sigma(\hat{s})]_{Y_2} < \Delta_2$. Indeed, by proceeding inductively, it will be necessary that for every $1 \leq i \leq n$ and every $n \in \mathbb{N}_+$:

$$0 < [\sigma(\hat{s})]_{Y_i} - [\sigma(\hat{s})]_{X_i} < \Delta_i.$$

As already observed in (4), $0 < \Delta_n \leq 2^{-n+1}$. Thus, any viable and dynamic execution strategy σ for Γ^n must satisfy:

$$0 < [\sigma(\hat{s})]_{Y_n} - [\sigma(\hat{s})]_{X_n} < \frac{1}{2^{n-1}} = \frac{1}{2^{|P^n|/3-1}}.$$

Therefore, once the Planner has observed the outcome $\hat{s}(X_n) = \top$ from the observation event X_n, then he must react by scheduling Y_n within time $2^{-n+1} = 2^{-|P^n|/3+1}$ in the future w.r.t. $[\sigma(\hat{s})]_{X_n}$.

Therefore, $\hat{\epsilon}(\Gamma^n) < 2^{-n+1} = 2^{-|P^n|/3+1}$ holds for every $n \geq 1$. □

3.6 Related Works

This section discusses of some alternative approaches offered by the current literature. Recall that the article of [113] has already been discussed in the introduction.

The work of [21] provided the first sound-and-complete algorithm for checking the dynamic controllability of CSTNs with Uncertainty (CSTNU), and thus it can be employed for checking the dynamic consistency of CSTNs as a special case. The algorithm reduces the DC-Checking of CSTNUs to the problem of solving Timed Game Automata (TGA). Nevertheless, no worst-case upper bound on the time complexity of the procedure was provided in [21]. Still, one may observe that solving TGAs is a problem of much higher complexity than solving MPGs. Compare the following known facts: solving 1-player TGAs is PSPACE-complete and solving 2-player TGAs is EXP-complete; on the contrary, the problem of determining MPGs lies in NP ∩ coNP and it is currently an open problem to prove whether it is in P. Indeed, the algorithm offered in [21] has not been proven to be singly-exponentially time bounded, to the best of our knowledge it is still open whether singly-exponential time TGA-based algorithms for DC do exist.

Next, a sound algorithm for checking the dynamic controllability of CSTNUs was given by Combi, Hunsberger, Posenato in [23]. However, it was not shown to be complete. To the best of our knowledge, it is currently open whether or not it can be extended in order to prove completeness w.r.t. the CSTNU model.

Regarding the particular CSTN model, [69] presented, at the same conference in which the preliminary version of this work appeared, a sound-and-complete DC-checking algorithm for CSTNs. It is based on the propagation of temporal constraints labeled by propositions. However, to the best of our knowledge, the worst-case complexity of the algorithm is currently unsettled. Also notice that the algorithm in [69] works on CSTNs only, regardless of the CHyTN model. Indeed, we believe that our approach (based on tractable games plus reaction-time $\hat{\epsilon}$) and the approach of [69] (based on the propagation of labeled temporal constraints) can benefit from each other; for instance, recently [68] presented an alternative, equivalent semantics for ϵ-dynamic consistency, as well as a sound-and-complete ϵ-DC-checking algorithm based on the propagation of labeled constraints.

Finally, in [17], it is introduced and studied π-DC, a sound notion of dynamic consistency with an instantaneous reaction time, i.e., one in which the Planner is allowed to react to any observation at the same instant of time in which the observation is made. It turns out that π-DC is not equivalent to ϵ-DC with $\epsilon = 0$, and that the latter is actually inadequate for modeling an instantaneous reaction-time. Still, a simple reduction from π-DC-Checking to DC-Checking is identified; combined with Theorem 4.2, this provides a π-DC-Checking procedure whose time complexity remains (pseudo) singly-exponential in the number of propositional variables.

4 Instantaneous Reaction-Time in Dynamic Consistency Checking of Conditional Simple Temporal Networks

Chapter Abstract

In order to address the DC-Checking problem, in Chapter 3 we introduced ε-DC (a refined, more realistic, notion of DC), and provided an algorithmic solution to it. Given that DC implies ε-DC for some sufficiently small $\varepsilon > 0$, and that for every $\varepsilon > 0$ it holds that ε-DC implies DC, we offered a sharp lower bounding analysis on the critical value of the *reaction-time* $\hat{\varepsilon}$ under which the two notions coincide. This delivered the first (pseudo) singly-exponential time algorithm for the DC-Checking of CSTNs. However, the ε-DC notion is interesting per se, and the ε-DC-Checking algorithm in Chapter 3 [34] rests on the assumption that the reaction-time satisfies $\varepsilon > 0$; leaving unsolved the question of what happens when $\varepsilon = 0$. In this chapter, we introduce and study π-DC, a sound notion of DC with an *instantaneous* reaction-time (i.e., one in which the planner can react to any observation at the same instant of time in which the observation is made). Firstly, we demonstrate by a counter-example that π-DC is not equivalent to 0-DC, and that 0-DC is actually inadequate for modeling DC with an instantaneous reaction-time. This shows that the main results obtained in Chapter 3 do not apply directly, as they were formulated, to the case of $\varepsilon = 0$. Motivated by this observation, as a second contribution, our previous tools are extended in order to handle π-DC, and the notion of *ps-tree* is introduced, also pointing out a relationship between π-DC and HyTN-Consistency. Thirdly, a simple *reduction* from π-DC to DC is identified. This allows us to design and to analyze the first sound-and-complete π-DC-Checking procedure. Remarkably, the time complexity of the proposed algorithm remains (pseudo) singly-exponential in the number of propositional letters.
This chapter is a revised version of [17].

4.1 Introduction and Motivation

In Chapter 2, it was unveiled that HyTNs and MPGs are a natural underlying combinatorial model for the DC-Checking of CSTNs [34]. Indeed, STNs have been recently generalized into *Hyper Temporal Networks (HyTNs)* [32,33], by considering weighted directed hypergraphs, where each hyperarc models a disjunctive temporal constraint called *hyper-constraint*. Also in Chapter 2, the computational equivalence between checking the consistency of HyTNs and determining the winning regions in Mean Payoff Games (MPGs) was also pointed out [32,33]. The approach was shown to be viable and robust thanks to some extensive experimental evaluations [33]. MPGs [14,49,123] are a family of two-player infinite games played on finite graphs, with direct and important applications in model-checking and formal verification [61], and they are known for having theoretical interest in computational complexity for their special place among the few (natural) problems lying in $\mathsf{NP} \cap \mathsf{coNP}$.

All this combined, in Chapter 3 it was provided the first (pseudo) singly-exponential time algorithm for the DC-Checking problem, also producing a dynamic execution strategy whenever the input CSTN is DC [34]. For this, it was introduced ε-DC (a refined, more realistic, notion of DC), and provided the first algorithmic solution to it. Next, given that DC implies ε-DC for some sufficiently small $\varepsilon > 0$, and that for every $\varepsilon > 0$ it holds that ε-DC implies DC, it was offered a sharp lower bounding analysis on the critical value of the *reaction-time* $\hat{\varepsilon}$ under which the two notions coincide. This delivered the first (pseudo) singly-exponential time algorithm for the DC-Checking of CSTN. However, the ε-DC notion is interesting per se, and the ε-DC-Checking algorithm in [34] rests on the assumption that the reaction-time satisfies $\varepsilon > 0$; leaving unsolved the question of what happens when $\varepsilon = 0$.

4.1.1 Contribution

In this chapter we introduce and study π-DC, a sound notion of DC with an *instantaneous* reaction-time (i.e., one in which the planner can react to any observation at the same instant of time in which the observation is made). Firstly, we provide a counter-example showing that π-DC is not just the $\varepsilon = 0$ special case of ε-DC. This implies that the algorithmic results obtained in [34] do not apply directly to the study of those situation where the planner is allowed to react instantaneously. Motivated by this observation, as a second contribution, we extend the previous formulation to capture a sound notion of DC with an instantaneous reaction-time, i.e., π-DC. Basically, it turns out that π-DC needs to consider an additional internal ordering among all the observation nodes that occur at the same time instant. Next, the notion of *ps-tree* is introduced to reflect the ordered structure of π-DC, also pointing out a relationship between π-DC and HyTN-Consistency. Thirdly, a simple *reduction* from π-DC to DC is identified. This allows us to design and to analyze the first sound-and-complete π-DC-Checking procedure. The time complexity of the proposed algorithm remains (pseudo) singly-exponential in the number $|P|$ of propositional letters.

4.2 Background and Notation

Recall that Simple Temporal Networks (STNs) [44] have been detailed in Subsection 3.2.1, Chapter 2. The *Hyper Temporal Network* (HyTN) model has been introduced in Section 2.4, Chapter 2. Also, we shall refer to the CSTN model [34, 67, 113] detailed in Definition 3.1, Section 3.3, Chapter 3. In all of the following definitions we implicitly refer to some CSTN $\Gamma = \langle V, A, L, \mathcal{O}, \mathcal{O}V, P \rangle$.

4.2.1 ε-Dynamic-Consistency

In CSTNs, decisions about the precise timing of actions are postponed until execution time, when informations meanwhile gathered at the observation nodes can be taken into account. However, the planner is allowed to factor in an outcome, and differentiate its strategy according to it, only *strictly* after the outcome has been observed (whence the strict inequality in Definition 3.7). Notice that this definition does not take into account the reaction-time, which, in most applications, is non-negligible. In order to deliver algorithms that can also deal with the *reaction-time* $\varepsilon > 0$ of the planner, we introduced in Chapter 3 [17, 34] a refined notion of DC.

Definition 4.1 (ε-Dynamic-Consistency). *Given any CSTN $\langle V, A, L, \mathcal{O}, \mathcal{O}V, P \rangle$ and any real number $\varepsilon \in (0, +\infty)$, an ES $\sigma \in \mathcal{S}_\Gamma$ is* ε-dynamic *if it satisfies all of the H_ε-constraints, namely, for any two scenarios $s_1, s_2 \in \Sigma_P$ and any event $u \in V^+_{s_1,s_2}$, the ES σ satisfies the following constraint, which is denoted by $H_\varepsilon(s_1; s_2; u)$:*

$$[\sigma(s_1)]_u \geq \min\Big(\{[\sigma(s_2)]_u\} \cup \{[\sigma(s_1)]_v + \varepsilon \mid v \in \Delta(s_1; s_2)\}\Big)$$

We say that a CSTN Γ is ε-dynamically-consistent (ε-DC) *if it admits $\sigma \in \mathcal{S}_\Gamma$ which is both viable and ε-dynamic.*

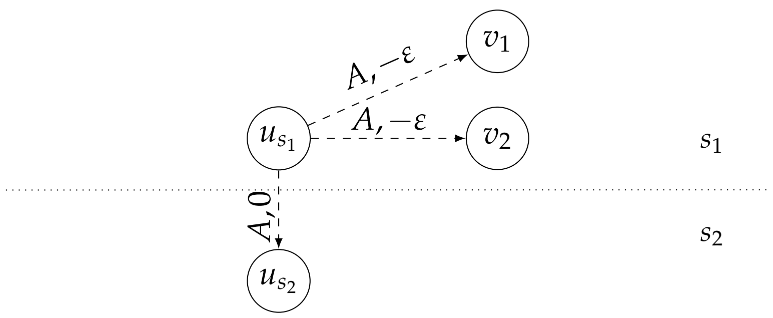


Figure 4.1: An $H_\varepsilon(s_1; s_2; u)$ constraint, modeled as a hyperarc.

As shown in Chapter 3 [17, 34], ε-DC can be modeled in terms of Head-HyTN-Consistency. Fig. 4.1 depicts an illustration of an $H_\varepsilon(s_1; s_2; u)$ constraint, modeled as an hyperarc.

Also, in [34] we proved that DC coincides with $\hat{\varepsilon}$-DC, provided that $\hat{\varepsilon} \triangleq |\Sigma_P|^{-1}|V|^{-1}$.

Theorem 4.1. *Let $\hat{\varepsilon} \triangleq |\Sigma_P|^{-1}|V|^{-1}$. Then, Γ is DC if and only if Γ is $\hat{\varepsilon}$-DC. Moreover, if Γ is ε-DC for some $\varepsilon > 0$, then Γ is ε'-DC for every $\varepsilon' \in (0, \varepsilon]$.*

Then, the main result offered in [34] is a (pseudo) singly-exponential time DC-checking procedure (based on HyTNs).

Theorem 4.2. *There exists an:*

$$O\Big(|\Sigma_P|^3|A|^2|V| + |\Sigma_P|^4|A||V|^2|P| + |\Sigma_P|^5|V|^3|P|\Big)W$$

time algorithm for checking DC on any input CSTN $\Gamma = \langle V, A, L, \mathcal{O}, \mathcal{O}V, P \rangle$. In particular, given any dynamically-consistent CSTN Γ, the algorithm returns a viable and dynamic ES for Γ. Here, $W \triangleq \max_{a \in A} |w_a|$.

4.3 DC with Instantaneous Reaction-Time

Theorem 4.1 points out the equivalence between ε-DC and DC, that arises for a sufficiently small $\varepsilon > 0$. However, Definition 4.1 makes sense even if $\varepsilon = 0$, so a natural question is what happens to the above mentioned relationship between DC and ε-DC when $\varepsilon = 0$. In this section we first show that 0-DC doesn't imply DC, and, moreover, that 0-DC is in itself too weak to capture an adequate notion of DC with an instantaneous reaction-time. In light of this we will introduce a stronger notion, which is named *ordered-Dynamic-Consistency (π-DC)*; this will turn out to be a suitable notion of DC with an instantaneous reaction-time.

Example 4.1 (CSTN $\Gamma_\Box$). *Consider the following CSTN:*

$$\Gamma_\Box = (V_\Box, A_\Box, L_\Box, \mathcal{O}_\Box, \mathcal{O}V_\Box, P_\Box);$$

see Fig. 4.2 for an illustration.
– $V_\Box = \{\bot, \top, A, B, C\}$;
– $A_\Box = \{(\top - \bot \leq 1, \lambda), (\bot - \top \leq -1, \lambda), (\top - A \leq 0, b \wedge \neg c), (\top - B \leq 0, a \wedge c), (\top - C \leq 0, \neg a \wedge \neg b), (\bot - A \leq 0, \lambda), (A - \bot \leq 0, \neg b), (A - \bot \leq 0, c), (\bot - B \leq 0, \lambda), (B - \bot \leq 0, \neg a), (A - \bot \leq 0, \neg c), (\bot - C \leq 0, \lambda), (C - \bot \leq 0, a), (C - \bot \leq 0, b)\}$;
– $L_\Box(A) = L_\Box(B) = L_\Box(C) = L_\Box(\bot) = L_\Box(\top) = \lambda$;
– $\mathcal{O}_\Box(a) = A$, $\mathcal{O}_\Box(b) = B$, $\mathcal{O}_\Box(c) = C$;
– $\mathcal{O}V_\Box = \{A, B, C\}$;
– $P_\Box = \{a, b, c\}$.

Proposition 4.1. *The CSTN $\Gamma_\Box$ (Example 4.1, Fig. 4.2) is 0-DC.*

Proof. Consider the execution strategy $\sigma_\Box : \Sigma_{P_\Box} \to \Psi_{V_\Box}$:
– $[\sigma_\Box(s)]_A \triangleq s(a \wedge b \wedge \neg c) + s(\neg a \wedge b \wedge \neg c)$;
– $[\sigma_\Box(s)]_B \triangleq s(a \wedge b \wedge c) + s(a \wedge \neg b \wedge c)$;
– $[\sigma_\Box(s)]_C \triangleq s(\neg a \wedge \neg b \wedge \neg c) + s(\neg a \wedge \neg b \wedge c)$;
– $[\sigma_\Box(s)]_\bot \triangleq 0$ and $[\sigma_\Box(s)]_\top \triangleq 1$, for every $s \in \Sigma_P$.

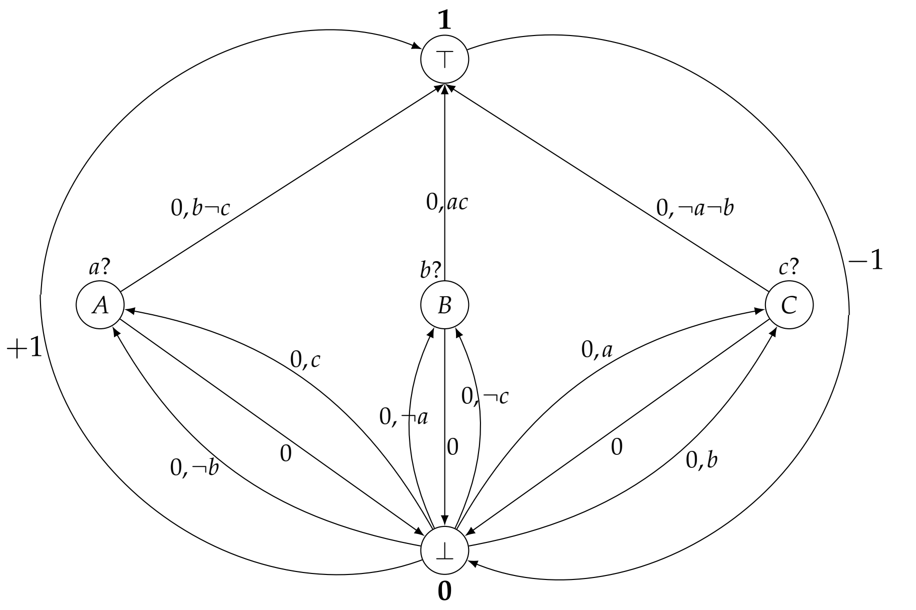


Figure 4.2: A CSTN $\Gamma_\Box$ which is 0-DC but not DC.

An illustration of $\sigma_\Box$ is offered in Fig 4.3. Three cubical graphs are depicted in which every node is labelled as v_s for some $(v, s) \in V_\Box \times \Sigma_{P_\Box}$: an edge connects $v_{1 s_1}$ and $v_{2 s_2}$ if and only if: (i) $v_1 = v_2$ and (ii) the Hamming distance between s_1 and s_2 is unitary; each scenario $s \in \Sigma_{P_\Box}$ is represented as $s = \alpha\beta\gamma$ for $\alpha, \beta, \gamma \in \{0, 1\}$, where $s(a) = \alpha$, $s(b) = \beta$, $s(c) = \gamma$; moreover, each node $v_s = (v, s) \in V_\Box \times \Sigma_{P_\Box}$ is filled in black if $[\sigma_\Box(s)]_v = 0$, and in white if $[\sigma_\Box(s)]_v = 1$. So all three 3-cubes own both black and white nodes, but each of them, in its own dimension, decomposes into two identically colored 2-cubes. Fig. 4.4 offers another visualization of $\sigma_\Box$ in which every component

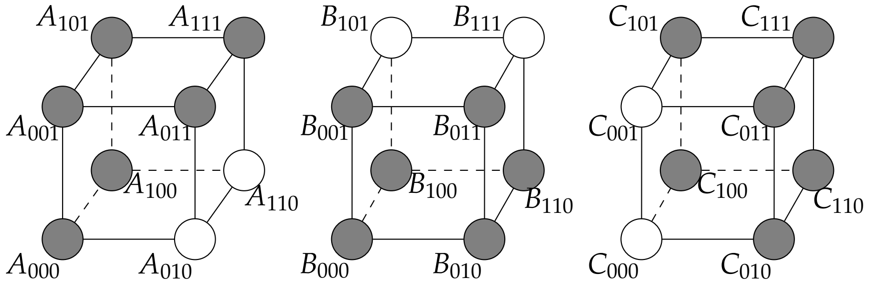


Figure 4.3: The ES $\sigma_\Box$ for the CSTN $\Gamma_\Box$.

of the depicted graph corresponds to a restriction STN $\Gamma^+_{\Box s}$ for some $s \in \Sigma_{P_\Box}$, where $s_i, s_j \in \Sigma_{P_\Box}$ are grouped together whenever $\Gamma^+_{\Box s_i} = \Gamma^+_{\Box s_j}$. It is easy to see from Fig. 4.4 that $\sigma_\Box$ is viable for $\Gamma_\Box$. In order to check that $\sigma_\Box$ is 0-dynamic, look again at Fig. 4.4, and notice that for every $s_i, s_j \in \Sigma_{\Box}$, where $s_i \neq s_j$, there exists an event node $X \in \{A,B,C\}$ such that $[\sigma_{\Box}(s_i)]_X = 0 = [\sigma_{\Box}(s_j)]_X$ and $s_i(X) \neq s_j(X)$. With this in mind it is easy to check that all of the H_0 constraints are thus satisfied by $\sigma_{\Box}$. Therefore, the CSTN $\Gamma_{\Box}$ is 0-DC. □

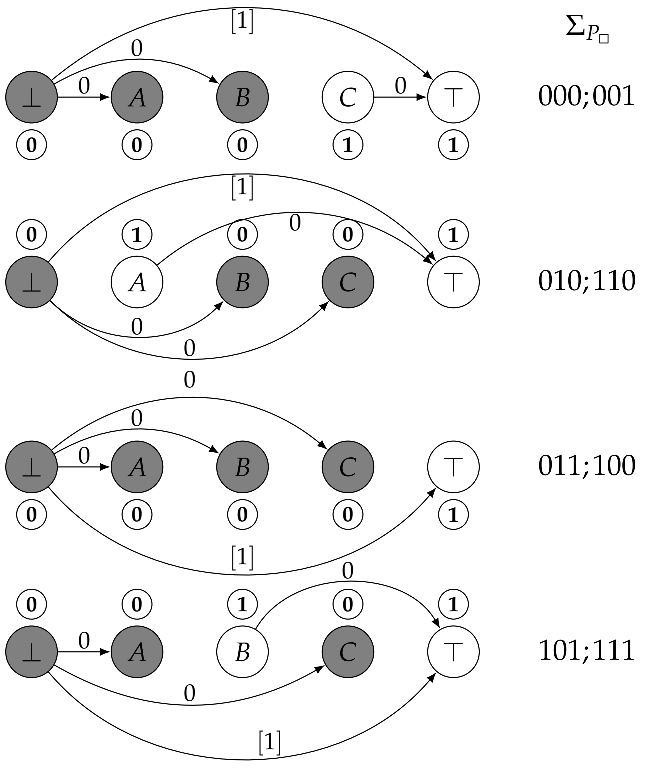


Figure 4.4: The restrictions $\Gamma^{+}_{\Box s}$ for $s \in \Sigma_{P_{\Box}}$, where the execution times $[\sigma_{\Box}(s)]_v \in \{\mathbf{0},\mathbf{1}\}$ are depicted in bold face.

Proposition 4.2. *The CSTN $\Gamma_{\Box}$ is not DC.*

Proof. Let σ be a viable ES for $\Gamma_{\Box}$. Then, σ must be the ES $\sigma_{\Box}$ depicted in Fig. 4.4, there is no other choice here. Let $\hat{s} \in \Sigma_{P_{\Box}}$. Then, it is easy to check from Fig. 4.4 that: (i) $[\sigma_{\Box}(\hat{s})]_{\bot} = 0$, $[\sigma_{\Box}(\hat{s})]_{\top} = 1$, and it holds $[\sigma_{\Box}(\hat{s})]_X \in \{0,1\}$ for every $X \in \{A,B,C\}$; (ii) there exists at least two observation events $X \in \{A,B,C\}$ such that $[\sigma(\hat{s})]_X = 0$; still, (iii) there is no $X \in \{A,B,C\}$ such that $[\sigma(s)]_X = 0$ for every $s \in \Sigma_{P_{\Box}}$, i.e., no observation event is executed first at all possible scenarios. Therefore, the ES $\sigma_{\Box}$ is not dynamic. □

We now introduce a stronger notion of dynamic consistency; it is named *ordered-Dynamic-Consistency (π-DC)*, and it takes explicitly into account an additional ordering between the observation events scheduled at the same execution time.

Definition 4.2 (π-Execution-Strategy). *An* ordered-Execution-Strategy (π-ES) *for* Γ *is a mapping:*

$$\sigma : s \mapsto ([\sigma(s)]^t, [\sigma(s)]^\pi),$$

where $s \in \Sigma_P$, $[\sigma(s)]^t \in \Phi_V$, *and finally,* $[\sigma(s)]^\pi : \mathcal{O}V_s^+ \rightleftharpoons \{1, \ldots, |\mathcal{O}V_s^+|\}$ *is bijective. The set of* π*-ES of* Γ *is denoted by* $\mathcal{S}_\Gamma$*. For any* $s \in \Sigma_P$*, the* execution time *of an event* $v \in V_s^+$ *in the schedule* $[\sigma(s)]^t \in \Phi_{V_s^+}$ *is denoted by* $[\sigma(s)]^t_v \in \mathbb{R}$*; the* position *of an observation* $\mathcal{O}_p \in \mathcal{O}V_s^+$ *in* $\sigma(s)$ *is* $[\sigma(s)]^\pi_{\mathcal{O}_p}$*. We require positions to be* coherent *w.r.t. execution times, i.e.,*

$$\forall(\mathcal{O}_p, \mathcal{O}_q \in \mathcal{O}V_s^+) \textit{ if } [\sigma(s)]^t_{\mathcal{O}_p} < [\sigma(s)]^t_{\mathcal{O}_q}, \textit{ then } [\sigma(s)]^\pi_{\mathcal{O}_p} < [\sigma(s)]^\pi_{\mathcal{O}_q}.$$

In addition, it is worth adopting the notation:

$$[\sigma(s)]^\pi_v \triangleq |\mathcal{O}V| + 1, \textit{ whenever } v \in V_s^+ \setminus \mathcal{O}V.$$

Definition 4.3 (π-History). *Let* $\sigma \in \mathcal{S}_\Gamma$*,* $s \in \Sigma_P$*, and let* $\tau \in \mathbb{R}$ *and* $\psi \in \{1, \ldots, |V|\}$*. The* ordered-history π-Hst(τ, ψ, s, σ) *of* τ *and* ψ *in the scenario s, under the* π*-ES* σ*, is defined as:*

$$\pi\text{-Hst}(\tau, \psi, s, \sigma) \triangleq \{(p, s(p)) \in P \times \{0,1\} \mid \\ \mathcal{O}_p \in \mathcal{O}V_s^+, [\sigma(s)]^t_{\mathcal{O}_p} \leq \tau, [\sigma(s)]^\pi_{\mathcal{O}_p} < \psi\}.$$

We are finally in the position to define π-DC.

Definition 4.4 (π-Dynamic-Consistency). *Any* $\sigma \in \mathcal{S}_\Gamma$ *is called* π-dynamic *when, for any two scenarios* $s_1, s_2 \in \Sigma_P$ *and any event* $v \in V^+_{s_1,s_2}$*, if* $\tau \triangleq [\sigma(s_1)]^t_v$ *and* $\psi \triangleq [\sigma(s_1)]^\pi_v$*, then:*

$$\text{Con}(\pi\text{-Hst}(\tau, \psi, s_1, \sigma), s_2) \Rightarrow [\sigma(s_2)]^t_v = \tau, [\sigma(s_2)]^\pi_v = \psi.$$

We say that Γ *is* π-dynamically-consistent (π-DC) *if it admits* $\sigma \in \mathcal{S}_\Gamma$ *which is both viable and* π*-dynamic. The problem of checking whether a given CSTN is* π*-DC is named* π-DC*-Checking.*

Remark 4.1. *It is easy to see that, due to the strict inequality "*$[\sigma(s)]^\pi_{\mathcal{O}_p} < \psi$*" in the definition of* π-Hst$(\cdot)$ *(Definition 4.3), in a* π*-dynamic* π*-ES, there must be exactly one* $\mathcal{O}_{p'} \in \mathcal{O}V$*, for some* $p' \in P$*, which is executed at first (w.r.t. both execution time and position) under all possible scenarios* $s \in \Sigma_P$*.*

Proposition 4.3. *The CSTN* $\Gamma_\Box$ *is not* π*-DC.*

Proof. The proof goes almost in the same way as that of Proposition 4.2. In particular, no observation event is executed first (i.e., at time $t = 0$ and position $\psi = 1$) in all possible scenarios. Since there is no first-in-time observation event, then, the ES σ is not π-dynamic. $\square$

We provide next a CSTN which is π-DC but not DC.

Example 4.2. *Define $\Gamma_\pi = (V_\pi, A_\pi, \mathcal{O}_\pi, \mathcal{O}V_\pi, P_\pi)$ as follows. $V_\pi = \{\mathcal{O}_p, X, \top\}$, $A_\pi = \{(\top - \mathcal{O}_p \le 1, \lambda), (\mathcal{O}_p - \top \le -1, \lambda), (X - \mathcal{O}_p \le 0, p), (\top - X \le 0, \neg p)\}$, $\mathcal{O}_\pi(p) = \mathcal{O}_p$, $\mathcal{O}V_\pi = \{\mathcal{O}_p\}$, $P_\pi = \{p\}$. Fig. 4.5 depicts the CSTN Γ_π.*

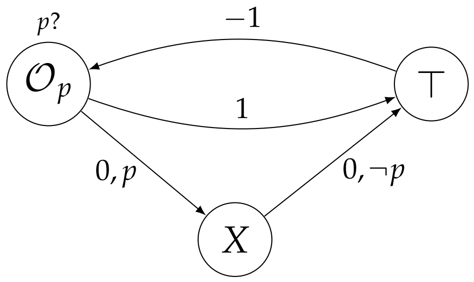


Figure 4.5: The CSTN Γ_π.

Proposition 4.4. *The CSTN Γ_π is π-DC, but it is not DC.*

Proof. Let $s_1, s_2 \in \Sigma_{P_\pi}$ be two scenarios such that $s_1(p) = 1$ and $s_2(p) = 0$. Consider the π-ES σ defined as follows: $[\sigma(s_1)]^t_{\mathcal{O}_p} = [\sigma(s_1)]^t_X = 0$, $[\sigma(s_1)]^t_\top = 1$; and $[\sigma(s_2)]^t_{\mathcal{O}_p} = 0$, $[\sigma(s_2)]^t_X = [\sigma(s_2)]^t_\top = 1$; finally, $[\sigma(s)]^\pi_{\mathcal{O}_p} = 1$, $[\sigma(s)]^\pi_X = [\sigma(s)]^\pi_\top = 2$, for all $s \in \{s_1, s_2\}$. Then, σ is viable and π-dynamic for Γ_π. To see that Γ_π is not DC, pick any $\varepsilon > 0$. Notice that any viable ES must schedule X either at $t = 0$ or $t = 1$, depending on the outcome of $\mathcal{O}_p$, which in turn happens at $t = 0$; however, in any ε-dynamic strategy, the planner can't react to the outcome of $\mathcal{O}_p$ before time $t = \varepsilon > 0$. This implies that Γ_π is not ε-DC. Since ε was chosen arbitrarily ($\varepsilon > 0$), then Γ_π can't be DC by Theorem 4.1. $\square$

So Γ_π is ε-DC for $\varepsilon = 0$ but for *no* $\varepsilon > 0$. In summary, the following chain of implications holds on the various DCs:

$$\boxed{[\varepsilon\text{-DC}, \forall \varepsilon \in (0, \hat{\varepsilon}]] \Leftrightarrow \text{DC} \overset{\not\Leftarrow}{\Rightarrow} \pi\text{-DC} \overset{\not\Leftarrow}{\Rightarrow} [\varepsilon\text{-DC, for } \varepsilon = 0]}$$

where $\hat{\varepsilon} \triangleq |\Sigma_P|^{-1} \cdot |V|^{-1}$ as in Theorem 4.1.

4.3.1 The ps-tree: "skeleton" structure for π-dynamic π-ESs

In this subsection we introduce a labelled tree data structure, named the *ps-tree*, which turns out to capture the "skeleton" ordered structure of π-dynamic π-ESs.

Definition 4.5 (PS-Tree). *Let P be any set of boolean variables. A permutation-scenario tree (ps-tree) π_T over P is an outward (non-empty) rooted binary tree such that:*

- *Each node u of π_T is labelled with a letter $p_u \in P$;*

- *All the nodes that lie along a path leading from the root to a leaf are labelled with distinct letters from P.*

- *Each arc (u,v) of π_T is labelled by some $b_{(u,v)} \in \{0,1\}$;*

- *The two arcs (u,v_l) and (u,v_r) exiting a same node u have opposite labels, i.e., $b_{(u,v_l)} \neq b_{(u,v_r)}$.*

Fig. 4.6 depicts an example of a ps-tree.

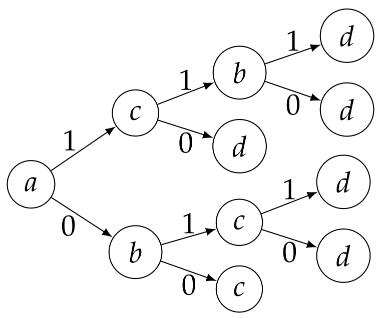


Figure 4.6: An example of a ps-tree over $P = \{a,b,c,d\}$.

Definition 4.6 (π_s, s_i, Coherent-PS-Tree). *Let π_T be a ps-tree over P, let r be the root and s be any leaf. Let $(r, v_2, \dots, s)$ be the sequence of the nodes encountered along the path going from r down to s in π_T. Then:*

- *The sequence of labels $\pi_s = (p_r, p_{v_2}, \dots, p_s)$ is a permutation of the subset of letters $\{p_r, p_{v_2}, \dots, p_s\} \subseteq P$.*

- *Each sequence of bits $(b_{(r,v_2)}, \dots, b_{(v_i,v_{i+1})})$, for each $i \in \{1,2,\dots,k_s - 1\}$ (where $v_1 \triangleq r$ and $v_{k_s} \triangleq s$), can be seen as a partial scenario s_i over P; i.e., define $s_i(v_j) \triangleq b_{(v_j,v_{j+1})}$, for every $j \in \{1,\dots,i\}$.*

- *π_T is* coherent (c-ps-tree) *with Γ if, for every leaf s of π_T,*

$$\{\mathcal{O}_{p_r}, \mathcal{O}_{p_{v_2}}, \dots, \mathcal{O}_{p_s}\} = \mathcal{O}V^+_{s'}$$

  holds for every complete scenario $s' \in \Sigma_P$ such that $\mathrm{Sub}(s', s_{k_s-1})$.

It is not difficult to see that a π-dynamic π-ES induces one and only one c-ps-tree π_T. So, the existence of a suitable c-ps-tree is a necessary condition for a π-ES to be π-dynamic. One may ask whether a π-dynamic π-ES can be reconstructed from its c-ps-tree; the following subsection answers affirmatively.

4.3.2 Verifying a c-ps-tree.

This subsection builds on the notion of c-ps-tree to work out the details of the relationship between π-DC and HyTN-Consistency. Once this picture is in place, it will be easy to reduce to HyTN-Consistency the problem of deciding whether a given CSTN admits a valid π-dynamic π-ES with a given c-ps-tree. This easy result already provides a first combinatorial algorithm for π-DC, though of doubly exponential complexity in $|P|$; a bound to be improved in later subsections, but that can help sizing the sheer dimensionality and depth of the problem.

Firstly, the notion of *Expansion* of CSTNs is recalled [34].

Definition 4.7 (Expansion $\langle V_\Gamma^{\text{Ex}}, \Lambda_\Gamma^{\text{Ex}}\rangle$). *Consider a CSTN $\Gamma = (V, A, L, \mathcal{O}, \mathcal{O}V, P)$. Consider the family of all (distinct) STNs (V_s, A_s), one for each scenario $s \in \Sigma_P$, defined as follows:*

$$V_s \triangleq \{v_s \mid v \in V_s^+\} \text{ and } A_s \triangleq \{(u_s, v_s, w) \mid (u, v, w) \in A_s^+\}.$$

The expansion $\langle V_\Gamma^{Ex}, \Lambda_\Gamma^{Ex}\rangle$ *of the CSTN Γ is defined as follows:*

$$\langle V_\Gamma^{Ex}, \Lambda_\Gamma^{Ex}\rangle \triangleq \Big(\bigcup_{s\in\Sigma_P} V_s, \bigcup_{s\in\Sigma_P} A_s\Big).$$

Notice, $(V_\Gamma^{\text{Ex}}, \Lambda_\Gamma^{\text{Ex}})$ is an STN with at most $|V_\Gamma^{\text{Ex}}| \leq |\Sigma_P| \cdot |V|$ nodes and at most $|\Lambda_\Gamma^{\text{Ex}}| \leq |\Sigma_P| \cdot |A|$ standard arcs.

We now show that the expansion of a CSTN can be enriched with some standard arcs and some hyperarcs in order to model the π-DC property, by means of an HyTN denoted $\mathcal{H}_0^{\pi_T}(\Gamma)$.

Definition 4.8 (HyTN $\mathcal{H}_0^{\pi_T}(\Gamma)$). *Let $\Gamma = (V, A, L, \mathcal{O}, \mathcal{O}V, P)$ be a given CSTN. Let π_T be a given c-ps-tree over P.*

Then, the HyTN $\mathcal{H}_0^{\pi_T}(\Gamma)$ is defined as follows:

- *For every scenarios $s_1, s_2 \in \Sigma_P$ and $u \in V_{s_1,s_2}^+ \setminus \mathcal{O}V$, define a hyperarc $\alpha = \alpha_0(s_1; s_2; u)$ as follows (with the intention to model $H_0(s_1; s_2; u)$, see Definition 4.1):*

  $$\alpha = \alpha_0(s_1; s_2; u) \triangleq \langle t_\alpha, H_\alpha, w_\alpha\rangle,$$

  where:

  - *$t_\alpha \triangleq u_{s_1}$ is the tail of the hyperarc α;*
  - *$H_\alpha \triangleq \{u_{s_2}\} \cup \Delta(s_1; s_2)$ is the set of the heads;*
  - *$w_\alpha(u_{s_2}) \triangleq 0$; $\forall (v \in \Delta(s_1; s_2))\ w_\alpha(v) \triangleq 0$.*

  Now, consider the expansion of the CSTN Γ $\langle V_\Gamma^{Ex}, \Lambda_\Gamma^{Ex}\rangle = \big(\bigcup_{s\in\Sigma_P} V_s, \bigcup_{s\in\Sigma_P} A_s\big)$ (as in Definition 4.7). Then:

- *For each internal node x of π_T, A'_x is a set of (additional) standard arcs defined as follows. Let $\pi_x = (r, \ldots, x')$ be the sequence of all and only the nodes along* *the path going from the root r to the parent x' of x in π_T (where we can assume $r' = r$). Let $P_x \subseteq P$ be the corresponding literals, p_x excluded, i.e., $P_x \triangleq \{p_z \in P \mid z$ appears in π_x and p_z is the label of z in $\pi_T\} \setminus \{p_x\}$. Let s_x be the partial scenario defined as follows:*

$$s_x : P_x \to \{0,1\} : \begin{cases} \lambda\,, & \text{if } x = r; \\ p_z \mapsto b_{(z,z')}, & \text{if } x \neq r. \end{cases}$$

where z' is the unique child of z in π_T lying on π_x. Let x_0 (x_1) be the unique child of x in π_T such that $b_{x,x_0} = 0$ ($b_{x,x_1} = 1$). For every complete $s'_x \in \Sigma_P$ such that $\mathrm{Sub}(s'_x, s_x)$, we define:

$$B'_{s'_x} \triangleq \begin{cases} \{\langle (\mathcal{O}_{p_{x_0}})_{s'_x}, (\mathcal{O}_{p_x})_{s'_x}, 0\rangle\}, & \text{if } s'_x(x) = 0; \\ \{\langle (\mathcal{O}_{p_{x_1}})_{s'_x}, (\mathcal{O}_{p_x})_{s'_x}, 0\rangle\}, & \text{if } s'_x(x) = 1. \end{cases}$$

Also, for every complete $s'_x, s''_x \in \Sigma_P$ such that $\mathrm{Sub}(s'_x, s_x)$ and $\mathrm{Sub}(s''_x, s_x)$, where $s'_x \neq s''_x$, we define: $C'_{s'_x, s''_x} \triangleq \{\langle (\mathcal{O}_{p_x})_{s'_x}, (\mathcal{O}_{p_x})_{s''_x}, 0\rangle\}$.

Finally,

$$A'_x \triangleq \bigcup_{s'_x \in \Sigma_P \,:\, \mathrm{Sub}(s'_x, s_x)} B'_{s'_x} \cup \bigcup_{\substack{s'_x, s''_x \in \Sigma_P \,:\, s'_x \neq s''_x, \\ \mathrm{Sub}(s'_x, s_x), \mathrm{Sub}(s''_x, s_x)}} C'_{s'_x, s''_x}.$$

- *Then, $\mathcal{H}_0^\pi(\Gamma)$ is defined as $\mathcal{H}_0^\pi(\Gamma) \triangleq \langle V_\Gamma^{Ex}, \mathcal{A}_{\mathcal{H}_0^\pi(\Gamma)}\rangle$, where,*

$$\mathcal{A}_{\mathcal{H}_0^\pi(\Gamma)} \triangleq \Lambda_\Gamma^{Ex} \cup \bigcup_{\substack{s_1, s_2 \in \Sigma_P \\ u \in V^+_{s_1, s_2}}} \alpha_\varepsilon(s_1; s_2; u) \cup \bigcup_{\substack{x \,:\, \textit{internal} \\ \textit{node of } \pi_T}} A'_x.$$

Notice that the following holds: each $\alpha_\varepsilon(s_1; s_2; u)$ has size:

$$|\alpha_\varepsilon(s_1; s_2; u)| = |\Delta(s_1; s_2)| + 1 \leq |P| + 1.$$

The following theorem establishes the connection between the π-DC of CSTNs and the consistency of HyTNs.

Theorem 4.3. *Given any CSTN $\Gamma = \langle V, A, L, \mathcal{O}, \mathcal{O}V, P\rangle$, it holds that the CSTN Γ is π-DC if and only if there exists a c-ps-tree π_T such that the HyTN $\mathcal{H}_0^{\pi_T}(\Gamma)$ is consistent.*

Moreover, $\mathcal{H}_0^{\pi_T}(\Gamma)$ has at most so many nodes:

$$|V_{\mathcal{H}_0^{\pi_T}(\Gamma)}| \leq |\Sigma_P| \cdot |V|,$$

so many hyperarcs:

$$|\mathcal{A}_{\mathcal{H}_0^{\pi_T}(\Gamma)}| = O(|\Sigma_P| \cdot |A| + |\Sigma_P|^2 |V|),$$

and it has size at most:

$$m_{\mathcal{A}_{\mathcal{H}_0^{\pi_T}(\Gamma)}} = O(|\Sigma_P| \cdot |A| + |\Sigma_P|^2 |V| \cdot |P|).$$

Proof. (1) Firstly, we prove that the CSTN Γ is π-DC if and only if there exists a c-ps-tree π_T such that the HyTN $\mathcal{H}_0^{\pi_T}(\Gamma)$ is consistent.

($\Rightarrow$) Let $\sigma \in \mathcal{S}_\Gamma$ be a given viable and π-dynamic execution strategy for the CSTN Γ. Since σ is π-dynamic, then for any two $s_1, s_2 \in \Sigma_P$ and any $v \in V^+_{s_1,s_2}$ the following holds on the execution time $\tau \triangleq [\sigma(s_1)]^t_v$ and position $\psi \triangleq [\sigma(s_1)]^\pi_v$:

$$Con(\pi\text{-}Hst(\tau, \psi, s_1, \sigma), s_2) \Rightarrow [\sigma(s_2)]^t_v = \tau, [\sigma(s_2)]^\pi_v = \psi.$$

It is easy to see that this induces one and only one c-ps-tree π_T: indeed, due to Remark 4.1, there must be exactly one $\mathcal{O}_{p'} \in \mathcal{O}V$, for some $p' \in P$, which is executed at first (w.r.t. to both execution time and position) under all possible scenarios; then, depending on the boolean result of p', a second observation p'' can be differentiated, and it can occur at the same or at a subsequent time instant, but still at a subsequent position; again, by Remark 4.1, there is exactly one $\mathcal{O}_{p''} \in \mathcal{O}V$ which comes first under all possible scenarios that agree on p'; and so on and so forth, thus forming a tree structure over P, rooted at p', which is captured exactly by our notion of c-ps-tree. Then, let $\phi_\sigma : V^{\text{Ex}}_\Gamma \to \mathbb{R}$ be the schedule of $\mathcal{H}_0^{\pi_T}(\Gamma)$ defined as: $\phi_\sigma(v_s) \triangleq [\sigma(s)]^t_v$ for every $v_s \in V^{\text{Ex}}_\Gamma$, where $s \in \Sigma_P$ and $v \in V^+_s$. It is not difficult to check from the definitions, at this point, that all of the standard arc and hyperarc constraints of $\mathcal{H}_0^{\pi_T}(\Gamma)$ are satisfied by ϕ_σ, that is to say that ϕ_σ must be feasible for $\mathcal{H}_0^{\pi_T}(\Gamma)$. Hence, $\mathcal{H}_0^{\pi_T}(\Gamma)$ is consistent.

($\Leftarrow$) Assume that there exists a c-ps-tree π_T such that the HyTN $\mathcal{H}_0^{\pi_T}(\Gamma)$ is consistent, and let $\phi : V^{\text{Ex}}_\Gamma \to \mathbb{R}$ be a feasible schedule for $\mathcal{H}_0^{\pi_T}(\Gamma)$. Then, let $\sigma_{\phi,\pi_T}(s) \in \mathcal{S}_\Gamma$ be the execution strategy defined as follows:

- $[\sigma_{\phi,\pi_T}(s)]^t_v \triangleq \phi(v_s)$, $\forall\ v_s \in V^{\text{Ex}}_\Gamma$, $s \in \Sigma_P$, $v \in V^+_s$;
- Let $s' \in \Sigma_P$ be any complete scenario. Then, s' induces exactly one path in π_T, in a natural way, i.e., by going from the root r down to some leaf s. Notice that the sequence of labels $(p_r, p_{v_2}, \ldots, p_s)$ can be seen as a bijection, i.e., $\pi_s : \mathcal{O}V^+_{s'} \rightleftharpoons \{1, \ldots, |\mathcal{O}V^+_{s'}|\}$. Then, for any $s' \in \Sigma_P$ and $v \in \mathcal{O}V^+_{s'}$, define $[\sigma_{\phi,\pi_T}(s')]^\pi_v \triangleq \pi_s(v)$.

It is not difficult to check from the definitions, at this point, that since ϕ is feasible for $\mathcal{H}_0^{\pi_T}(\Gamma)$, then σ_{ϕ,π_T} must be viable and π-dynamic for the CSTN Γ. Hence, the CSTN Γ is π-DC.

(2) The size bounds for $\mathcal{H}_0^{\pi_T}(\Gamma)$ follow from Definition 4.8. □

In Algo. 10, it is offered the pseudocode for constructing the HyTN $\mathcal{H}_0^{\pi_T}(\Gamma)$, as prescribed by Definition 4.8.

If Γ is π-DC, there is an integer weighted π-dynamic π-ES, as below.

Algorithm 9: `construct_`$\mathcal{H}(\Gamma,\pi_T)$

```
Input: a CSTN $\Gamma \triangleq \langle V,A,L,\mathcal{O},\mathcal{O}V,P\rangle$, a c-ps-tree $\pi_T$ coherent with $\Gamma$.
1  foreach ($s \in \Sigma_P$) do
2      $V_s \leftarrow \{v_s \mid v \in V_s^+\}$;
3      $A_s \leftarrow \{a_s \mid a \in A_s^+\}$;
4  $V_\Gamma^{\text{Ex}} \leftarrow \cup_{s\in\Sigma_P} V_s$;
5  $\Lambda_\Gamma^{\text{Ex}} \leftarrow \cup_{s\in\Sigma_P} A_s$;
6  foreach ($s_1,s_2 \in \Sigma_P$, $s_1 \neq s_2$) do
7      foreach ($u \in V_{s_1,s_2}^+ \setminus \mathcal{O}V$) do
8          $t_\alpha \leftarrow u_{s_1}$;
9          $H_\alpha \leftarrow \{u_{s_2}\} \cup (\Delta(s_1;s_2))$;
10         $w_\alpha(u_{s_2}) \leftarrow 0$;
11         foreach $v \in \Delta(s_1;s_2)$ do
12             $w_\alpha(v_{s_1}) \leftarrow 0$;
13         $\alpha_0(s_1;s_2;u) \leftarrow \langle t_\alpha,H_\alpha,w_\alpha\rangle$;
14 foreach ($x$ : internal node of $\pi_T$) do
15     $A'_x \leftarrow$ as defined in Definition 4.8;
16 $\mathcal{A}_{\mathcal{H}_0^{\pi_T}(\Gamma)} \leftarrow \Lambda_\Gamma^{\text{Ex}} \cup \bigcup_{s_1,s_2\in\Sigma_P,\ u\in V_{s_1,s_2}^+} \alpha_0(s_1;s_2;u) \cup \bigcup_{x\text{ : internal node of }\pi_T} A'_x$;
17 $\mathcal{H}_0^{\pi_T}(\Gamma) \leftarrow \langle V_\Gamma^{\text{Ex}}, \mathcal{A}_{H_0^{\pi_T}(\Gamma)}\rangle$;
18 return $\mathcal{H}_0^{\pi_T}(\Gamma)$;
```

Algorithm 9: Constructing the HyTN $\mathcal{H}_0^{\pi_T}(\Gamma)$.

Proposition 4.5. *Assume $\Gamma = \langle V,A,L,\mathcal{O},\mathcal{O}V,P\rangle$ to be π-DC. Then, there is some π-ES $\sigma \in \mathcal{S}_\Gamma$ which is viable, π-dynamic, and* integer weighted, *namely, for every $s \in \Sigma_P$ and every $v \in V_s^+$, the following property holds:*

$$[\sigma(s)]_v^t \in \{0,1,2,\ldots,\mathcal{M}_\Gamma\} \subseteq \mathbb{N},$$

where $\mathcal{M}_\Gamma \triangleq \big(|\Sigma_P||V| + |\Sigma_P||A| + |\Sigma_P|^2|V|\big)W$.

Proof. By Theorem 4.3, since Γ is π-DC, there exists some c-ps-tree π_T such that the HyTN $\mathcal{H}_0^{\pi_T}(\Gamma)$ is consistent; moreover, by Theorem 4.3 again, $\mathcal{H}_0^{\pi_T}(\Gamma)$ has $|V_{\mathcal{H}_0^{\pi_T}(\Gamma)}| \le |\Sigma_P||V|$ nodes and $|\mathcal{A}_{\mathcal{H}_0^{\pi_T}(\Gamma)}| \le |\Sigma_P||A| + |\Sigma_P|^2|V|$ hyperarcs. Since $\mathcal{H}_0^{\pi_T}(\Gamma)$ is consistent, it follows from Theorem 4.3 [also see Lemma 1 and Theorem 8 in [33]] that $\mathcal{H}_0^{\pi_T}(\Gamma)$ admits an integer weighted and feasible schedule ϕ such that:

$$\phi : V_{\mathcal{H}_0^{\pi_T}(\Gamma)} \to \{0,1,2,\ldots,\mathcal{M}_\Gamma\},$$

where $\mathcal{M}_\Gamma \le (|V_{\mathcal{H}_0^{\pi_T}(\Gamma)}| + |\mathcal{A}_{\mathcal{H}_0^{\pi_T}(\Gamma)}|)W$.

Therefore, it holds that $\mathcal{M}_\Gamma \le (|\Sigma_P||V| + |\Sigma_P||A| + |\Sigma_P|^2|V|)W$. $\square$

Given a CSTN Γ and some c-ps-tree π_T, it is thus easy to check whether there exists some π-ES for Γ whose ordering relations are exactly the same as those prescribed by π_T. Indeed, it is sufficient to construct $\mathcal{H}_0^{\pi_T}(\Gamma)$ with Algorithm 9, then checking the consistency of $\mathcal{H}_0^{\pi_T}(\Gamma)$ with the algorithm mentioned in Theorem 2.7. This results into Algorithm 10. The corresponding time complexity is also that of Theorem 2.7.

Algorithm 10: `check_`π`-DC_on_c-ps-tree`(Γ, π_T)

```
Input: a CSTN $\Gamma \triangleq \langle V, A, L, \mathcal{O}, \mathcal{O}V, P\rangle$, a c-ps-tree $\pi_T$ coherent with $\Gamma$.
1 $\mathcal{H}_0^{\pi_T}(\Gamma) \leftarrow$ construct_$\mathcal{H}(\Gamma, \pi_T)$; // ref. Algorithm 9
2 $\phi \leftarrow$ check_HEAD-HYTN-CONSISTENCY($\mathcal{H}_0^{\pi_T}(\Gamma)$); // ref. Thm 2.7
3 if ($\phi$ is a feasible schedule of $\mathcal{H}_0^{\pi_T}(\Gamma)$) then
4     return $\langle YES, \phi, \pi_T\rangle$;
5 return NO;
```

Algorithm 10: Checking π-DC given a c-ps-tree π_T, by reduction to Head-HyTN-Consistency.

Notice that, in principle, one could generate all of the possible c-ps-trees π_T given P, one by one, meanwhile checking for the consistency state of $\mathcal{H}_0^{\pi_T}(\Gamma)$ with Algorithm 10. However, it is not difficult to see that, in general, the total number $f_{|P|}$ of possible c-ps-trees over P is not singly-exponential in $|P|$. Indeed, a moment's reflection revelas that for every $n > 1$ it holds that $f_n = n \cdot f_{n-1}^2$, and $f_1 = 1$. So, any algorithm based on the exhaustive exploration of the whole space comprising all of the possible c-ps-trees over P would not have a (pseudo) singly-exponential time complexity in $|P|$. Nevertheless, we have identified another solution, that allows us to provide a sound-and-complete (pseudo) singly-exponential time π-DC-Checking procedure: it is a simple and self-contained reduction from π-DC-Checking to DC-Checking. This allows us to provide the first sound-and-complete (pseudo) singly-exponential time π-DC-Checking algorithm which employs our previous DC-Checking algorithm (i.e., that underlying Theorem 4.2) in a direct manner, as a black box, thus avoiding a more fundamental restructuring of it.

4.3.3 A Singly-Exponential Time π-DC-Checking Algorithm

This section presents a sound-and-complete (pseudo) singly-exponential time algorithm for solving π-DC, also producing a viable and π-dynamic π-ES whenever the input CSTN is really π-DC. The main result of this section goes as follows.

Theorem 4.4. *There exists an algorithm for checking π-DC on any input given CSTN $\Gamma = (V, A, L, \mathcal{O}, \mathcal{O}V, P)$ with the following (pseudo) singly-exponential time complex-*

ity:

$$O\Big(|\Sigma_P|^4|A|^2|V|^3 + |\Sigma_P|^5|A||V|^4|P| + |\Sigma_P|^6|V|^5|P|\Big)W.$$

Moreover, when Γ is π-DC, the algorithm also returns a viable and π-dynamic π-ES for Γ. Here, $W \triangleq \max_{a\in A}|w_a|$.

The algorithm mentioned in Theorem 4.4 consits of a simple reduction from π-DC to (classical) DC in CSTNs.

Basically, the idea is to give a small margin γ so that the planner can actually do before, in the sense of the time value $[\sigma(s)]_v$, what he did "before" in the ordering π. Given any ES in the relaxed network, the planner would then turn it into a π-ES for the original network (which has some more stringent constraints), by rounding-down each time value $[\sigma(s)]_v$ to the largest integer less than or equal to it, i.e., $\lfloor[\sigma(s)]_v\rfloor$. The problem is that one may (possibly) violate some constraints when there is a "leap" in the rounding (i.e., a difference of one unit, in the rounded value, w.r.t. what one would have wanted). Anyhow, we have identified a technique that allows us to get around this subtle case, provided that γ is exponentially small.

Definition 4.9. *Relaxed CSTN Γ'. Let $\Gamma = \langle V, A, L, \mathcal{O}, \mathcal{O}V, P\rangle$ be any CSTN with integer constraints. Let $\gamma \in (0,1)$ be a real. Define $\Gamma'_\gamma \triangleq \langle V, A'_\gamma, L, \mathcal{O}, \mathcal{O}V, P\rangle$ to be a CSTN that differs from Γ only in the numbers appearing in the constraints. Specifically, each constraint $\langle u - v \leq \delta, \ell\rangle \in A$ is replaced in Γ'_γ by a slightly relaxed constraint, $\langle u - v \leq \delta'_\gamma, \ell\rangle \in A'_\gamma$, where:*

$$\delta'_\gamma \triangleq \delta + |V|\cdot\gamma.$$

The following two lemmata hold for any CSTN Γ.

Lemma 4.1. *Let γ be any real in $(0, |V|^{-1})$.*

If Γ is π-DC, then Γ'_γ is DC.

Proof. Since Γ is π-DC, by Proposition 4.5, there exists an integer weighted, viable and π-dynamic, π-ES σ for Γ. Let us fix some real $\gamma \in (0, |V|^{-1})$. Define the ES $\sigma'_\gamma \in \mathcal{S}_{\Gamma'_\gamma}$ as follows, for every $s \in \Sigma_P$ and $v \in V_s^+$:

$$[\sigma'_\gamma(s)]_v \triangleq [\sigma(s)]^t_v + [\sigma(s)]^\pi_v \cdot \gamma.$$

Since $[\sigma(s)]^\pi_v \leq |V|$, then:

$$[\sigma(s)]^\pi_v \cdot \gamma < |V| \cdot |V|^{-1} = 1,$$

and so the total ordering of the values $[\sigma'_\gamma(s)]_v$, for a given $s \in \Sigma_P$, coincides with $[\sigma(s)]^\pi$. Hence, the fact that σ'_γ is dynamic follows directly from the π-dynamicity of σ. Moreover, no LTC $(u - v \leq \delta'_\gamma, \ell)$ of Γ'_γ is violated in any scenario $s \in \Sigma_P$ since, if $\Delta'_{\gamma,u,v} \triangleq [\sigma'_\gamma(s)]_u - [\sigma'_\gamma(s)]_v$ then:

$$\begin{aligned}\Delta'_{\gamma,u,v} &= \big([\sigma(s)]^t_u + [\sigma(s)]^\pi_u \cdot \gamma\big) - \big([\sigma(s)]^t_v + [\sigma(s)]^\pi_v \cdot \gamma\big) \\ &\le [\sigma(s)]^t_u - [\sigma(s)]^t_v + |V| \cdot \gamma \\ &\le \delta + |V| \cdot \gamma = \delta'.\end{aligned}$$

So, σ'_γ is viable. Since σ'_γ is also dynamic, then Γ'_γ is DC. □

The next lemma shows that the converse direction holds as well, but for (exponentially) smaller values of γ.

Lemma 4.2. *Let γ be any real in $(0, |\Sigma_P|^{-1} \cdot |V|^{-2})$.*
If Γ'_γ is DC, then Γ is π-DC.

Proof. Let $\sigma'_\gamma \in \mathcal{S}_{\Gamma'_\gamma}$ be some viable and dynamic ES for Γ'_γ.
Firstly, we aim at showing that, w.l.o.g., the following lower bound holds:

$$[\sigma'_\gamma(s)]_v - \lfloor [\sigma'_\gamma(s)]_v \rfloor \ge |V| \cdot \gamma, \text{ for } \textit{all } s \in \Sigma_P \text{ and } v \in V^+_s. \qquad \text{(LB)}$$

This will allow us to simplify the rest of the proof. In order to prove it, let us pick any $\eta \in [0,1)$ such that:

$$[\sigma'_\gamma(s)]_v - \eta - k \in [0, |V| \cdot \gamma), \text{ for } \textit{no } v \in V, s \in \Sigma_P, k \in \mathbb{Z}.$$

Observe that such a value η exists. Indeed, there are only $|\Sigma_P| \cdot |V|$ choices of pairs $(s,v) \in \Sigma_P \times V$ and each pair rules out a (circular) semi-open interval of length $|V| \cdot \gamma$ in $[0,1)$, so the total measure of invalid values for η in the semi-open real interval $[0,1)$ is at most $|\Sigma_P| \cdot |V| \cdot |V| \cdot \gamma < 1$. So η exists.
See Fig. 4.7 for an intuitive illustration of this fact.

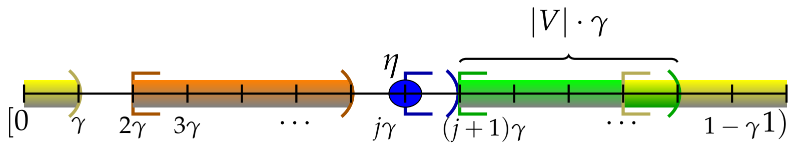


Figure 4.7: An illustration of the proof of Lemma 4.2.

By subtracting η to all time values $\{[\sigma'_\gamma(s)]_v\}_{v \in V, s \in \Sigma_P}$ we can assume w.l.o.g. that $\eta = 0$ holds for the rest of this proof; and thus, that (LB) holds. Now, define $[\sigma(s)]^t_v \triangleq \lfloor [\sigma'_\gamma(s)]_v \rfloor$, and let $[\sigma(s)]^\pi$ be the ordering induced by $\sigma'_\gamma(s)$. Observe that σ is a well-defined π-ES (i.e., that $[\sigma(s)]^\pi$ is coherent w.r.t. $[\sigma(s)]^t$), thanks to the fact that $\lfloor \cdot \rfloor$ is a monotone operator. Since the ordering $[\sigma(s)]^\pi$ is the same as that of $\sigma'_\gamma(s)$, then σ is π-dynamic.

It remains to prove that σ is viable. For this, take any constraint $(u - v \leq \delta, \ell) \in A$ in Γ, and suppose that:

$$[\sigma'_\gamma(s)]_u - [\sigma'_\gamma(s)]_v \leq \delta'_\gamma = \delta + |V| \cdot \gamma. \quad \text{(A)}$$

If $[\sigma'_\gamma(s)]_u - [\sigma'_\gamma(s)]_v \leq \delta$, then clearly $[\sigma(s)]^t_u - [\sigma(s)]^t_v \leq \delta$. So, the interesting case that we really need to check is when:

$$0 < [\sigma'_\gamma(s)]_u - [\sigma'_\gamma(s)]_v - \delta \leq |V| \cdot \gamma.$$

For this, we observe that the following $(*)$ holds by (LB):

$$\lfloor [\sigma'_\gamma(s)]_u \rfloor \leq [\sigma'_\gamma(s)]_u - |V| \cdot \gamma. \quad (*)$$

Also, it is clear that:

$$\lfloor [\sigma'_\gamma(s)]_v \rfloor > [\sigma'_\gamma(s)]_v - 1. \qquad (**)$$

Then,

$$\begin{aligned}
[\sigma(s)]^t_u - [\sigma(s)]^t_v &= \lfloor [\sigma'_\gamma(s)]_u \rfloor - \lfloor [\sigma'_\gamma(s)]_v \rfloor && \text{(by def. of } [\sigma(s)]^t_x,\ x \in \{u,v\}) \\
&< ([\sigma'_\gamma(s)]_u - |V| \cdot \gamma) - ([\sigma'_\gamma(s)]_v - 1) && \text{(by } (*) \text{ and } (**)) \\
&\leq ([\sigma'_\gamma(s)]_u - [\sigma'_\gamma(s)]_v) - |V| \cdot \gamma + 1 && \text{(by rewriting)} \\
&\leq \delta'_\gamma - |V| \cdot \gamma + 1 && \text{(by (A))} \\
&\leq \delta + 1. && \text{(by } \delta'_\gamma = \delta + |V| \cdot \gamma)
\end{aligned}$$

Now, since we have the strict inequality $[\sigma(s)]^t_u - [\sigma(s)]^t_v < \delta + 1$, and since $[\sigma(s)]^t_u - [\sigma(s)]^t_v \in \mathbb{Z}$, then $[\sigma(s)]^t_u - [\sigma(s)]^t_v \leq \delta$, as desired. So, σ is viable. Since σ is both viable and π-dynamic, then Γ is π-DC. $\square$

Fig. 4.7 illustrates the proof of Lemma 4.2, in which a family of (circular) semi-open intervals of length $|V| \cdot \gamma$ are depicted as shaded rectangles. Lemma 4.2 ensures that at least one chunk on length $l_\gamma \geq 1 - |\Sigma_P| \cdot |V|^2 \cdot \gamma$ is not covered by the union of those (circular) semi-open intervals, and it is therefore free to host η; in Fig. 4.7, this is represented by the blue interval, and $\eta = j \cdot \gamma$ for some $j \in [0, \gamma^{-1})$. Also notice that γ can be fixed as follows:

$$\gamma \triangleq \frac{1}{|\Sigma_P| \cdot |V|^2 + 1};$$

then, $l_\gamma \geq |\Sigma_P|^{-1} \cdot |V|^{-2}$.

In summary, Lemma 4.1 and Lemma 4.2 imply Theorem 4.5.

Theorem 4.5. *Let Γ be a CSTN and let $\gamma \in (0, |\Sigma_P|^{-1} \cdot |V|^{-2})$.*
Then, Γ is π-DC if and only if Γ'_γ is DC.

This allows us to design a simple algorithm for solving π-DC-Checking, by reduction to DC-Checking, which is named `Check-`π`-DC()` (Algorithm 11). Its pseudocode follows below.

Algorithm 11: `Check-`π`-DC`(Γ)

```
Input: a CSTN $\Gamma \triangleq \langle V, A, L, \mathcal{O}, \mathcal{OV}, P\rangle$
$\gamma \leftarrow \frac{1}{|\Sigma_P|\cdot|V|^2+1}$;
$A'_\gamma \leftarrow \big\{\langle u - v \leq \delta + |V|\cdot\gamma, \ell\rangle \mid \langle u - v \leq \delta, \ell\rangle \in A\big\}$;
$\Gamma'_\gamma \leftarrow \langle V, A'_\gamma, L, \mathcal{O}, \mathcal{OV}, P\rangle$;
$\sigma'_\gamma \leftarrow$ check_DC$(\Gamma'_\gamma)$; // see Theorem 4.2
if $\sigma'_\gamma$ is a viable and dynamic ES for $\Gamma'_\gamma$ then
    $\eta \leftarrow$ pick $\eta \in [0,1)$ as in the proof of Lemma 4.2;
    foreach $(s,v) \in \Sigma_P \times V_s^+$ do
        $[\sigma'_\gamma(s)]_v \leftarrow [\sigma'_\gamma(s)]_v - \eta$; // shift by $\eta$;
    let $\sigma \in \Sigma_\Gamma$ be constructed as follows;
    foreach $s \in \Sigma_P$ do
        foreach $v \in V_s^+$ do
            $[\sigma(s)]^t_v \leftarrow \big\lfloor [\sigma'_\gamma(s)]_v \big\rfloor$;
        $[\sigma(s)]^\pi \leftarrow$ the ordering on $P$ induced by $\sigma'_\gamma(s)$;
    return $\langle$YES$,\sigma\rangle$;
return NO;
```

Algorithm 11: Checking π-DC by reduction to DC-Checking.

Description of Algorithm 11 It takes in input a CSTN Γ. When Γ is π-DC, it aims at returning $\langle$`YES`$,\sigma\rangle$, where $\sigma \in \mathcal{S}_\Gamma$ is a viable and π-dynamic π-ES for Γ. Otherwise, if Γ is not π-DC, then `Check-`π`-DC()` (Algorithm 11) returns `NO`. Of course the algorithm implements the reduction described in Definition 4.9, whereas the π-ES is computed as prescribed by Lemma 4.2. At line 1, we set $\gamma \leftarrow \frac{1}{|\Sigma_P|\cdot|V|^2+1}$. Then, at lines 2-3, Γ'_γ is constructed as in Definition 4.9, i.e., $\Gamma'_\gamma \leftarrow \langle V, A'_\gamma, L, \mathcal{O}, \mathcal{OV}, P\rangle$, where $A'_\gamma \leftarrow \big\{\langle u - v \leq \delta + |V|\cdot\gamma, \ell\rangle \mid \langle u - v \leq \delta, \ell\rangle \in A\big\}$. At this point, at line 5, the DC-Checking algorithm of Theorem 4.2 is invoked on input Γ'_γ. Let σ'_γ be its output. If Γ'_γ is not DC, then `Check-`π`-DC()` (Algorithm 11) returns `NO` at line 15. When σ'_γ is a viable and dynamic ES for Γ'_γ at line 5, then `Check-`π`-DC()` (Algorithm 11) proceeds as follows. At line 6, some $\eta \in [0,1)$ is computed as in the proof of Lemma 4.2, i.e., such that $[\sigma'_\gamma(s)]_v - \eta - k \in [0, |V|\cdot\gamma)$ holds for *no* $v \in V, s \in \Sigma_P, k \in \mathbb{Z}$. Notice that it is easy to find such η in practice. Indeed, one may view the real semi-open interval $[0,1)$ as if it was partitioned into chunks (i.e., smaller semi-open intervals) of length γ; as observed in the proof of Lemma 4.2, there are only $|\Sigma_P|\cdot|V|$ choices of pairs $(s,v) \in \Sigma_P \times V$, and each pair rules out a (circular) semi-open interval of length $|V|\cdot\gamma$; therefore, there is at least one chunk of length $l_\gamma \geq |\Sigma_P|^{-1} \cdot |V|^{-2}$, within $[0,1)$, where η can be placed, and we can easily find it just by inspecting (exhaustively) the pairs $(s,v) \in \Sigma_P \times V$. In fact, the algorithm underlying Theorem 4.2 always deliver an earliest-ES (i.e., one in which the time values are the smallest possible, in the space of all consistent ESs), so that for each interval of length $|V| \cdot \gamma$, the only time values that we really need to check and rule out are $|V|$ multiples of γ. Therefore, at line 6, η exists and it can be easily found in time $O(|\Sigma_P| \cdot |V|^2)$. So, at line 7, for each $s \in \Sigma_P$ and $v \in V_s^+$, the value $[\sigma'_\gamma(s)]_v$ is shifted to the left by setting $[\sigma'_\gamma(s)]_v \leftarrow [\sigma'_\gamma(s)]_v - \eta$. Then, the following π-ES $\sigma \in \mathcal{S}_\Gamma$ is constructed at lines 9-13: for each $s \in \Sigma_P$ and $v \in V_s^+$, the execution-time is set $[\sigma(s)]_v^t \leftarrow \lfloor [\sigma'_\gamma(s)]_v \rfloor$, and the ordering $[\sigma(s)]^\pi$ follows the ordering on P that is induced by $\sigma'_\gamma(s)$. Finally, $\langle \texttt{YES}, \sigma \rangle$ is returned to output at line 14.

To conclude, we can prove the main result of this section.

Proof of Theorem 4.4. The correctness of Algorithm 11 follows directly from Theorems 4.5 and 4.2, plus the fact that $\eta \in [0,1)$ can be computed easily, at line 6, as we have already mentioned above. The (pseudo) singly-exponential time complexity of Algorithm 11 follows from that of Theorem 4.2 plus the fact that all the integer weights in Γ are scaled-up by a factor $1/\gamma = |\Sigma_P| \cdot |V|^2 + 1$ in Γ'_γ; also notice that $\eta \in [0,1)$ can be computed in time $O(|\Sigma_P| \cdot |V|^2)$, as we have already mentioned. Therefore, all in, the time complexity stated in Theorem 4.2 increases by a factor $1/\gamma = |\Sigma_P| \cdot |V|^2 + 1$. □

4.4 Related Works

This section discusses of some related approaches offered in the current literature. The article of Tsamardinos, *et al.* [113] introduced DC for CSTNs. Subsequently, this notion has been analyzed and further formalized in [67], finally leading to a sound notion of DC for CSTNs. However, neither of these two works takes into account an instantaneous reaction-time. Cimatti, *et al.* [21] provided the first sound-and-complete procedure for checking the Dynamic-Controllability of CSTNs with Uncertainty (CSTNUs) and this algorithm can be employed for checking DC on CSTNs as a special case. Their approach is based on reducing the problem to solving Timed Game Automata (TGA). However, solving TGAs is a problem of much higher complexity than solving MPGs. Indeed, no upper bound is given in [21] on the time complexity of their solution. Moreover, neither ε-DC nor any other notion of DC with an instantaneous reaction-time are dealt with in that work. The first work to approach a notion of DC with an instantaneous reaction-time is [69]; its aim was to offer a sound-and-complete propagation-based DC-checking algorithm for CSTNs. The subsequent work [68] extended and amended [69] so that to check ε-DC, both for $\varepsilon > 0$ and for $\varepsilon = 0$. However, to the best of our knowledge, the worst-case complexity of those algorithms is currently unsettled. Moreover, it is not clear to us how one variant of the algorithm offered in [68, 69] (i.e., the one that aims at checking DC with an instantaneous reaction-time) can adequately handle cases like the CSTN counter-example $\Gamma_{\Box}$ that we have provided in Example 4.1. In summary, we believe that the present work can possibly help in clarifying DC with an instantaneous reaction-time also when the perspective had to be that of providing sound-and-complete algorithms based on the propagation of labelled temporal constraints.

4.5 Conclusion

The notion of ε-DC has been introduced and analysed in [34] where an algorithm was also given to check whether a CSTN is ε-DC. By the interplay between ε-DC and the standard notion of DC, also disclosed in [34], this delivered the first (pseudo) singly-exponential time algorithm checking whether a CSTN is DC (essentially, DC-Checking reduces to ε-DC-Checking for a suitable value of ε). In this chapter, we proposed and formally defined π-DC, a natural and sound notion of DC for CSTNs in which the planner is allowed to react instantaneously to the observations that are made during the execution. A neat counter-example shows that π-DC with instantaneous reaction-time is not just the special case of ε-DC with $\varepsilon = 0$. Therefore, to conclude, we offer the first sound-and-complete π-DC-Checking algorithm for CSTNs. The time complexity of the procedure is still (pseudo) singly-exponential in $|P|$. The solution is based on a simple reduction from π-DC-Checking to DC-Checking of CSTNs.

Part II

Infinite Games on Graphs

5 Linear Time Algorithm for Update Games via Strongly-Trap-Connected Components

Chapter Abstract

An arena is a finite directed graph whose vertices are divided into two classes, i.e., $V = V_{\Box} \cup V_{\bigcirc}$; this forms the basic playground for a plethora of 2-player infinite pebble games. In this chapter, we introduce and study a refined notion of reachability for arenas, named *trap-reachability*, where Player $\Box$ attempts to reach vertices without leaving a prescribed subset $U \subseteq V$, while Player $\bigcirc$ works against. It is shown that every arena decomposes into *strongly-trap-connected components (STCCs)*. Our main result is a linear time algorithm for computing this unique decomposition. Both the graph structures and the algorithm generalize the classical decomposition of a directed graph into its strongly-connected components (SCCs). The algorithm builds on a generalization of the depth-first search (DFS), taking inspiration from Tarjan's SCCs classical algorithm. The structures of palm-trees and jungles described in Tarjan's original paper need to be revisited and generalized (i.e., tr-palm-trees and tr-jungles) in order to handle the 2-player infinite pebble game's setting.

This theory has direct applications in solving Update Games (UGs) faster. Dinneen and Khoussainov showed in 1999 that deciding who's the winner in a given UG costs $O(mn)$ time, where n is the number of vertices and m is that of arcs. We solve that problem in $\Theta(m + n)$ linear time. The result is obtained by observing that the UG is a win for Player $\Box$ if and only if the arena comprises one single STCC. It is also observed that the tr-palm-tree returned by the algorithm encodes routing information that an $\Theta(n)$-space agent can consult to win the UG in $O(1)$ time per move. Finally, the polynomial-time complexity for deciding Explicit McNaughton-Müller Games is also improved, from cubic to quadratic.
This chapter is a revised version of [39].

5.1 Introduction

In the construction of reactive systems, like communication protocols or control systems, a central aim is to put the development of hardware and software on a mathematical basis which is both firm and practical. A characteristic feature of such systems is their perpetual interaction with the environment as well as their non-terminating behaviour. The theory of infinite duration games offers many appealing results under this prospect [61]. For instance, consider the following communication network problem. Suppose we have data stored on each node of a network and we want to continuously update all nodes with consistent data: often one requirement is to share key information between all nodes of a network, this can be done by having a data packet of current information continuously going through all nodes. Unfortunately not all routing choices are always under our control, some of them may be controlled by the network environment, which could play against us. This is essentially an infinite 2-player pebble game played on an *arena*, i.e., a finite directed simple graph in which the vertices are divided into two classes, i.e., $V_\Box$ and $V_\bigcirc$, where Player $\Box$ wants to visit all vertices infinitely often by moving the pebble on them, while Player $\bigcirc$ works against. This is called *Update Game (UG)* in [10,45,46]. Dinneen and Khoussainov [45] showed that deciding who's the winner in a given UG costs $O(mn)$ time, where n is the number of vertices and m is that of the arcs.

Contribution and Organization. In Section 5.6, as a main result, the same problem of deciding who's the winner in a given UG is solved in $\Theta(m+n)$ linear time; it is also observed that the graph structure returned by the algorithm encodes routing information that an $\Theta(n)$-space agent can consult to win the UG in $O(1)$ time per move. For this, in Section 5.2, we introduce and study a refined notion of reachability for arenas, named *trap-reachability*, where Player $\Box$ attempts to reach vertices without leaving a prescribed subset $U \subseteq V$, while Player $\bigcirc$ works against. In Section 5.3, it is shown that every arena decomposes into *strongly-trap-connected components (STCCs)*, and a linear time algorithm for computing this unique decomposition is offered in Section 5.5. Both the graph structures and the STCCs algorithm generalize the classical decomposition of a directed graph into its strongly-connected components (SCCs) [110]. The algorithm builds on a generalization of the depth-first search (DFS), taking inspiration from Tarjan's SCCs classical algorithm, the structures of palm-trees and jungles described in Tarjan's original paper [110] need to be revisited and generalized (i.e., tr-palm-trees and tr-jungles) in order to handle the 2-player infinite pebble game's setting, this is done in Section 5.4. With this, in Section 5.7, the polynomial-time complexity for deciding Explicit McNaughton-Müller Games is also improved, from cubic to quadratic.

5.1.1 Background and Notation

An *arena* is a tuple $\mathcal{A} \triangleq (V, A, (V_\Box, V_\bigcirc))$ where $G^{\mathcal{A}} \triangleq (V, A)$ is a finite directed simple graph (i.e., there are no loops nor parallel arcs) and $(V_\Box, V_\bigcirc)$ is a partition of V into the set $V_\Box$ of vertices owned by Player $\Box$, and the set $V_\bigcirc$ of those owned by Player $\bigcirc$. Still $G^{\mathcal{A}}$ is not required to be a bipartite graph on colour classes $V_\Box$ and $V_\bigcirc$. The ingoing and outgoing neighbourhoods of $u \in V$ are $N^{\text{in}}_{\mathcal{A}}(u) \triangleq \{v \in V \mid (v,u) \in A\}$ and $N^{\text{out}}_{\mathcal{A}}(u) \triangleq \{v \in V \mid (u,v) \in A\}$, respectively. Disjoint-union is denoted by $\uplus$, e.g., $V = V_\Box \uplus V_\bigcirc$.

A *game* on $\mathcal{A}$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one. Initially the pebble is located on some $v_s \in V$, this is the *starting position*. At each round, if the pebble is currently on $v \in V_i$, for some $i \in \{\Box,\bigcirc\}$, Player i chooses an arc $(v,v') \in A$; and then the next round starts with the pebble on v'.

A finite (or infinite) *path* in $G^{\mathcal{A}}$ is a sequence $v_0v_1\ldots v_n\ldots \in V^*$ (or V^ω) such that $\forall_{j\geq 0}\ (v_j,v_{j+1}) \in A$; the *length* of $v_0v_1\ldots v_n$ is n. A *play* on $\mathcal{A}$ is any infinite path in $G^{\mathcal{A}}$. A *strategy* for Player i, where $i \in \{\Box,\bigcirc\}$, is a map $\sigma_i : V^* \times V_i \to V$ such that for every finite path $p'v$ in $G^{\mathcal{A}}$, where $p' \in V^*$ and $v \in V_i$, it holds that $(v,\sigma_i(p',v)) \in A$. The set of all strategies of Player i in $\mathcal{A}$ is denoted by $\Sigma^{\mathcal{A}}_i$. A play $v_0v_1\ldots v_n\ldots$ is *consistent* with some $\sigma \in \Sigma^{\mathcal{A}}_i$ if $v_{j+1} = \sigma(v_0v_1\ldots v_j)$ whenever $v_j \in V_i$. Given two strategies $\sigma_\Box \in \Sigma^{\mathcal{A}}_\Box$ and $\sigma_\bigcirc \in \Sigma^{\mathcal{A}}_\bigcirc$, and some $v_s \in V$, the *outcome* play $\rho_{\mathcal{A}}(v_s,\sigma_\Box,\sigma_\bigcirc)$ is the (unique) play that starts at v_s and is consistent with both $\sigma_\Box$ and $\sigma_\bigcirc$. For any $v \in V$, we denote by $\rho_{\mathcal{A}}(v_s,\sigma_\Box,\sigma_\bigcirc,v)$ the (unique) prefix of $\rho_{\mathcal{A}}(v_s,\sigma_\Box,\sigma_\bigcirc)$ which ends at the first occurence of v, if any; otherwise, $\rho_{\mathcal{A}}(v_s,\sigma_\Box,\sigma_\bigcirc,v) \triangleq \rho_{\mathcal{A}}(v_s,\sigma_\Box,\sigma_\bigcirc)$. For any finite (or infinite) path $p \in V^*$ (or $p \in V^\omega$), the *alphabet* of p is $\Xi(p) \triangleq \{v \in V \mid v \text{ appears in } p\}$.

Let $T \triangleq (V_T, A_T)$ be an inward directed tree, rooted at $r_T \in V_T$. We simply write $u \in T$ for $u \in V_T$. For each $u \in T$, there is only one *path* p_u going from u to r_T; the *depth* $d(u)$ of u is the length of p_u. An *ancestor* of $u \in T$ is any $v \in \Xi(p_u)$; it's a *proper ancestor* if $v \neq u$, and it's the *parent* $\pi_T(u)$ of u if $(u,v) \in A_T$. The *children* of $u \in T$ are all the $v \in T$ such that $\pi_T(v) = u$. A *descendant* of $u \in T$ is any $v \in T$ such that $u \in \Xi(p_v)$; it's a *proper descendant* if $v \neq u$. A *leaf* of T is any $u \in T$ having no children, i.e., $N^{\text{in}}_T(u) = \emptyset$. The *lowest common ancestor (LCA)* of a subset of vertices $S \subseteq T$ is

$$\gamma_S \triangleq \arg\max \Big\{ d(\gamma) \mid \gamma \in T \text{ and } \forall_{s\in S}\ s \text{ is a descendant of } \gamma \text{ in } T \Big\}.$$

The subtree of T that is rooted at $u \in T$ is denoted by T_u. Given a LIFO stack `St` containing some element $v \in$ `St`, we denote by `St`(v) the set of all elements $u \in$ `St` going from the top of `St` down 'til the first occurence of v, extremes included.

5.2 Trap-Reachability

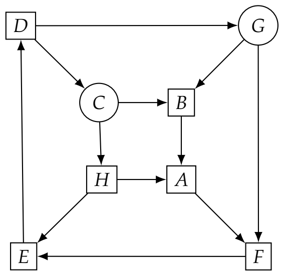
Recalling palm-trees and jungles. In a seminal work of Tarjan [110] some foundamental properties and applications of the *depth-first search (DFS)* were analyzed. Particularly, specific graph structures underlying the DFS were discussed in detail, namely *palm-trees* and *jungles*. This allowed the author to provide a linear time procedure, nowadays known as *Tarjan's SCCs algorithm*, for computing strongly-connected components (SCCs) in finite directed simple graphs.

Following [110], assume G is a finite directed simple graph that we wish to explore. Initially all the vertices of G are unexplored. We start from some vertex of G and choose an *outgoing* arc to follow. At each step, we select an unexplored arc (leading from a vertex already reached) and explore (traverse) that arc. When selecting an arc to explore, we always choose an arc emanating from the vertex *most recently* reached which still has unexplored arcs. Traversing the selected arc leads to some vertex, either new or already reached; if already reached, we backtrack and select another unexplored arc. Whenever we run out of arcs leading from old vertices, we choose some unreached vertex, if any exists, and begin a new exploration from this point. Eventually, the procedure will traverse all the arcs of G, each exactly once. This is a *depth-first search (DFS)* of G; one may call it *fwd*-DFS, because at each step the chosen arc is *outgoing*.

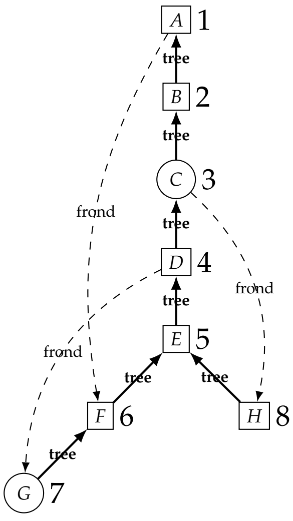
Recalling *palm-trees* from [110], consider in more detail what happens when a DFS is performed on G. The set of arcs leading to an unexplored vertex, when traversed during the search, forms an outward directed *tree* T. The other arcs fall into four categories: (i) some arcs are running from ancestors to descendants in T, these may well be ignored as they do not affect the SCCs of G; still, (ii) some other arcs run from descendants to ancestors in T, these are quite relevant instead, they are called *fronds*; (iii) other arcs run from one subtree to another within the same tree T, these are *internal cross-links*; (iv) suppose to continue the DFS until all arcs are explored, the process creates a family of trees which contains all vertices of G, i.e., a *spanning forest* F of G, plus sets of (fronds and) cross-links which may also connect two different trees in F; these are *external cross-links*. It is easy to see that if the vertices of G are numbered in the order in which they are reached during the search, e.g., by $\texttt{idx} : V \to \{1, \ldots, |V|\}$, then any (internal or external) cross-link (u,v) always has $\texttt{idx}[u] > \texttt{idx}[v]$. Any tree T of F, comprising fronds and internal cross-links, it is called *palm-tree*.

A directed graph consisting of a spanning forest, plus fronds and cross-links, it is named *jungle*, i.e., a family of palm-trees plus external cross-links, which is a natural representation of the graph-reachability structure of the input graph G.

Rev-DFS, rev-palm-trees and rev-jungles. In this work we need to impose an *opposite* direction w.r.t. that in which the arcs are traversed, so at each step of the DFS one actually chooses an *ingoing* arc to follow instead of an outgoing one. In this way, the corresponding search algorithm may be called *rev*-DFS. A moment's reflection reveals that this symmetric twist doesn't affect the basic properties of the DFS. For instance, if the vertices are numbered in the order in which they are reached during the rev-DFS, e.g., by $\texttt{idx} : V \to \{1, \ldots, |V|\}$,



(a) An Arena $\mathcal{A}$.



(b) The rev-palm-tree generated by rev-DFS, with indices of vertices.

1. (B,A)
2. (C,B)
3. (D,C)
4. (E,D)
5. (F,E)
6. (A,F)
7. (G,F)
8. (D,G)
9. (H,E)
10. (C,H)
11. (G,B)
12. (H,A)

(c) The order of arcs' exploration.

Figure 5.1: An arena (a), and a rev-palm-tree (b), generated by rev-DFS (c).

now a cross-link (u,v) always has $\texttt{idx}[u] < \texttt{idx}[v]$. So, a family of *rev-palm-trees* is constructed during rev-DFS. Let us call *rev-jungle* the graph structure underlying a rev-DFS, that is a family of rev-palm-trees comprising *fronds* and *cross-links*.

Trap-Reachability. A trivial *graph-reachability* property holds in any rev-palm-tree $T = (V_T, A_T)$: for any $u,v \in T$ such that v is an ancestor of u in T, there exists a simple path from u to v in T, i.e., v is graph-reachable from u in T. With this in mind, let's explore an *arena* $\mathcal{A} = (V, A, (V_\Box, V_\bigcirc))$ by a rev-DFS. Let $J^\mathcal{A}$ be the resulting rev-jungle, and let $T^\mathcal{A}$ be any rev-palm-tree of $J^\mathcal{A}$. An example is depicted in Fig. 5.1a and the corresponding rev-palm-tree $T^\mathcal{A}$ is in Fig. 5.1b; notice, $T^\mathcal{A}$ is still an arena. So, let's consider *reachability* on arenas, which is most relevant to 2-player infinite pebble games: given $\mathcal{A}$, and any two $u,v \in V$, we say that v is *reachable* from u in $\mathcal{A}$ if and only if there is some $\sigma_\Box \in \Sigma_\Box^\mathcal{A}$ (i.e., $\sigma_\Box = \sigma_\Box(u,v)$) such that for every $\sigma_\bigcirc \in \Sigma_\bigcirc^\mathcal{A}$, it holds $v \in \Xi\big(\rho_\mathcal{A}(u, \sigma_\Box, \sigma_\bigcirc)\big)$. Then, the rev-palm-tree $T^\mathcal{A}$, constructed as above, doesn't respect reachability: consider the two vertices $F, B \in V_\Box$ in the rev-palm-tree $T^\mathcal{A}$ shown in Fig. 5.1b; starting from F, Player $\Box$ admits no strategy which allows him to reach B, even though B is an ancestor of F in T; indeed, any play starting from F must first reach D, at that point, if Player $\Box$ plays (D,G) then Player $\bigcirc$ can go back to F by playing (G,F), otherwise, if Player $\Box$ plays (D,C), then Player $\bigcirc$ can play (C,H) thus reaching H, and notice that once on H the continuation of the play must reach D back again. So, starting from F, Player $\bigcirc$ can prevent Player $\Box$ to reach B. Thus we now aim at generalizing the classical DFS, palm-trees and jungles, from directed graphs to arenas, in such a way as to preserve reachability *within* the (suitably adapted) palm-trees. Particularly, a desirable "DFS on arenas" should maintain the following basic property: for any (suitably adapted) palm-tree T, if $u,v \in T$ and v is an ancestor of u in T, there exists $\sigma_\Box \in \Sigma^{\mathcal{A}}_\Box$ which allows Player $\Box$ to eventually reach v starting from u, *without leaving* T at the same time, no matter which $\sigma_\bigcirc \in \Sigma^{\mathcal{A}}_\bigcirc$ is chosen by Player $\bigcirc$.

This is the genesis of *trap-reachability*.

Definition 5.1. *Given an arena $\mathcal{A}$ on vertex set V, let $U \subseteq V$ and $u,v \in U$. We say that v is* U-trap-reachable *from u when* there exists $\sigma_\Box \in \Sigma^{\mathcal{A}}_\Box$ *(i.e., $\sigma_\Box = \sigma_\Box(u,v)$) such that* for every $\sigma_\bigcirc \in \Sigma^{\mathcal{A}}_\bigcirc$:

[reachability] $v \in \Xi\big(\rho_{\mathcal{A}}(u,\sigma_\Box,\sigma_\bigcirc)\big)$*; and,*

[entrapment] $\Xi\big(\rho_{\mathcal{A}}(u,\sigma_\Box,\sigma_\bigcirc,v)\big) \subseteq U$.

In this case, we denote $\sigma_\Box : u \overset{U}{\rightsquigarrow} v$, or $u \overset{U}{\rightsquigarrow} v$ when $\sigma_\Box$ is implicit; if $U = V$, $\sigma_\Box : u \rightsquigarrow v$ and $u \rightsquigarrow v$ will be enough notation.

Remark: Notice that any $u \in U$ is always U-trap-reachable from itself, for every $U \subseteq V$.

5.3 Strongly-Trap-Connectedness

In the rest of this work, $\mathcal{A} = (V, A, (V_\Box, V_\bigcirc))$ denotes the generic arena taken as input.

Definition 5.2. *We say that $U \subseteq V$ is* strongly-trap-connected *when for every $(u,v) \in U \times U$ there exists some $\sigma_\Box \in \Sigma^{\mathcal{A}}_\Box$ (i.e., $\sigma_\Box = \sigma_\Box(u,v)$) such that:*

$$\sigma_\Box : u \overset{U}{\rightsquigarrow} v.$$

Notice, $\emptyset$ and $\{v\}$ are strongly-trap-connected for any $v \in V$.

Definition 5.3. *A* strongly-trap-connected component (STCC) *is a maximal strongly-trap-connected $\mathcal{C} \subseteq V$ (i.e., such that if $\mathcal{C} \subseteq \mathcal{C}'$ and $\mathcal{C}'$ is strongly-trap-connected, then $\mathcal{C} = \mathcal{C}'$).*

Next, we observe the following property concerning strongly-trap-connectedness.

Lemma 5.1. *Let $V_1, V_2 \subseteq V$ be strongly-trap-connected.*
If $V_1 \cap V_2 \neq \emptyset$, then $V_1 \cup V_2$ is strongly-trap-connected.

Proof. Pick some $u,v \in V_1 \cup V_2$ and $z \in V_1 \cap V_2$, arbitrarily. Since $\{u,z\} \subseteq V_1$, and since V_1 is strongly-trap-connected, there exists some $\sigma_\Box(u,z) \in \Sigma^{\mathcal{A}}_\Box$ such that $\sigma_\Box(u,z) : u \overset{V_1}{\rightsquigarrow} z$; similarly, there is $\sigma_\Box(z,v) \in \Sigma^{\mathcal{A}}_\Box$ such that $\sigma_\Box(z,v) : z \overset{V_2}{\rightsquigarrow} v$. Then, consider the following $\sigma_\Box(u,v) \in \Sigma^{\mathcal{A}}_\Box$:

$$\sigma_\Box(u,v) \triangleq \begin{cases} \textit{(1)} \text{ Starting from } u, \text{ play } \sigma_\Box(u,z) \text{ until } z \text{ is first reached; then,} \\ \textit{(2)} \text{ once on } z, \text{ play } \sigma_\Box(z,v) \text{ until } v \text{ is finally reached.} \end{cases}$$

Clearly, $\sigma_{\Box}(u,v) : u \overset{V_1 \cup V_2}{\rightsquigarrow} v$. Since u and v were chosen arbitrarily, then $V_1 \cup V_2$ is strongly-trap-connected. $\Box$

Lemma 5.1 allows us to define and study an equivalence relation, i.e., $\sim_{\text{stc}} \subseteq V \times V$; it will turn out that the STCCs of $\mathcal{A}$ are the equivalence classes of $\sim_{\text{stc}}$.

Definition 5.4. *The binary relation $\sim_{stc}$ on V is defined as follows:*

$$\sim_{stc} \triangleq \big\{(u,v) \in V \times V \mid \exists_{U \subseteq V} \text{ such that } U \text{ is strongly-trap-connected and } \{u,v\} \subseteq U\big\}.$$

Lemma 5.2. *It holds that $\sim_{stc}$ is an equivalence relation on V.*

So, let $\{\mathcal{C}_i\}_{i=1}^{k}$ be the distinct equivalence classes of $\sim_{stc}$, for some $k \in \mathbb{N}$. Then, the following holds.

1. *If $U \subseteq V$ is strongly-trap-connected and $U \cap \mathcal{C}_i \neq \emptyset$, then $U \subseteq \mathcal{C}_i$;*
2. *$\mathcal{C}_i$ is strongly-trap-connected for each $i \in [k]$;*
3. *Let $U \subseteq V$ be strongly-trap-connected. Then, $\mathcal{C}_i \subsetneq U$ for* no *$i \in [k]$.*

Proof of ($\sim_{stc}$ is an equivalence relation on V). To begin, (i) $\sim_{\text{stc}}$ is *reflexive*: for any $u \in V$, let $U \triangleq \{u\}$; then, $u \overset{U}{\rightsquigarrow} u$, so U is strongly-trap-connected; this shows $u \sim_{\text{stc}} u$. (ii) $\sim_{\text{stc}}$ is *symmetric*, (actually, by definition): for any $u,v \in V$, assume $u \sim_{\text{stc}} v$; then, there exists some $U \subseteq V$ which is strongly-trap-connected and $u,v \in U$; so, the same set U certifies $v \sim_{\text{stc}} u$. (iii) $\sim_{\text{stc}}$ is *transitive*: indeed, for any $a,b,c \in V$, assume $a \sim_{\text{stc}} b$ and $b \sim_{\text{stc}} c$. Since $a \sim_{\text{stc}} b$, there exists V_1 which is strongly-trap-connected and such that $a,b \in V_1$; similarly, there exists V_2 which is strongly-trap-connected and such that $b,c \in V_2$. Consider $U \triangleq V_1 \cup V_2$. Since $b \in V_1 \cap V_2$, and V_1, V_2 are both strongly-trap-connected, then U is strongly-trap-connected by Lemma 5.1. Moreover, $a,c \in U$. So, $a \sim_{\text{stc}} c$. $\Box$

Proof of (1). Since $U \cap \mathcal{C}_i \neq \emptyset$, let $z \in U \cap \mathcal{C}_i$. Let $v \in U$, arbitrarily. Since U is strongly-trap-connected and $z,v \in U$, then $v \sim_{\text{stc}} z$. Therefore, $v \in \mathcal{C}_i$ (because $z \in \mathcal{C}_i$, which is an equivalence class of $\sim_{\text{stc}}$). $\Box$

Proof of (2). Let $u,v \in \mathcal{C}_i$, arbitrarily. Then, $u \sim_{\text{stc}} v$. So, there exists some $U \subseteq V$ which is strongly-trap-connected and such that $u,v \in U$. Thus, $u \overset{U}{\rightsquigarrow} v$. Notice, $u,v \in U \cap \mathcal{C}_i \neq \emptyset$. Then, by Item 1 of Lemma 5.2, $U \subseteq \mathcal{C}_i$. Since $u \overset{U}{\rightsquigarrow} v$ and $U \subseteq \mathcal{C}_i$, then $u \overset{\mathcal{C}_i}{\rightsquigarrow} v$. So, $\mathcal{C}_i$ is strongly-trap-connected. $\Box$

Proof of (3). Assume that $\mathcal{C}_i \subseteq U$, for some $i \in [k]$, and some $U \subseteq V$ which is strongly-trap-connected. Then, since $U \cap \mathcal{C}_i = \mathcal{C}_i \neq \emptyset$, by Item 1 of Lemma 5.2 we have $U \subseteq \mathcal{C}_i$. So, $\mathcal{C}_i = U$. $\Box$

Proposition 5.1. *Let $\mathcal{C} \subseteq V$, and consider the $\sim_{stc}$ relation on V. It holds that $\mathcal{C}$ is a STCC of $\mathcal{A}$ if and only if $\mathcal{C}$ is an equivalence class of $\sim_{stc}$.*

Proof. ($\Rightarrow$) If $\mathcal{C}$ is a STCC of $\mathcal{A}$, then $\mathcal{C}$ is strongly-trap-connected. So, $u \sim_{\text{stc}} v$ for every $u,v \in \mathcal{C}$. Then, $\mathcal{C} \subseteq \mathcal{C}'$ holds for some equivalence class $\mathcal{C}'$ of $\sim_{\text{stc}}$. By Item 2 of Lemma 5.2, $\mathcal{C}'$ is strongly-trap-connected. Thus, by maximality, $\mathcal{C}$ is not a proper subset of $\mathcal{C}'$. Therefore, $\mathcal{C} = \mathcal{C}'$.

($\Leftarrow$) If $\mathcal{C}$ is an equivalence class of $\sim_{\text{stc}}$, then: $\mathcal{C}$ is strongly-trap-connected by Item 2 of Lemma 5.2; and $\mathcal{C}$ is maximal by Item 3 of Lemma 5.2. Therefore, $\mathcal{C}$ is a STCC of $\mathcal{A}$. □

5.4 TR-Depth-First-Search

In this section, a DFS algorithm for the exploration on arenas is designed. Its rationale is that a new node $u \in V$ is attached to the r_T-rooted DFS-tree T under formation as soon as trap-reachability of r_T from u, within T itself, is already guaranteed (rather than requiring the weaker graph-reachability of r_T from u, like rev-DFS would do on graphs). The algorithm is called *Trap-Reachability-Depth-First-Search (tr-DFS)*. The pseudo-code of `tr-DFS()` is given in Algo. 12, and that of subprocedure `tr-DFS-visit()` is in Proc. 1.

Given $\mathcal{A}$, `tr-DFS`$(\mathcal{A})$ explores $\mathcal{A}$ so to construct another arena $J_{\mathcal{A}}$, like rev-DFS constructs a jungle. The `tr-DFS`$(\mathcal{A})$ (Algo. 12) is a generalization of rev-DFS, in the sense that, if $V_{\bigcirc} = \emptyset$, it works as rev-DFS and $J_{\mathcal{A}}$ coincides with a Tarjan's jungle. So, $J_{\mathcal{A}}$ comprises a forest of trees, called *tr-palm-trees*, with *fronds* and *cross-links*. Initially, four sets of arcs $A_{\text{tree}}, A_{\text{frond}}, A_{\text{petiole}}, A_{\text{cross}}$ are initialized to $\emptyset$, then some arcs will be added to them during `tr-DFS`$(\mathcal{A})$; when `tr-DFS`$(\mathcal{A})$ will halt, $A' \leftarrow A_{\text{tree}} \uplus A_{\text{frond}} \uplus A_{\text{petiole}} \uplus A_{\text{cross}}$ will be the arc set of $J_{\mathcal{A}}$. Let's say $u \in V$ *joins* $J_{\mathcal{A}}$ precisely when (u,v) is added to A_{tree}, for some $v \in V$.

An index `idx` $: V \to \{1,\ldots,|V|\}$ numbers the vertices in the order in which they join $J_{\mathcal{A}}$; initially, $\forall_{u\in V}$ `idx`$[u] \leftarrow +\infty$. Let's say $u \in V$ is *visited* if `idx`$[u] < +\infty$, and *unvisited* if `idx`$[u] = +\infty$. Any $u \in V_{\Box}$ joins $J_{\mathcal{A}}$ as soon as it is visited by the search (see lines 6-8 of `tr-DFS-visit()`, Proc. 1); but the *$V_{\bigcirc}$-rule* (i.e., that allowing $u \in V_{\bigcirc}$ to join $J_{\mathcal{A}}$) is more involved: any $u \in V_{\bigcirc}$ joins $J_{\mathcal{A}}$ as soon as all $u' \in N^{\text{out}}_{\mathcal{A}}(u)$ have already did it; and when $u \in V_{\bigcirc}$ joins some tr-palm-tree $\mathcal{P}$ of $J_{\mathcal{A}}$, with parent π (i.e., when $u \in V_{\bigcirc}$ *and* $(u,\pi) \in A_{\text{tree}}$ for some $\pi \in V$), then *all* arcs going out of u are tagged *petiole-arcs*; and π is the *LCA* of $N^{\text{out}}_{\mathcal{A}}(u)$ in $\mathcal{P}$. In fact, besides fronds and cross-links, tr-palm-trees have an additional class of arcs, the *petiole-arcs*: these are those arcs thanks to which $u \in V_{\bigcirc}$ can join $J_{\mathcal{A}}$. By considering LCAs the *$V_{\bigcirc}$-rule* allows us to preserve trap-reachability, as shown in Proposition 5.4. To implement the *$V_{\bigcirc}$-rule*, an additional counter `cnt` $: V_{\bigcirc} \to \mathbb{N}$ is employed, and the following invariant is maintained:

$$\forall_{u\in V_{\bigcirc}}\ \texttt{cnt}[u] = \big|\{v \in N^{\text{out}}_{\mathcal{A}}(u) \mid \texttt{idx}[v] = +\infty\}\big|, \qquad (I_{\texttt{cnt}})$$

Algorithm 12: tr-Depth-First Search

```
Procedure tr-DFS($\mathcal{A}$)
    input : An arena $\mathcal{A} = (V, A, (V_{\bigcirc}, V_{\Box}))$.
    output: A tr-jungle $\mathcal{J}_{\mathcal{A}}$.
    $A_{\text{tree}}, A_{\text{frond}}, A_{\text{petiole}}, A_{\text{cross}} \leftarrow \emptyset$;
    foreach $u \in V$ do
        idx[$u$] $\leftarrow +\infty$;
        active[$u$] $\leftarrow$ false;
        ready_St[$u$] $\leftarrow \emptyset$;
        if $u \in V_{\bigcirc}$ then
            cnt[$u$] $\leftarrow |N^{\text{out}}_{\mathcal{A}}(u)|$;
    next_idx $\leftarrow 1$;
    foreach $u \in V_{\Box}$ do
        if idx[$u$] $= +\infty$ then
            tr-DFS-visit($u, \mathcal{A}$);
    foreach $u \in V_{\bigcirc}$ do
        if idx[$u$] $= +\infty$ then
            idx[$u$] $\leftarrow$ next_idx;
            next_idx $\leftarrow$ next_idx $+ 1$;
    $A' \leftarrow A_{\text{tree}} \cup A_{\text{frond}} \cup A_{\text{petiole}} \cup A_{\text{cross}}$;
    return $\mathcal{J}_{\mathcal{A}} \leftarrow (V, A', (V_{\Box}, V_{\bigcirc}))$;
```

Algorithm 12: The Trap-Reachability-Depth-First Search.

also, for each $u \in V$, there's a LIFO stack of vertices named $\texttt{ready\_St}[u]$.

SubProcedure 1: tr-DFS-visit

```
Procedure $\mathit{tr\text{-}DFS\text{-}visit}(v,\mathcal{A})$
    input : One vertex $v \in V$ of $\mathcal{A}$.
    $\texttt{active}[v] \leftarrow \texttt{true}$;
    $\texttt{idx}[v] \leftarrow \texttt{next\_idx}$;
    $\texttt{next\_idx} \leftarrow \texttt{next\_idx} + 1$;
    // Check the in-neighbourhood of $v$
    foreach $u \in N^{in}_{\mathcal{A}}(v)$ do
        if $\mathit{idx}[u] = +\infty$ then
            if $u \in V_{\Box}$ then
                add $(u,v)$ to $A_{\text{tree}}$;
                $\texttt{tr-DFS-visit}(u,\mathcal{A})$;
            else
                $\texttt{cnt}[u] \leftarrow \texttt{cnt}[u] - 1$;
                if $\mathit{cnt}[u] = 0$ and $\exists$(LCA of $N^{out}_{\mathcal{A}}(u)$ in $(V, A_{tree})$) then
                    $\gamma \leftarrow$ the LCA of $N^{\text{out}}_{\mathcal{A}}(u)$ in $(V, A_{\text{tree}})$;
                    $\texttt{ready\_St}[\gamma].\texttt{push}(u)$;
        else if $\mathit{active}[u] = \mathit{true}$ then
            add $(u,v)$ to $A_{\text{frond}}$;
        else add $(u,v)$ to $A_{\text{cross}}$;
    // Check the ready-stack of $v$, i.e., $\texttt{ready\_St}[v]$
    while $\mathit{ready\_St}[v] \neq \emptyset$ do
        $u \leftarrow \texttt{ready\_St}[v].\texttt{pop}()$; // $u \in V_{\bigcirc}$
        add $(u,v)$ to $A_{\text{tree}}$;
        for each $t \in N^{\text{out}}_{\mathcal{A}}(u)$ do add $(u,t)$ to $A_{\texttt{petiole}}$;
        $\texttt{tr-DFS-visit}(u,\mathcal{A})$;
    $\texttt{active}[v] \leftarrow \texttt{false}$;
```

Procedure 1: The TR-DFS Visit Procedure.

Initially, $\forall_{u\in V}$ $\texttt{ready\_St}[u] \leftarrow \emptyset$ and $\forall_{u\in V_{\bigcirc}}$ $\texttt{cnt}[u] \leftarrow |N^{\text{out}}_{\mathcal{A}}(u)|$ (lines 5-7); then, $\texttt{cnt}[u]$ is decremented whenever some $v \in N^{\text{out}}_{\mathcal{A}}(u)$ is visited. When $\texttt{cnt}[u] = 0$ (at line 11 of $\texttt{tr-DFS-visit}(v,\mathcal{A})$, Proc. 1), all $v \in N^{\text{out}}_{\mathcal{A}}(u)$ have already joined $J_{\mathcal{A}}$: notice, if any two vertices in $N^{\text{out}}_{\mathcal{A}}(u)$ belong to two distinct tr-palm-trees in $J_{\mathcal{A}}$, there would be no way to preserve trap-reachability, in case u already joined $J_{\mathcal{A}}$, because Player $\bigcirc$ can always choose to move from u to any of the two shafts, and the LCA γ of $N^{\text{out}}_{\mathcal{A}}(u)$ in (V, A_{tree}) is undefined; still, if all vertices in $N^{\text{out}}_{\mathcal{A}}(u)$ belong to the same tr-palm-tree, the LCA γ of $N^{\text{out}}_{\mathcal{A}}(u)$ in (V, A_{tree}) does exist; so, firstly we seek for the LCA γ, and if it exists, push u on top of $\texttt{ready\_St}[\gamma]$ (lines 11-13 of $\texttt{tr-DFS-visit()}$, Proc. 1). So doing, $u \in V_{\bigcirc}$ joins $J_{\mathcal{A}}$ as soon as $\texttt{tr-DFS-visit()}$ backtracks, from the last $v \in N^{\text{out}}_{\mathcal{A}}(u)$ that has been visited, up to γ (possibly $\gamma = v$): at that point, $\texttt{ready\_St}[\gamma]$ will be checked and emptied (lines 17-21), and u will be found there inside, so

(u,γ) will be added to A_{tree} (line 19); also, for each $t \in N^{\text{out}}_{\mathcal{A}}(u)$ the arc (u,t) will be added to A_{petiole}, and $\texttt{tr-DFS-visit}(u,\mathcal{A})$ will be invoked.

To classify the remaining arcs into fronds or cross-links, an additional flag $\texttt{active} : V \to \{\texttt{true},\texttt{false}\}$ is employed; initially, $\forall_{u\in V}\ \texttt{active}[u] \leftarrow \texttt{false}$; then, $\texttt{active}[u]$ is set to $\texttt{true}$ when u is visited by the subprocedure $\texttt{tr-DFS-visit}(u,\mathcal{A})$ (line 1), finally, $\texttt{active}[u]$ is set back to $\texttt{false}$ when $\texttt{tr-DFS-visit}(u,\mathcal{A})$ ends. During $\texttt{tr-DFS-visit}(v,\mathcal{A})$, when some $u \in N^{\text{in}}_{\mathcal{A}}(v)$ such that $\texttt{idx}[u] \neq +\infty$ is explored, if $\texttt{active}[u] = \texttt{true}$ then (u,v) is added to A_{frond}, otherwise to A_{cross}.

There's still one point which is worth mentioning. During `tr-DFS()`, firstly, all $u \in V_{\Box}$ are considered, see lines 9-11 of `tr-DFS()` (Algo. 12); so, for each unvisited $u \in V_{\Box}$, $\texttt{tr-DFS-visit}(u,\mathcal{A})$ is invoked; after that, for each $u \in V_{\bigcirc}$ which is still unvisited, it is assigned $\texttt{idx}[u]$ incrementally, and $\texttt{tr-DFS-visit}(u,\mathcal{A})$ is not invoked anymore. Indeed, w.l.o.g we can assume that $\forall_{v\in V} |N^{\text{out}}_{\mathcal{A}}(v)| \geq 2$ holds, by pre-processing $\mathcal{A}$ as follows: for any $v \in V$, if $N^{\text{out}}_{\mathcal{A}}(v) = \emptyset$, remove v and all of its ingoing arcs; if $N^{\text{out}}_{\mathcal{A}}(v) = \{v'\}$, add (u,v') to A for each $u \in N^{\text{in}}_{\mathcal{A}}(v)$, finally remove v and all of its arcs. So doing, even if $\texttt{tr-DFS-visit}(v,\mathcal{A})$ would've been invoked for some $v \in V_{\bigcirc}$, say at line 14 of `tr-DFS()`, there would've been no $u \in V$ such that $(u,v) \in A_{\text{tree}}$: consider `tr-DFS-visit()` (Proc. 1), notice (u,v) could not have been added to A_{tree} neither at line 7 (because all $u \in V_{\Box}$ would've been already visited before at that time) nor at line 19 (since $\forall_{v\in V_{\bigcirc}} |N^{\text{out}}_{\mathcal{A}}(v)| \geq 2$, there would've been no way for "$\exists$ LCA of $N^{\text{out}}_{\mathcal{A}}(u)$ in (V, A_{tree})" at line 11). So, this way of going is fine. Let us now analyze the complexity of `tr-DFS()` (Algo. 12).

Proposition 5.2. *Given $\mathcal{A}$, the $\texttt{tr-DFS}(\mathcal{A})$ (Algo. 12) halts in time $\Theta\big(|V| + |A| + Time[\text{LCA}]\big)$, and it works with space $\Theta\big(|V| + |A| + Space[\text{LCA}]\big)$, where Time[LCA] (Space[LCA]) is the aggregate total time (space) taken by all of the LCA computations at lines 11-12 of `tr-DFS-visit()` (Proc. 1). Moreover, each $v \in V$ is numbered by `idx` exactly once.*

Proof. Firstly notice that the init-phase (lines 1-7 of Algo. 12) takes $\Theta(|V| + |A|)$ time. Secondly, Algo. 12 basically performs a sequence of invocations to $\texttt{tr-DFS-visit}(v,\mathcal{A})$ (Proc. 1), each one is for some $v \in V$. Any such $\texttt{tr-DFS-visit}(v,\mathcal{A})$ is invoked iff $\texttt{idx}[v] = +\infty$, and then $\texttt{idx}[v]$ is set to some non-zero value at line 2. Thus, the total number of invocations of `tr-DFS-visit()` (Proc. 1) is at most $|V|$; actually, by line 9 of `tr-DFS()` (Algo. 12), it is exactly $|V|$; so, each vertex $v \in V$ is numbered by $\texttt{idx} : V \to \{1, \ldots, |V|\}$ exactly once. Concerning time complexity, each invocation of the subprocedure $\texttt{tr-DFS-visit}(v,\mathcal{A})$ explores $N^{\text{in}}_{\mathcal{A}}(v)$ as follows: for each $u \in N^{\text{in}}_{\mathcal{A}}(v)$, the LCA of $N^{\text{out}}_{\mathcal{A}}(u)$ is computed at lines 11-12. Also, at the end of $\texttt{tr-DFS-visit}(v,\mathcal{A})$, the stack $\texttt{ready\_St}[v]$ is emptied; still, due to the condition $\texttt{cnt}[u] = 0$ that is checked at line 11 of `tr-DFS-visit()`, any $u \in V_{\bigcirc}$ can be pushed on $\texttt{ready\_St}[v]$ at most once and for at most one $v \in V$. Therefore, the $\Theta\big(|V| + |A| + \text{Time}[\text{LCA}]\big)$ time bound holds. Concerning space complexity, a similar argument shows that the aggregate total space for storing

$\{\texttt{ready\_St}[v]\}_{v\in V}$ is only $O(|V|)$. Also, the total size of idx, active and cnt is $\Theta(|V|)$, and that of A' is $\Theta(|A|)$ (see line 16 of tr-DFS(), Algo. 12). So, the working space is $\Theta\big(|V| + |A| + \text{Space}[\text{LCA}]\big)$. $\square$

Next, we analyze the structure of the arena $J_{\mathcal{A}}$ which is constructed by tr-DFS($\mathcal{A}$) (Algo. 12).

Let's start by formally defining *tr-palm-trees*. Some examples are shown in Fig. 5.2 and Fig. 5.3.

Definition 5.5. *A* tr-palm-tree *is a pair* $(\mathcal{P}, \texttt{idx})$, *where:*
(i) $\mathcal{P} \triangleq (V, A, (V_\square, V_\bigcirc))$ *is an arena, where:*

$$V \triangleq V_\square \uplus V_\bigcirc \text{ and } A \triangleq A_{tree} \uplus A_{frond} \uplus A_{petiole} \uplus A_{cross};$$

(ii) $\texttt{idx} : V \to \{1+j, \dots, |V|+j\}$, *for some fixed* $j \in \mathbb{N}$, *is a labelling of* V;
(iii) the following four main properties hold:

(tr-pt-1) $\mathcal{T}_\mathcal{P} \triangleq (V, A_{tree})$ *is an inward directed rooted tree such that:*

(a) the root $r_{\mathcal{T}_\mathcal{P}}$ *of* $\mathcal{T}_\mathcal{P}$ *is controlled by Player* $\square$, *i.e.,* $r_{\mathcal{T}_\mathcal{P}} \in V_\square$;

(b) $\texttt{idx}[u] > \texttt{idx}[v]$ *if* $(u,v) \in A_{tree}$;

(tr-pt-2) Each $(u,v) \in A_{frond}$ *connects some* $u \in V_\square$ *to one of its proper descendants* $v \in V$ *in* $\mathcal{T}_\mathcal{P}$;

(tr-pt-3) Each $(u,v) \in A_{petiole}$ *connects some* $u \in V_\bigcirc$ *to one of the descendants* v *of its parent* $\pi_{\mathcal{T}_\mathcal{P}}(u)$ *(i.e., possibly to* $\pi_{\mathcal{T}_\mathcal{P}}(u)$*); in particular, given any* $u \in V_\bigcirc$, *the following hold:*

(a) $\{v \in V \mid (u,v) \in A_{petiole}\} \uplus \{\pi_{\mathcal{T}_\mathcal{P}}(u)\} = N^{out}_\mathcal{P}(u)$;

(b) $\pi_{\mathcal{T}_\mathcal{P}}(u)$ *is the LCA of* $\{v \in V \mid (u,v) \in A_{petiole}\}$ *in* $\mathcal{T}_\mathcal{P}$;

(c) $\texttt{idx}[u] > \texttt{idx}[v]$ *for every* $v \in N^{out}_\mathcal{P}(u)$.

(tr-pt-4) Each arc $(u,v) \in A_{cross}$ *connects some* $u \in V_\square$ *to some* $v \in V$ *such that:*

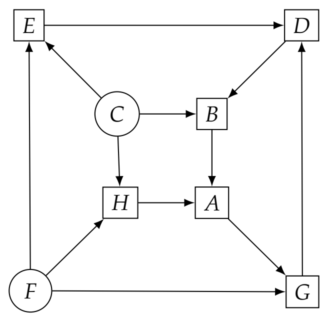
(a) v *is not a descendant of* u *in* $\mathcal{T}_\mathcal{P}$;

(b) either v *is a proper ancestor of* u *in* $\mathcal{T}_\mathcal{P}$ *or* $\texttt{idx}[u] < \texttt{idx}[v]$.

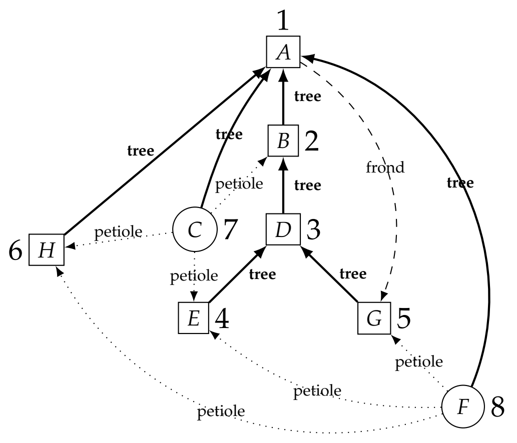
Definition 5.6. *A* tr-jungle *is any arena* $\mathcal{J} \triangleq (V, A, (V_\square, V_\bigcirc))$ *comprising a family of tr-palm-trees* $\{\mathcal{P}^i\}_{i=1}^k$, *for some* $k \in \mathbb{N}$, *and satisfying the following properties:*
(tr-jn-1) $\forall_{i\in[k]}\ \mathcal{P}^i \triangleq (V^i, A^i, (V^i_\square, V^i_\bigcirc))$, *where* $V^i_\square \subseteq V_\square, V^i_\bigcirc \subseteq V_\bigcirc, A^i \subseteq A$;
(tr-jn-2) $\forall_{i,j\in[k]}\ V^i \cap V^j = \emptyset$ *if* $i \neq j$;
(tr-jn-3) If $(u,v) \in A$ *for some* $u \in V^i$ *and* $v \in V^j$ *such that* $i \neq j$, *then:*

$$u \in V^i_\square \text{ and } i < j;$$

(tr-jn-4) If $v \in V \setminus \bigcup_{i=1}^k V^i$, *then* $v \in V_\bigcirc$ *and* $N^{out}_J(v) \subseteq V^i$ *for* no $i \in [k]$.



(a) An arena $\mathcal{A}$.



(b) The tr-palm-tree generated by tr-DFS rooted at A, with indices of vertices and labelled arcs.

1.(B,A) 7.(A,G)

2.(D,B) 8.(F,G)

3.(E,D) 9.(C,B)

4.(C,E) 10.(H,A)

5.(F,E) 11.(C,H)

6.(G,D) 12.(F,H)

(c) The order of arcs' exploration.

Figure 5.2: An arena (a), and a tr-palm-tree (b), generated by tr-DFS (c).

Definition 5.7. *Given a tr-palm-tree* $(\mathcal{P}, \texttt{idx})$*, for* $\mathcal{P} = (V, A, (V_\Box, V_\bigcirc))$*,* $A = A_{tree} \uplus A_{frond} \uplus A_{petiole} \uplus A_{cross}$*, the* support *of* $\mathcal{P}$ *is the arena* $\mathcal{P}_* \triangleq (V, A_*, (V_\Box, V_\bigcirc))$*, where:* $A_* \triangleq \big\{(u,v) \in A \mid u \in V_\Box\big\} \uplus A_{petiole}$*.*

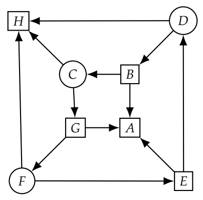
Notice that $A_* = A \setminus \big\{(u,v) \in A_{tree} \mid u \in V_\bigcirc\big\}$ *holds by (tr-pt-3).*

Given any tr-jungle $\mathcal{J}$ *with tr-palm-trees's family* $\{\mathcal{P}^i\}_{i=1}^k$*, for some* $k \in \mathbb{N}$*, let* $\overline{V} \triangleq V \setminus \bigcup_{i=1}^k V^i$ *(where* V^i *is the vertex set of* $\mathcal{P}^i$*). The* support *of* $\mathcal{J}$ *is the arena* $\mathcal{J}_*$ *which is obtained from* $\mathcal{J}$ *by replacing* $\mathcal{P}^i$ *with its support* $\mathcal{P}^i_*$*, for every* $i \in [k]$*, and by leaving intact all the vertices in* $\overline{V}$ *and all the arcs* (u,v) *of* $\mathcal{J}$ *such that: either, (i)* $u \in V^i$ *and* $v \in V^j$ *for some* $i \neq j$*; or, (ii)* $u \in \overline{V}$ *or* $v \in \overline{V}$ *(possibly both of them).*

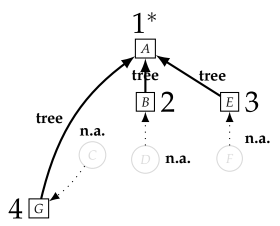
Proposition 5.3. *Let* $\mathcal{A} = (V, A, (V_\Box, V_\bigcirc))$ *be an arena. The following two propositions hold.*

1. *Let* J *be the arena constructed by* `tr-DFS`$(\mathcal{A})$ *(Algo. 12). Then,* J *is a tr-jungle.*

2. *Let* J *be a tr-jungle with support* J_**. Then,* `tr-DFS`(J_*) *(Algo. 12) constructs* J



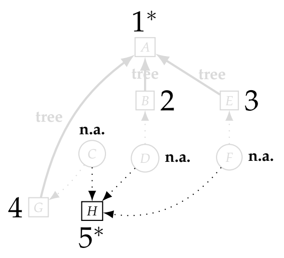
(a) An arena $\mathcal{A}$.



(b) The tr-palm-tree generated by a tr-DFS rooted at A, with indices of vertices and labelled arcs.

1. (B,A)
2. (D,B)
3. (E,A)
4. (F,E)
5. (G,A)
6. (C,G)

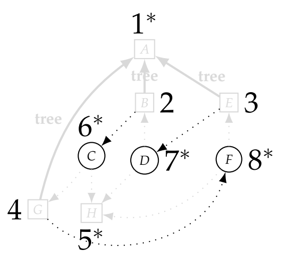
(c) The order of arcs' exploration.



(d) The tr-palm-tree generated by a tr-DFS rooted at H.

7. (C,H)
8. (D,H)
9. (F,H)

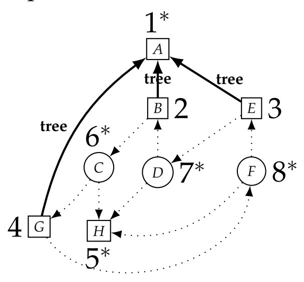
(e) The order of arcs' exploration.



(f) The tr-palm-trees generated by an tr-DFS rooted at C, D, F.

10. (B,C)
11. (E,D)
12. (G,F)

(g) The order of arcs' exploration.



(h) The resulting tr-jungle, which is generated by multiple tr-DFSs rooted at A, H, C, D and F.

Figure 5.3: An arena (a), and the construction of a tr-jungle (b-h).

itself, i.e., $J_{J_*} = J$.

Proof of (1). Recall, `tr-DFS`$(\mathcal{A})$ performs a sequence of invocations to the subprocedure `tr-DFS-visit`$(\cdot,\mathcal{A})$; by Proposition 5.2, each $v \in V$ is numbered by `idx` exactly once. Let k be the number of times that the subprocedure `tr-DFS-visit`$(u_i,\mathcal{A})$ is invoked at line 11 of `tr-DFS`$()$ (Algo. 12). For each $i = 1,2,\ldots,k$, it holds $u_i \in V_\Box$ by line 9, then, let $V^i \subseteq V$ be the set of vertices that are numbered by idx during tr-DFS-visit$(u_i,\mathcal{A})$, recursive calls included. Let A^i be the set of arcs that are added to any of A_{tree}, A_{frond}, A_{petiole}, A_{cross} during that same tr-DFS-visit$(u_i,\mathcal{A})$, and let $A^i_t \triangleq \{(a,b) \in A^i \mid a,b \in V^i\}$ and $A^i_c \triangleq A^i \setminus A^i_t$. Let $\mathcal{P}^i \triangleq (V^i, A^i_t, (V_\square \cap V^i, V_\bigcirc \cap V^i))$, and let idx^i be the restriction of idx to V^i. It is not difficult to see that $(\mathcal{P}^i,$ idx$^i)$ is a tr-palm-tree: indeed, for each $i \in [k]$, $(\mathcal{P}^i,$ idx$^i)$ is constructed by tr-DFS-visit$(u_i,\mathcal{A})$ and thus it satisfies *(tr-pt-1)* to *(tr-pt-4)*, where any $u \in V_\bigcirc$ joins $\mathcal{P}^i$ according to: (i) the checking of the cnt$[u] = 0$ condition at line 11, (ii) the LCA computation at lines 11-12, (iii) the emptying of ready_St$[v]$ at lines 17-21; also recall that, for every $u \in V_\bigcirc$, cnt$[u]$ was initialized to $|N^{\text{out}}_{\mathcal{A}}(u)|$ at line 7 of tr-DFS() (Algo. 12), and then cnt$[u]$ is decremented at line 10 of tr-DFS-visit$(v,\mathcal{A})$ each time some $v \in N^{\text{out}}_{\mathcal{A}}(u)$ is visited; with this in mind, it is easy to check that *(tr-pt-1)* to *(tr-pt-4)* are satisfied. Next, we claim that J is a tr-jungle with tr-palm-tree family $\{\mathcal{P}^i\}_{i\in[k]}$. Indeed, *(tr-jn-1)* and *(tr-jn-2)* clearly hold. Concerning *(tr-jn-3)*, let $(u,v) \in A$ for some $u \in V^i$ and $v \in V^j$ such that $i \neq j$; then $u \in V^i_\square$, because $\mathcal{P}^i$ is a tr-palm-tree so that *(tr-pt-3)* holds for $V_\bigcirc$; also, $i < j$ since otherwise u would've joined $\mathcal{P}^i$ at lines 6-8 of tr-DFS-visit(). Concerning *(tr-jn-4)*, let $v \in V \setminus \bigcup_{i=1}^k V^i$, then $v \in V_\bigcirc$ by lines 9-15 of tr-DFS(); also, $N^{\text{out}}_J(v) \subseteq V^i$ holds for *no* $i \in [k]$, otherwise v would've joined $\mathcal{P}^i$ thanks to lines 9-13 and 17-21 of tr-DFS-visit(). So J is a tr-jungle. $\square$

Proof of (2). Notice that J_* is obtained from J simply by removing from the tr-palm-trees of J all the arcs $(u,v) \in A_{\text{tree}}$ such that $u \in V_\bigcirc$. Consider the ordering $<_{\texttt{idx}}$ on V induced by the labelling idx of J, i.e., $\forall_{a,b\in V}\, a <_{\texttt{idx}} b \iff \texttt{idx}[a] < \texttt{idx}[b]$. Construct an adjacency list of J_* such that: (i) the main list of vertices is ordered according to $<_{\texttt{idx}}$; (ii) for each $u \in V$, also $N^{\text{in}}_J(u)$ is ordered according to $<_{\texttt{idx}}$. So doing, since J satisfies *(tr-jn-1)* to *(tr-jn-4)* and their tr-palm-trees satisfy *(tr-pt-1)* to *(tr-pt-4)*, then tr-DFS(J_*) (Algo. 12) reconstructs J itself, i.e., that $J_{J_*} = J$. $\square$

The next proposition shows that tr-jungles do respect trap-reachability.

Proposition 5.4. *Let $\mathcal{J}$ be a tr-jungle with family of tr-palm-trees $\{\mathcal{P}^i\}_{i=1}^k$, for some $k \in \mathbb{N}$, assume that $\mathcal{P}^i = (V^i, A^i, (V^i_\square, V^i_\bigcirc))$ holds for each $i \in [k]$. There exists $\sigma_\square \in \Sigma^{\mathcal{J}}_\square$ (i.e., $\sigma_\square = \sigma_\square(i)$) such that, for any two vertices $u,v \in V^i$, if u is a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$, then: $\sigma_\square : u \overset{V^i}{\leadsto} v$.*

Proof. By lines 9-11 of tr-DFS(), for every $u \in V_\square$ there exists some $i_u \in [k]$ such that $u \in V^{i_u}$.

Then, consider the strategy $\sigma_\square \in \Sigma^{\mathcal{J}}_\square$ (i.e., $\sigma_\square = \sigma_\square(i)$) defined as follows:

$$\forall_{u\in V_\square}\, \sigma_\square(u) \triangleq \begin{cases} \pi_{\mathcal{T}_{\mathcal{P}^{i_u}}}(u), & \text{if } u \text{ is } \textit{not} \text{ the root of } \mathcal{T}_{\mathcal{P}^{i_u}}; \\ \text{any } u' \in N^{\text{out}}_{\mathcal{J}}(u), & \text{if } u \text{ is the root of } \mathcal{T}_{\mathcal{P}^{i_u}}. \end{cases}$$

Let $i \in [k]$ and $u, v \in V^i$ be fixed, arbitrarily. Recall that, by *(tr-pt-1)*, the vertices of $\mathcal{T}_{\mathcal{P}^i}$ are numbered by `idx` so that $\texttt{idx}[v] < \texttt{idx}[u]$ if v is a proper ancestor of u in $\mathcal{T}_{\mathcal{P}^i}$. To prove the thesis, we argue by induction on the value of $\texttt{idx}[u]$. Let $z \triangleq \min_{x \in V^i} \texttt{idx}[x]$. Assume $\texttt{idx}[u] = z$. Then, $u = r_{\mathcal{T}_{\mathcal{P}^i}}$ is the root of $\mathcal{T}_{\mathcal{P}^i}$. Since $u, v \in V^i$ and u is a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$, then $v = u$; so, the thesis trivially holds. Instead, assume $\texttt{idx}[u] > z$. Again, if $u = v$ the claim holds trivially. So, let $u \neq v$. Assume the thesis holds for every $x \in V^i$ which is still a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$ and such that $\texttt{idx}[v] \leq \texttt{idx}[x] < \texttt{idx}[u]$.

We have two cases to analyze, either $u \in V_\Box$ or $u \in V_\bigcirc$.

(i) If $u \in V_\Box$, then $\sigma_\Box(u) = \pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$. By *(tr-pt-1)*, $\texttt{idx}[\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)] < \texttt{idx}[u]$. Since $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is the parent of u in $\mathcal{T}_{\mathcal{P}^i}$ and $u \neq v$, then $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is still a descendant of v; thus, by induction hypothesis: $\sigma_\Box : \pi_{\mathcal{T}_{\mathcal{P}^i}}(u) \overset{V^i}{\leadsto} v$. Therefore, since $\sigma_\Box : u \overset{V^i}{\leadsto} \pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ and $\sigma_\Box : \pi_{\mathcal{T}_{\mathcal{P}^i}}(u) \overset{V^i}{\leadsto} v$, the thesis holds.

(ii) If $u \in V_\bigcirc$, firstly recall that by *(tr-pt-3)*, $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is the LCA of $\Lambda^i(u) \triangleq \{u' \in V \mid (u, u') \in A^i_{\text{petiole}}\}$; notice that $\Lambda^i(u) = N^{\text{out}}_{\mathcal{J}_*}(u)$, where $\mathcal{J}_*$ is the support of $\mathcal{J}$. Fix $u' \in N^{\text{out}}_{\mathcal{J}_*}(u)$, arbitrarily. It holds that u' is still a descendant of $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ in $\mathcal{T}_{\mathcal{P}^i}$, because $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is the LCA of $N^{\text{out}}_{\mathcal{J}_*}(u)$. Thus, since $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$, then u' is also a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$. And, by item *(c)* of *(tr-pt-3)*, $\texttt{idx}[u'] < \texttt{idx}[u]$. Thus, by induction hypothesis, $\sigma_\Box : u' \overset{V^i}{\leadsto} v$. Since u' was chosen arbitrarily, then $\sigma_\Box : u \overset{V^i}{\leadsto} v$. This concludes the inductive step of the proof. So, in any case, $\sigma_\Box : u \overset{V^i}{\leadsto} v$. □

Still it remains to be seen how to perform the LCAs computations that are needed at lines 11-12 of `tr-DFS()` (Proc. 1). In the next paragraph, we suggest to adopt a *disjoint-set forest* data structure with *non-ranked* `Union()` and *path-compression* `Find()`.

LCAs by Disjoint-Set Forest. A *disjoint-set forest (DSF)* data structure, hereby denoted by $\mathcal{D}$, also called *union-find* data structure or *merge-find* set, is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets, each of which is represented by a tree.

It supports the following operations:
$\mathcal{D}.\texttt{MakeSet}(\cdot)$, $\mathcal{D}.\texttt{Union}(\cdot,\cdot)$ and $\mathcal{D}.\texttt{Find}(\cdot)$, such that:
(dsf-1) The representative element of each set is the root of that set's tree;
(dsf-2) $\texttt{MakeSet}(v)$ initializes the parent of a vertex $v \in V$ to be v itself;
(dsf-3) $\texttt{Union}(u,v)$ combines two trees, T_1 rooted at u and T_2 rooted at v, into a new tree T_3 still rooted at v, i.e., by adding u as a child of v (*non-ranked union*).
(dsf-4) $\texttt{Find}(v)$, starting from v, it traverses the ancestors of v until the root r of the tree containing v is finally reached. While doing this, it changes each ancestor's parent reference to point to r (*path-compression*); the resulting

tree is much flatter, speeding up future operations, not only on these traversed elements but also on those referencing them.

We can now describe how to perform the LCAs computations at lines 11-12 of $\texttt{tr-DFS-visit}()$ (Proc. 1). We refer to the following procedure as to the *"DSF-based* $\texttt{tr-DFS()}$*"*. We have a global DSF data structure named $\mathcal{D}$. The init-phase is almost the same as Algo. 12, the only additions being that, for each $v \in V$:

(dsf-init-1) $\mathcal{D}.\texttt{MakeSet}(v)$ is executed;

(dsf-init-2) If $v \in V_{\bigcirc}$, an array $\texttt{low\_ready} : V \to \mathbb{N} \cup \{+\infty\}$ is initialized as $\texttt{low\_ready}[v] \leftarrow +\infty$. Given $\mathcal{A}$ in input, the DSF-based $\texttt{tr-DFS}(\mathcal{A})$ is going to keep the following invariant property:

$$\forall_{v \in V_{\bigcirc}}\ \texttt{low\_ready}[v] = \min\big\{\texttt{idx}[u] \mid u \in N^{\text{out}}_{\mathcal{A}}(v)\big\}. \qquad (\mathrm{I}_{\text{low}})$$

Next, the visit-phase begins as at lines 9-15 of $\texttt{tr-DFS}()$ (Algo. 12): for each $v \in V_{\Box}$, if v is still unvisited (i.e., if $\texttt{idx}[v] = +\infty$) then $\texttt{tr-DFS-visit}(v, \mathcal{A})$ is invoked. As soon as all $V_{\Box}$ vertices have been visited, then, all $v \in V_{\bigcirc}$ that are still unvisited are handled as at lines 12-15 of $\texttt{tr-DFS}()$ (Algo. 12).

Also, the halting-phase is the same as before, see lines 16-17 of $\texttt{tr-DFS}()$ (Algo. 12).

Let us now describe the distinctive features of the DSF-based $\texttt{tr-DFS()}$. Let $v \in V$, then:

(dsf-visit-1) Whenever the DSF-based $\texttt{tr-DFS-visit}(v, \mathcal{A})$ makes a recursive call on some input vertex $u \in N^{\text{in}}_{\mathcal{A}}(v) \cup \texttt{ready\_St}[v]$ (see lines 8 and 21 of Proc. 1), soon after that, it is executed $\mathcal{D}.\texttt{Union}(u, v)$.

(dsf-visit-2) Suppose that the DSF-based $\texttt{tr-DFS-visit}(v, \mathcal{A})$ is currently exploring some $v \in V$, and that it comes to consider some $u \in N^{\text{in}}_{\mathcal{A}}(v) \cap V_{\bigcirc}$ (at line 4 and 9). Then, $\texttt{low\_ready}$ is updated as follows:

$$\texttt{low\_ready}[u] \leftarrow \min(\texttt{low\_ready}[u], \texttt{idx}[v]);$$

this aims at satisfying the I_{low} invariant. Next, $\texttt{cnt}[u]$ is decremented (as at line 10 of Proc. 1).

If the condition $\texttt{cnt}[u] = 0$ holds (line 11 of Proc. 1), the following is done:

(a) It is assigned:

$$\texttt{low\_}v \leftarrow \text{“the unique } x \in N^{\text{out}}_{\mathcal{A}}(u) \text{ such that } \texttt{idx}[x] = \texttt{low\_ready}[u]\text{”};$$

(b) Then, it is computed $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low\_}v)$;

(c) Then, if $\texttt{active}[\gamma] = \texttt{true}$, it is executed $\texttt{ready\_St}[\gamma].\texttt{push}(u)$; indeed, in that case, we can prove (see Proposition 5.5) that the LCA of $N^{\text{out}}_{\mathcal{A}}(u)$ in (V, A_{tree}) does exist, and it is really γ.

The rest of the DSF-based $\texttt{tr-DFS-visit}()$ is the same as Proc. 1.

This concludes the description of the DSF-based $\texttt{tr-DFS()}$.

At this point we shall prove that the above mentioned property concerning γ and LCAs holds.

Proposition 5.5. *Suppose that* `tr-DFS-visit`$(v,\mathcal{A})$ *(DSF-based) is invoked, for some* $v \in V$*, and that it comes to consider some* $u \in N^{in}_{\mathcal{A}}(v) \cap V_{\bigcirc}$ *(at line 4 and 9). Assume that* `idx`$[u] = +\infty$ *at line 5 and that* `cnt`$[u] = 0$ *at line 11. Let* γ *be the vertex returned by* $\mathcal{D}$.`find`(`low_`v)*, where* `low_`v *is the unique* $x \in V$ *such that* `idx`$[x] =$ `low_ready`$[u]$*. If* `active`$[\gamma] =$ `true` *holds, then the LCA of* $N^{out}_{\mathcal{A}}(u)$ *in* (V, A_{tree}) *is* γ*.*

Proof. Firstly, during the execution of the DSF-based `tr-DFS`$(\mathcal{A})$, the (V, A_{tree}) still grows as a forest. Indeed, if a new arc (u,v) is added to A_{tree} it holds that `idx`$[u] = +\infty$ (by line 5 of `tr-DFS-visit()`) and that `idx`$[v] < +\infty$ (by line 2 of `tr-DFS-visit`$(v,\mathcal{A})$); so, no cycle can be formed. Thus, when `tr-DFS-visit`$(v,\mathcal{A})$ is invoked for $v \in V$, we can consider the unique maximal tree $\mathcal{T}^v$ in (V, A_{tree}) which contains v (constructed up to that point). Let p_v be the path in $\mathcal{T}^v$ going from v to the root $r_{\mathcal{T}^v}$. By *(dsf-visit-1)*, by definition of `low_`v, and since $\gamma = \mathcal{D}$.`find`(`low_`v) and `active`$[\gamma] =$ `true`, then γ lies on p_v. Thus, γ is the LCA of `low_`v and v in $\mathcal{T}^v$ (possibly $\gamma =$ `low_`v). Next, we argue that $N^{\text{out}}_{\mathcal{A}}(u) \subseteq \mathcal{T}^v_\gamma$, where $\mathcal{T}^v_\gamma$ is the subtree of $\mathcal{T}^v$ comprising all and only the descendants of γ (i.e., the subtree of $\mathcal{T}^v$ rooted at γ). Indeed, by *(dsf-visit-2)*, it holds that:

$$\texttt{idx}[\texttt{low\_}v] = \min\{\texttt{idx}[x] \mid x \in N^{\text{out}}_{\mathcal{A}}(u)\}.$$

So, when `cnt`$[u] = 0$ holds at line 11 of `tr-DFS-visit`$(v,\mathcal{A})$, and since γ is an ancestor of `low_`v, then:

$$\forall_{x \in N^{\text{out}}_{\mathcal{A}}(u)} \texttt{idx}[\gamma] \le \texttt{idx}[\texttt{low\_}v] \le \texttt{idx}[x] < +\infty.$$

Notice that all vertices in $\mathcal{T}^v$ which are not descendants of γ still have a smaller `idx` than γ (i.e., they were all visited before γ), and all those which are proper descendants of γ have a greater `idx` than γ. All these combined, it follows $N^{\text{out}}_{\mathcal{A}}(u) \subseteq \mathcal{T}^v_\gamma$. So, γ is a common ancestor of $N^{\text{out}}_{\mathcal{A}}(u)$ in $\mathcal{T}^v$; but γ is also the LCA of $\{\texttt{low\_}v, v\} \subseteq N^{\text{out}}_{\mathcal{A}}(u)$, therefore, γ is really the LCA of $N^{\text{out}}_{\mathcal{A}}(u)$ in $\mathcal{T}^v$. □

By Proposition 5.5, then, Proposition 5.3 holds even for the DSF-based `tr-DFS()`. Concerning complexity, by relying on the result offered in [80], we now show that the DSF-based `tr-DFS()` halts in linear time.

Definition 5.8 ([80]). *Let* $T = (V, A)$ *be any rooted tree. Let* $u_1, \ldots, u_k$ *be a path in* T *listed from a leaf* u_1 *in the direction towards the root of* T *(i.e.,* u_k *is some ancestor of* u_1*). The* path compression $C = (u_1, \ldots, u_k)$ *is an operation that modifies* T *as follows:*

(i) It deletes from T *all the arcs* (u_i, u_{i+1})*, for* $i = 1, \ldots, k-1$*;*

(ii) It makes each of the vertices u_i*, for* $i = 1, \ldots, k-1$*, a new son of* u_k*;*

(iii) It deletes all new sons of u_k *of degree* 1 *which may occur (particularly,* u_1 *is deleted).*

The vertex u_k is called the root *of C. We also say that C* starts *from u_1. The* length *of C is $|C| \triangleq k-1$.*

Any sequence $S = (C_1, \ldots, C_n)$ of path compressions on a tree T is called a strong postorder path compression system (SPPCS) *if and only if the following four properties hold:*

(i) *Each C_i is a path compression on the tree T^i obtained from T after that the path compressions $C_1, \ldots, C_{i-1}$ have been executed (where $T^1 = T$);*

(ii) *Each leaf of T is a starting point of exactly one path compression of S;*

(iii) $(1,2,\ldots,n)$ *is a linear ordering of all the n leaves of T induced by a fixed postorder of T;*

(iv) *Let the root of a compression C_i, for any $2 \le i \le n$, be some vertex u of T. Then, all the compressions C_j such that $j < i$ and $j \in T_u$ have roots in a descendant of u in T.*

The length *of S is defined as $|S| \triangleq \sum_{i=1}^{n} |C_i|$.*

Theorem 5.1 ([80]). *Let S be a SPPCS on a tree T with n leaves. Then, $|S| \le 5 \cdot n$.*

Proposition 5.6. *Given $\mathcal{A}$, assume that the DSF-based* `tr-DFS`$(\mathcal{A})$ *is invoked. Then, it halts in $\Theta\big(|V| + |A|\big)$ time.*

Proof. Recall that during the `tr-DFS()`, the graph (V, A_{tree}) grows as a forest. By *(dsf-visit-1)*, that forest coincides with the DSF that is constructed by means of the $\mathcal{D}$.`Union`$(\cdot,\cdot)$ operations.

So, in order to rely on Theorem 5.1, let us consider the following directed rooted tree $T^* \triangleq (V_{T^*}, A_{T^*})$:

$$\begin{aligned}
V_{T^*} \triangleq & V \uplus \{r_{T^*}\} \uplus \{l_{(u,v)} \mid (u,v) \in A, u \in V_{\bigcirc}\};\\
A_{T^*} \triangleq & A_{\texttt{tree}} \uplus \{(r_{\mathcal{T}}, r_{T^*}) \mid \mathcal{T} \text{ is a tree in } (V, A_{\texttt{tree}}) \text{ and } r_{\mathcal{T}} \text{ is its root}\}\\
& \uplus \{(l_{(u,v)}, v) \mid l_{(u,v)} \in V_{T^*}\}.
\end{aligned}$$

where $r_{T^*}, l_{(u,v)} \notin V$ for every $l_{(u,v)} \in V_{T^*}$. Notice that r_{T^*} is the root of T^* and $\{l_{(u,v)} \in V_{T^*}\}$ is a subset of the leaves of T^*; so, for each $u \in V_{\bigcirc}$ and $v \in N^{\text{out}}_{\mathcal{A}}(u)$, there is a new leaf $l_{(u,v)}$ attached to v in T^*. Also notice that $|V_{T^*}| = 1 + |V| + |\{(u,v) \in A, u \in V_{\bigcirc}\}| \le 1 + |V| + |A|$ and $|A_{T^*}| = |V_{T^*}| - 1$. Now, observe that each $\mathcal{D}$.`Find`() operation, that is possibly made by `tr-DFS`$(\mathcal{A})$, it is made only by `tr-DFS-visit`$(v, \mathcal{A})$ (for some $v \in V$) as prescribed by *items (a) and (b)* of *(dsf-visit-2)* and only if `cnt`$[u] = 0$ holds at line 11, i.e., $\gamma \leftarrow \mathcal{D}$.`Find`(`low_`$v$), where `low_`$v$ is the unique $x \in N^{\text{out}}_{\mathcal{A}}(u)$ such that `idx`$[x] =$ `low_ready`$[u]$, and $u \in V_{\bigcirc} \cap N^{\text{in}}_{\mathcal{A}}(v)$ at lines 9-10. Each of these $\mathcal{D}$.`Find()` operations acts in a natural manner on T^*: indeed, $\mathcal{D}$.`Find`(`low_`v) induces a path compression $C_{\texttt{low\_}v}$ on T^*, if we assume that $C_{\texttt{low\_}v}$ starts at the leaf $l_{(u,\texttt{low\_}v)}$ and that it terminates at γ (i.e., γ is the root of $C_{\texttt{low\_}v}$). Since $\gamma \leftarrow \mathcal{D}$.`Find`(`low_`$v$) is executed only if `cnt`$[u] = 0$ holds at line 11 of the subprocedure `tr-DFS-visit`$(v, \mathcal{A})$, then each path compression on T^* starts from a distinct leaf $l_{(u,\texttt{low\_}v)}$. It is safe to assume that each leaf of T^* is a starting point of exactly one path compression, because for each leaf l' of T^* that has not been the starting point of any path compression, we can impose a *void* path compression, i.e., one that starts and terminates at l'. Then, we argue that the family of all path compressions on T^* that are induced by the whole execution of `tr-DFS`$(\mathcal{A})$ is a SPPCS: indeed, during the search, T^* is (implicitly) visited in a post-ordering; when some $v \in V$ is visited, and some $u \in N^{\text{in}}_{\mathcal{A}}(v) \cap V_{\bigcirc}$ is considered at line 4 of `tr-DFS-visit`$(v,\mathcal{A})$, then the root γ of the path compression $C_{\texttt{low\_}v}$ is the LCA of $\{v, \texttt{low\_}v\}$ in T^* (as shown in Proposition 5.5). Thus, we argue that *(sppcs-4)* holds. Assume some path compression $C_{\texttt{low\_}v'}$ was done before $C_{\texttt{low\_}v}$ and that $\texttt{low\_}v' \in T^*_{\gamma}$. So, $\texttt{idx}[\gamma] < \texttt{idx}[\texttt{low\_}v']$. Also, by *(dsf-visit-2)*, $C_{\texttt{low\_}v'}$ was induced during `tr-DFS-visit`$(v',\mathcal{A})$, for some $v' \in V$. Thus, since $C_{\texttt{low\_}v'}$ was done before $C_{\texttt{low\_}v}$, then $\texttt{idx}[v'] < \texttt{idx}[v]$; and by definition of $\texttt{low\_}v'$, then $\texttt{idx}[\texttt{low\_}v'] < \texttt{idx}[v']$. Therefore, $\texttt{idx}[\gamma] < \texttt{idx}[v'] < \texttt{idx}[v] < +\infty$ holds when $C_{\texttt{low\_}v'}$ is performed. This means that $v' \in T^*_{\gamma}$. Since the root γ' of $C_{\texttt{low\_}v'}$ is the LCA of $\{v', \texttt{low\_}v'\}$ in T^* (as shown in Proposition 5.5), and since $\{v', \texttt{low\_}v\} \subseteq T^*_{\gamma}$, then $\gamma' \in T^*_{\gamma}$; so, *(sppcs-4)* holds. At this point, by Theorem 5.1, the total length of all path compressions that are induced by `tr-DFS()` on T^* is $O(|V_{T^*}|) = O(|V| + |A|)$. It is clear that the space required for storing $\mathcal{D}$ is $\Theta(|V| + |A|)$. So, also by Proposition 5.2, the complexity of the DSF-based `tr-DFS()` is $\Theta(|V| + |A|)$. $\square$

5.5 Linear Time Algorithm for STCCs

Lemma 5.3. *Let $C_0, \ldots, C_{k-1} \subseteq V$ be some STCCs of $\mathcal{A}$, for some $k \geq 2$. For each $i \in \{0, \ldots, k-1\}$ fix some $u_i \in C_i$, arbitrarily; and let $i' \triangleq (i+1) \mod k$. Assume that the following* tr-cycle *(i.e., cyclic trap-reachability relation) holds for some $\{\sigma_{\Box}(u_i, u_{i'})\}_{i=0}^{k-1} \subseteq \Sigma^{\mathcal{A}}_{\Box}$:*

$$\sigma_{\Box}(u_i, u_{i'}) : u_i \overset{C_i \cup \{u_{i'}\}}{\rightsquigarrow} u_{i'}, \text{ for each } i = 0, \ldots, k-1.$$

Then, $C_i = C_{i'}$ for every $i \in \{0, \ldots, k-1\}$.

Proof. Let $C^* \triangleq \cup_{i=0}^{k-1} C_i$. We argue that C^* is strongly-trap-connected. Let $x, y \in C^*$ be fixed arbitrarily, where $x \in C_{i_x}$ and $y \in C_{i_y}$, for some $i_x, i_y \in \{0, \ldots, k-1\}$. If $i_x = i_y$, the following holds for some $\sigma_{\Box}(x,y) \in \Sigma^{\mathcal{A}}_{\Box}$ (because $C_{i_x} = C_{i_y}$ is strongly-trap-connected):

$$\sigma_{\Box}(x,y) : x \overset{C_{i_x} = C_{i_y}}{\rightsquigarrow} y.$$

Otherwise, $i_x \neq i_y$. Then, $\sigma_{\Box}(x, u_{i_x}) : x \overset{C_{i_x}}{\rightsquigarrow} u_{i_x}$ for some $\sigma_{\Box}(x, u_{i_x}) \in \Sigma^{\mathcal{A}}_{\Box}$ (because C_{i_x} is strongly-trap-connected). Next, this holds for each $i = i_{i_x}, i_{i_x} + 1, \ldots, k -$

$1,0,1,\ldots,i_y-1$ and for $i' \triangleq (i+1) \mod k$:

$$\sigma_{\Box}(u_i,u_{i'}) : u_i \overset{C_i \cup \{u_{i'}\}}{\rightsquigarrow} u_{i'}.$$

Finally, $\sigma_{\Box}(u_{i_y},y) : u_{i_y} \overset{C_{i_y}}{\rightsquigarrow} y$ for some $\sigma_{\Box}(u_{i_y},y) \in \Sigma_{\Box}^{\mathcal{A}}$ (because C_{i_y} is strongly-trap-connected). Therefore, by composition, there exists $\sigma_{\Box}(x,y) \in \Sigma_{\Box}^{\mathcal{A}}$ such that $\sigma_{\Box}(x,y) : x \overset{C^*}{\rightsquigarrow} y$; so C^* is strongly-trap-connected. At this point, for every $i \in \{0,\ldots,k-1\}$, since $C_i \subseteq C^*$, since C^* is strongly-trap-connected, and since C_i is a STCC of $\mathcal{A}$ (so the maximality property mentioned in Def. 5.3 holds), then, $C_i = C^*$. $\Box$

Proposition 5.7. *Let $J_{\mathcal{A}}$ be the tr-jungle constructed by* `tr-DFS`*$(\mathcal{A})$ (Algo. 12) (see Proposition 5.3). Let $\{\mathcal{P}^i\}_{i=1}^k$ be the tr-palm-tree's family of $J_{\mathcal{A}}$, for some $k \in \mathbb{N}$, where $\mathcal{P}^i \triangleq (V^i, A^i, (V_{\Box} \cap V^i, V_{\bigcirc} \cap V^i))$ and $A^i \triangleq A^i_{tree} \uplus A^i_{frond} \uplus A^i_{petiole} \uplus A^i_{cross}$; also, $F \triangleq (V, \bigcup_{i=0}^k A^i_{tree})$ is a forest by Defs. 5.5-5.6. Let $\mathcal{C} \subseteq V$ be any STCC of $\mathcal{A}$. Then, $\mathcal{C}$ induces a subtree $T_{\mathcal{C}}$ in F (i.e., $F[\mathcal{C}]$ is an inward directed rooted tree).*

Proof. By Proposition 5.2, each $v \in V$ is numbered by `idx` exactly once. Let $v^* \triangleq \arg\min_{x \in \mathcal{C}} \texttt{idx}[x]$ be the first vertex $v^* \in \mathcal{C}$ that is visited during the invocation of `tr-DFS`$(\mathcal{A})$ (Algo. 12). By Proposition 5.3 and Defs. 5.5-5.6, the set of all vertices that are visited during the whole (i.e., including recursive calls) execution of the subprocedure `tr-DFS-visit`$(v^*, \mathcal{A})$ induces a subtree $T_{v^*} \triangleq (V_{T_{v^*}}, A_{T_{v^*}})$ of F which is rooted at v^*; so, $v^* \in \mathcal{C} \cap T_{v^*} \neq \emptyset$.

<u>Fact-1</u>: $\mathcal{C} \subseteq T_{v^*}$. For the sake of contradiction, suppose $\mathcal{C} \setminus T_{v^*} \neq \emptyset$. Then, since $\mathcal{C}$ is strongly-trap-connected, there exists $\hat{u} \in \mathcal{C} \setminus T_{v^*}$ such that one of the following two holds: either (i) $\hat{u} \in V_{\Box}$ and there exists $u' \in N^{\text{out}}_{\mathcal{A}}(\hat{u})$ such that $u' \in \mathcal{C} \cap T_{v^*}$; or (ii) $\hat{u} \in V_{\bigcirc}$ and for all $u' \in N^{\text{out}}_{\mathcal{A}}(\hat{u})$ it holds that $u' \in \mathcal{C} \cap T_{v^*}$. Also notice, since $v^* \triangleq \arg\min_{x \in \mathcal{C}} \texttt{idx}[x]$, then $\texttt{idx}[v^*] < \texttt{idx}[\hat{u}]$; thus $\hat{u}$ was not visited before v^*. All these combined, by definition of `tr-DFS-visit()` (Proc. 2) and since T_v^* is a subtree of F, it must be that $\hat{u}$ is visited and that it joins F during the execution of `tr-DFS-visit`$(v^*, \mathcal{A})$; so, $\hat{u} \in T_{v^*}$. But this contradicts our assumption $\hat{u} \in \mathcal{C} \setminus T_{v^*}$. Therefore, $\mathcal{C} \setminus T_{v^*} = \emptyset$; so, $\mathcal{C} \subseteq T_{v^*}$.

<u>Fact-2</u>: If $u \in \mathcal{C} \setminus \{v^*\}$ and $u' \triangleq \pi_{T_{v^*}}(u)$, then $u' \in \mathcal{C}$. Indeed, since $u \in \mathcal{C}$ and $\mathcal{C} \subseteq T_{v^*}$, then u is a descendant of v^* in T_{v^*}. Since $u \neq v^*$, then u' is also a descendant of v^* in T_{v^*}. Thus, by Proposition 5.4, there exists some $\sigma_{\Box} \in \Sigma_{\Box}^{\mathcal{A}}$ such that:

$$\sigma_{\Box} : u \overset{T_{v^*}}{\rightsquigarrow} u' \text{ and } \sigma_{\Box} : u' \overset{T_{v^*}}{\rightsquigarrow} v^*;$$

thus, since $u, v^* \in \mathcal{C}$ and $\mathcal{C}$ is a STCC, then $u' \in \mathcal{C}$ holds by Lemma 5.3.

By *(fact-1)* and *(fact-2)*, $\mathcal{C}$ induces a tree $T_{\mathcal{C}}$ in F (i.e., $T_{\mathcal{C}}$ is a subtree of T_{v^*} still rooted at v^*). $\Box$

Definition 5.9. *The root v^* of the tree $T_{\mathcal{C}}$ (as in the proof of Proposition 5.7) is the* root *of the STCC $\mathcal{C}$.*

The problem of computing the STCCs of any arena $\mathcal{A}$ reduces to that of finding the roots of the STCCs; just as the classical problem of finding the SCCs of a directed graph reduced to that of finding the roots of the SCCs. We have identified a simple test to determine if a vertex is the root of a STCC. It is based on a *lowlink* indexing, generalizing the lowlink calculation performed by Tarjan's SCC algorithm [110].

Definition 5.10. *Let $J_{\mathcal{A}}$ be the tr-jungle constructed by $\texttt{tr-DFS}(\mathcal{A})$ (Algo. 12). Let $\texttt{idx}: V \to \{1,\ldots,|V|\}$ be the indexing computed during that execution, and let $\{\mathcal{P}^i\}_{i=1}^{k}$ be the tr-palm-tree's family of $J_{\mathcal{A}}$, for some $k \in \mathbb{N}$, where $\mathcal{P}^i \triangleq (V^i, A^i, (V_\Box \cap V^i, V_\bigcirc \cap V^i))$ and $A^i \triangleq A^i_{tree} \cup A^i_{frond} \cup A^i_{petiole} \cup A^i_{cross}$; also, $F \triangleq (V, \bigcup_{i=0}^{k} A^i_{tree})$ is a forest by Defs. 5.5-5.6.*

Given any $v \in V$, the tr-lowlink : $V \to \mathbb{N}$ is defined as follows:

$$
\begin{aligned}
\textit{tr-lowlink}(v) \triangleq \min\Big(&\big\{\texttt{idx}[v]\big\} \cup \\
&\cup \big\{\texttt{idx}[u] \mid u \in V \setminus \{v\} \text{ and } \exists_{i\in[k]} \text{ such that the following two hold:} \\
&\quad \text{(tr-ll-1) } \exists_{t\geq 1}\, \exists_{(u,v_1,\ldots,v_{t-1},(v_t=v))\in (V^i)^+} \text{such that:} \\
&\qquad \text{(a) } (u,v_1) \in A_{frond} \cup A_{cross}; \\
&\qquad \text{(b) if } t \geq 2, \forall_{j\in\{1,\ldots,t-1\}} \text{ it holds } (v_j, v_{j+1}) \in A_{tree}. \\
&\quad \text{(tr-ll-2) } \exists_{\gamma\in V^i} \text{ such that:} \\
&\qquad \text{(a) } \gamma \text{ is a common ancestor of } u \text{ and } v \text{ in } (V^i, A^i_{tree}); \\
&\qquad \text{(b) } \gamma \text{ and } u \text{ are in the same STCC of } \mathcal{A}. \\
&\quad \big\} \\
\Big).&
\end{aligned}
$$

If $v \in V$, $N^{in}_{\mathcal{A},LCA}(v) \triangleq \{u \in N^{in}_{\mathcal{A}}(v) \mid$ the LCA γ of $\{u,v\}$ in F exists and $\gamma \in \mathcal{C}_u\}$.

Let us prove some useful properties of tr-lowlink() and $N^{\text{in}}_{\mathcal{A},\text{LCA}}$.

Proposition 5.8. *Let $\mathcal{A}$, $\texttt{idx}$, F and tr-lowlink() be as in Def. 5.10. Given any $v \in V$:*

1. *If tr-lowlink$(v) = \texttt{idx}[u]$ for some $u \in V \setminus \{v\}$ such that* (tr-ll-1) *and* (tr-ll-2) *hold, then $u \in V_\Box$.*

2. *tr-lowlink$(v) = \min\big\{\texttt{idx}[v]\big\} \cup \big\{\texttt{idx}[u] \mid u \in N^{in}_{\mathcal{A},LCA}(v)\big\} \cup$*
   $\cup \big\{$tr-lowlink$(u) \mid u$ is a child of v in $F\big\}$.

Proof of (1). By Item (a) of (tr-ll-1), it holds $(u,v_1) \in A_{\text{frond}} \cup A_{\text{cross}}$. Recall that (u,v_1) can be added to $A_{\text{frond}} \cup A_{\text{cross}}$ only at lines 15-16 of the subprocedure $\texttt{tr-DFS-visit}(v_1,\mathcal{A})$ (Proc. 1). So, u was visited before v_1. Still, $u \in N^{\text{in}}_{\mathcal{A}}(v_1)$ by Item (a) of (tr-ll-1); then, it is not possible that $u \in V_\bigcirc$, because any $x \in V_\bigcirc$

can join F only if $\mathtt{cnt}[x] = 0$ holds at line 11 of `tr-DFS-visit()` (Proc. 1), and u was visited before v_1 and yet it joined F. Therefore, $u \in V_{\Box}$. $\Box$

Proof of (2). Assume that tr-lowlink$(v) = \mathtt{idx}[u]$ for some $u \in V \setminus \{v\}$ such that (tr-ll-1) and (tr-ll-2) hold. Then, $\exists_{t\geq 1} \exists_{(u,v_1,\ldots,v_{t-1},(v_t=v))\in(V^i)^+}$ as in Def. 5.10. If $t = 1$, then $v_1 = v$; so, $u \in N^{\text{in}}_{\mathcal{A},\text{LCA}}(v)$ (it is easy to see that, if (tr-ll-2) holds for *some* common ancestor of $\{u,v\}$ in F, then it holds for the LCA). If $t > 1$, then v_1 is a proper descendant of v in F. At this point, it is easy to check tr-lowlink$(v) = \mathtt{idx}[u] =$ tr-lowlink$(v_1) =$ tr-lowlink(c) holds for some child c of v in F, indeed, this follows from Def. 5.10 and Proposition 5.7. $\Box$

Similarly to Tarjan's lowlink based algorithm [110], the tr-lowlink(v) is thus the smallest index of any vertex u which is in the same STCC as v and such that u can reach v by traversing: at most one frond (i.e., A^i_{frond}) or cross-link (i.e., A^i_{cross}) arc by *item (a)* of *(tr-ll-1)*, then, zero or more tree (i.e., A^i_{tree}) arcs by *item (b)* of *(tr-ll-1)*.

What follows is of a pivotal importance for computing STCCs by relying on tr-lowlinks.

Proposition 5.9. *Let $J_{\mathcal{A}}$ be the tr-jungle constructed by `tr-DFS`$(\mathcal{A})$ (Algo. 12), and let `idx`$: V \to \{1,\ldots,|V|\}$ be the indexing constructed during that execution. Finally, let tr-lowlink $: V \to \mathbb{N}$ be as in Def. 5.10.*

For any $v \in V$, it holds that v is the root of some STCC of $\mathcal{A}$ if and only if tr-lowlink$(v) = \mathtt{idx}[v]$.

Proof. Let $v \in V$. By Proposition 5.7, the STCC $\mathcal{C}_v$ induces a subtree $T_{\mathcal{C}_v}$ in F. Let v^* be the root of $T_{\mathcal{C}_v}$.

($\Rightarrow$) Assume $v = v^*$. Then, we argue there can be *no* $u \in V \setminus \{v\}$ such that tr-lowlink$(v) = \mathtt{idx}[v]$, i.e., such that $\mathtt{idx}[u] < \mathtt{idx}[v]$ and both *(tr-ll-1)* and *(tr-ll-2)* (see Def. 5.10) hold on u. For the sake of contradiction, assume the existence of such an u. Then, since $\mathtt{idx}[u] < \mathtt{idx}[v]$ and v is the root of $T_{\mathcal{C}_v}$, it would be $u \notin \mathcal{C}_v$. By *(tr-ll-1)* (Def. 5.10), there exists a path $\langle u,v_1,\ldots,v_{t-1},(v_t = v)\rangle$ in $\mathcal{A}$, for some $t \geq 1$, such that $(u,v_1) \in A_{\text{frond}} \uplus A_{\text{cross}}$ and, if $t \geq 2$, $\forall_{j\in\{1,\ldots,t-1\}}$ it holds that $(v_j,v_{j+1}) \in A_{\text{tree}}$. Also, by *(tr-ll-2)* (Def. 5.10), there exists a common ancestor γ of u and v in $(V, A_{\mathtt{tree}})$, and $\gamma \in \mathcal{C}_u$. All these combined, by Proposition 5.4 and Lemma 5.3 (applied to the tr-cycle $v\gamma uv$), it would be $\mathcal{C}_v = \mathcal{C}_u$. This is absurd, because $u \in \mathcal{C}_u$ and $u \notin \mathcal{C}_v$. Indeed, there is no such u. Therefore, tr-lowlink$(v) = \mathtt{idx}[v]$.

($\Leftarrow$) Assume $v \neq v^*$. Since $\mathcal{C}_v$ is strongly-trap-connected and $v, v^* \in \mathcal{C}_v$, then $v^* \rightsquigarrow v$. Let T_v be the subtree of $T_{\mathcal{C}_v}$ that is rooted at v. Since $v^* \rightsquigarrow v$, there exists some $u \in V \setminus V_{T_v}$ (possibly, $u = v^*$) and some $v_1 \in T_v$ (possibly, $v_1 = v$) such that: (i) $v^* \rightsquigarrow u$ and $u \rightsquigarrow v_1$; plus, (ii) $(u,v_1) \in A_{\text{frond}} \uplus A_{\text{cross}}$. Since $u \notin T_v$ and $v_1 \in T_v$, and since $(u,v_1) \in A_{\text{frond}} \uplus A_{\text{cross}}$, then $u \in V_{\Box}$ and $\mathtt{idx}[u] < \mathtt{idx}[v]$. Moreover, since $v^* \rightsquigarrow u \rightsquigarrow v_1 \rightsquigarrow v$, and finally (by Proposition 5.4) $v \rightsquigarrow v^*$, then u is in the same STCC as $\{v^*,v_1,v\}$ (by Lemma 5.3, applied to the tr-cycle $v^*uv_1vv^*$); i.e., $u \in \mathcal{C}_v$. Thus, u satisfies both *(tr-ll-1)*, *(tr-ll-2)*, so tr-lowlink$(v) \leq$ `idx`$[u]$; and since `idx`$[u] <$ `idx`$[v]$, then tr-lowlink$(v) <$ `idx`$[v]$. □

When `tr-lowlink`$(v) =$ `idx`$[v]$ holds, as in Proposition 5.9, we say that v is a fixpoint of tr-lowlink(); so, given any arena $\mathcal{A}$, the roots of the STCCs of $\mathcal{A}$ are exactly the fixpoints of tr-lowlink().

Our algorithm for decomposing $\mathcal{A}$ into STCCs is described next, it is based on the DSF-based `tr-DFS()` (see Algo. 12); still, it has some additional and distinctive features:

(1) All vertices that have already been reached during the `tr-DFS()`, but whose STCC has not yet been completely identified, are stored on a stack `St`;

(2) The stack `St` is (partially) emptied, and a brand new STCC $\mathcal{C}$ is completely identified, when the tr-lowlink's fixpoint condition `tr-lowlink`$(v) =$ `idx`$[v]$ is met (see Proposition 5.9).

(3) The STCC algorithm does not build any tr-jungle's forest F explicitly (i.e., there is no real need to keep track of $A_{\text{tree}}, A_{\text{frond}}, A_{\text{petiole}}, A_{\text{cross}}$); still, a tr-jungle's forest F is defined implicitly, by the sequence of vertices that are visited and backtracked during the search process.

It will be convenient to consider this tr-jumgle during the proof of correctness, so let us refer to the corresponding (implicitly constructed) F as to the *STCC forest* (recall that it would have been $F \triangleq (V, A_{\text{tree}})$ in Algo. 12).

The STCCs main procedure is called `compute-STCCs()`, it takes in input an arena $\mathcal{A}$, and it aims at printing out all the STCCs $\mathcal{C}_1,\ldots,\mathcal{C}_k$ of $\mathcal{A}$ (w/o repetitions). A procedure named `STCCs-visit()` is also employed. The pseudo-code is given in Algo. 13 and Proc. 2, respectively.

The initialization phase goes from line 1 to 9 of Algo. 13. The visit-phase starts by setting `next_idx` $\leftarrow 1$ and `St` $\leftarrow \emptyset$ (lines 10-18 of Algo. 13). Firstly, $V_\Box$ is considered: for each unvisited $u \in V_\Box$, `STCCs-visit`$(u,\mathcal{A})$ is invoked. Then, all the $u \in V_\bigcirc$ which are still unvisited are handled as in Algo. 12.

Consider `STCCs-visit`$(v,\mathcal{A})$ (Proc. 2), for $v \in V$. This is similar to the DSF-based implementation of `tr-DFS-visit()` (Proc. 1), the significant changes going as follows. Initially, v is pushed on top of `St` and `on_Stack`$[v] \leftarrow$ `true` is set (lines 4-5).

Then, each time `STCCs-visit`$(u,\mathcal{A})$ is invoked recursively (line 9 and 25), for some $u \in N^{\text{in}}_{\mathcal{A}}(v)$:

$$\texttt{tr-lowlink}[v] \leftarrow \min(\texttt{tr-lowlink}[v], \texttt{tr-lowlink}[u])$$

is updated and $\mathcal{D}$.`Union`(u,v) is executed.

When exploring $N^{\text{in}}_{\mathcal{A}}(v)$ (line 6): If $u \in N^{\text{in}}_{\mathcal{A}}(v) \cap V_\bigcirc$ is unvisited (i.e., `idx`$[u] = +\infty$), and if `cnt`$[u] = 0$ holds, we seek for the LCA γ of $N^{\text{out}}_{\mathcal{A}}(u)$ in F, as in the DSF-based implementation of `tr-DFS-visit()`; but then u is pushed on `ready_St`$[\gamma]$ if and only if `on_Stack`$[\gamma] =$ `true` (i.e., there's no `active` array

Algorithm 13: Computing STCCs

```
Procedure compute-STCCs($\mathcal{A}$)
    input  : An arena $\mathcal{A} = (V, A, (V_\bigcirc, V_\Box))$.
    output: The STCCs of $\mathcal{A}$.
1   foreach $u \in V$ do
2       idx[$u$] $\leftarrow +\infty$;
3       tr-lowlink[$u$] $\leftarrow +\infty$;
4       on_Stack[$u$] $\leftarrow$ false;
5       $\mathcal{D}$.make_set($u$);
6       ready_St[$u$] $\leftarrow \emptyset$;
7       if $u \in V_\bigcirc$ then
8           low_ready[$u$] $\leftarrow +\infty$;
9           cnt[$u$] $\leftarrow |N^{\text{out}}_{\mathcal{A}}(u)|$;
10  next_idx $\leftarrow 1$; St $\leftarrow \emptyset$;
11  foreach $u \in V_\Box$ do
12      if idx[$u$] $= +\infty$ then
13          STCCs-visit($u, \mathcal{A}$);
14  foreach $u \in V_\bigcirc$ do
15      if idx[$u$] $= +\infty$ then
16          idx[$u$] $\leftarrow$ next_idx;
17          next_idx $\leftarrow$ next_idx $+ 1$;
18          ta_lowlink[$u$] $\leftarrow$ idx[$u$];
```

Algorithm 13: Computing STCCs

to make this decision anymore). Else, if $u \in N^{\text{in}}_{\mathcal{A}}(v)$ has been already visited (i.e., if idx$[u] \neq +\infty$), and if on_Stack$[u]$ = true, then tr-lowlink$[v] \leftarrow$ min(tr-lowlink$[v]$,idx$[u]$) is updated (lines 20-21).

After that, anyway, ready_St$[v]$ is emptied and checked as in the DSF-based tr-DFS-visit() (see lines 22-27). Finally, if tr-lowlink$[v]$ = idx$[v]$ (see Proposition 5.9), a new STCC $\mathcal{C}$ is constructed: so, yet another vertex u is removed from St and added to $\mathcal{C}$, until $u = v$; soon after that, $\mathcal{C}$ is printed out. This concludes the description of the STCCs algorithm (Algo. 13).

SubProcedure 2: The STCCs-visit() procedure.

```
   Procedure STCCs-visit($v$,$\mathcal{A}$)
       input : A vertex $v \in V$.
1      idx[$v$] ← next_idx;
2      tr-lowlink[$v$] ← next_idx;
3      next_idx ← next_idx + 1;
4      St.push($v$);
5      on_Stack[$v$] ← true
       // Check the in-neighbourhood of $v$
6      foreach $u \in N^{in}_{\mathcal{A}}(v)$ do
7          if idx[$u$] = $+\infty$ then
8              if $u \in V_\Box$ then
9                  STCCs-visit($u$,$\mathcal{A}$);
10                 tr-lowlink[$v$] ← min(tr-lowlink[$v$],tr-lowlink[$u$]);
11                 $\mathcal{D}$.Union($u$,$v$);
12             else
13                 low_ready[$u$] ← min(low_ready[$u$],idx[$v$]);
14                 cnt[$u$] ← cnt[$u$] − 1;
15                 if cnt[$u$] = 0 then
16                     low_$v$ ← the unique $x$ such that idx[$x$] = low_ready[$u$];
17                     $\gamma$ ← $\mathcal{D}$.find(low_$v$);
18                     if on_Stack[$\gamma$] = true then
19                         ready_St[$\gamma$].push($u$);
20         else if on_Stack[$u$] = true then
21             tr-lowlink[$v$] ← min(tr-lowlink[$v$],idx[$u$]);
       // Check the ready-stack of $v$, i.e., ready_St[$v$]
22     while ready_St[$v$] $\neq \emptyset$ do
23         $u$ ← ready_St[$v$].pop(); // $u \in V_\bigcirc$
24         if $\forall(x \in N^{out}_{\mathcal{A}}(u))$ on_Stack[$x$] = true then
25             SCCs-visit($u$,$\mathcal{A}$);
26             tr-lowlink[$v$] ← min(tr-lowlink[$v$],tr-lowlink[$u$]);
27             $\mathcal{D}$.union($u$,$v$);
       // Check whether a new STCC has to be constructed and printed to output
28     if tr-lowlink[$v$] = idx[$v$] then
29         $\mathcal{C}$ ← $\emptyset$;
30         repeat
31             $u$ ← St.pop();
32             on_Stack[$u$] ← false;
33             add $u$ to $\mathcal{C}$;
34         until $u = v$
35         output($\mathcal{C}$);
```

5.5.1 Proof of Correctness of Algo. 13

Firstly, we need to prove that Algo. 13 computes tr-lowlink() correctly.

Proposition 5.10. *Assume* `compute-STCCs`$(\mathcal{A})$ *(Algo. 13) is invoked.*
When it halts, it holds that $\forall_{v\in V}$ `tr-lowlink`$[v] =$ *tr-lowlink*(v).

Proof. Let us say that v is *active* from when `STCCs-visit`$(v,\mathcal{A})$ is invoked 'til it halts; and from when `STCCs-visit`$(v,\mathcal{A})$ finally halts hereafter, we say that v is *deactivated*. So, let $(v_1,\ldots,v_i,\ldots,v_{|V|})$ be the order in which the vertices in V are deactivated during `compute-STCCs`$(\mathcal{A})$ (Algo. 13).

For every $v \in V$, let's define:

$$N^{\text{in}}_{\mathcal{A},\texttt{St}}(v) \triangleq \big\{u \in N^{\text{in}}_{\mathcal{A}}(v) \mid u \in \texttt{St} \text{ at line 22 of } \texttt{STCCs-visit}(v,\mathcal{A}) \text{ (Proc. 2)}\big\}.$$

The proof proceeds by induction on $i = 1,\ldots,|V|$. Let F be the corresponding STCC forest.

Base Case: $i = 1$. Notice that v_1 is a leaf of some tr-palm-tree in F. In this case, `tr-lowlink`$[v_1]$ can be assigned only at line 21 of `STCCs-visit`$(v_1,\mathcal{A})$, particularly, as follows:

$$\texttt{tr-lowlink}[v_1] = \min\{\texttt{idx}[v_1]\} \cup \{\texttt{idx}[u] \mid u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1)\}. \qquad \text{(eq. 1)}$$

Since v_1 is the first vertex in V which is ever deactivated (particularly, v_1 is a leaf in F), then:

$$N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1) = \big\{u \in N^{\text{in}}_{\mathcal{A}}(v_1) \mid u \text{ is a proper ancestor of } v_1 \text{ in } F\big\}, \qquad \text{(eq. 2)}$$

for the same reason, plus Item 1 of Proposition 5.8, it holds that:

$$\text{tr-lowlink}(v_1) = \min\big\{\texttt{idx}[v_1]\big\} \uplus \big\{\texttt{idx}[u] \mid u \in N^{\text{in}}_{\mathcal{A}}(v_1) \cap \mathcal{C}_{v_1} \cap V_\Box\big\}. \qquad \text{(eq. 3)}$$

Observe, since v_1 is a leaf of F, and by (eq. 2) plus lines 8-11 of `STCCs-visit`$(v_1,\mathcal{A})$ (Proc. 2), it holds that $N^{\text{in}}_{\mathcal{A}}(v_1) \cap \mathcal{C}_{v_1} \cap V_\Box \subseteq N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1)$; also, by (eq. 2) and Proposition 5.4, $N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1) \subseteq N^{\text{in}}_{\mathcal{A}}(v_1) \cap \mathcal{C}_{v_1} \cap V_\Box$. Then, $N^{\text{in}}_{\mathcal{A}}(v_1) \cap \mathcal{C}_{v_1} \cap V_\Box = N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1)$. Therefore, by (eq. 1) and (eq. 3), `tr-lowlink`$[v_1] =$ tr-lowlink(v_1).

Inductive Step: $i > 1$. In this case, `tr-lowlink`$[v_i]$ can be assigned either at line 2 or 10 or 21 or 26 of `STCCs-visit`$(v_i,\mathcal{A})$, particularly, as follows:

$$\texttt{tr-lowlink}[v_i] = \min\big\{\texttt{idx}[v_1]\big\} \uplus \big\{\texttt{idx}[u] \mid u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v)\big\} \uplus \\ \uplus \big\{\texttt{tr-lowlink}[u] \mid u \text{ is a child of } v \text{ in } F\big\}.$$

On the other side, let:

$$N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i) \triangleq \{u \in N^{\text{in}}_{\mathcal{A}}(v_i) \mid \text{the LCA } \gamma \text{ of } \{u,v_i\} \text{ in } F \text{ exists and } \gamma \in \mathcal{C}_u\},$$

then the following holds by Item 2 of Proposition 5.8:

$$\text{tr-lowlink}(v_i) = \min\big\{\texttt{idx}[v_i]\big\} \uplus \big\{\texttt{idx}[u] \mid u \in N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)\big\} \\ \uplus \big\{\text{tr-lowlink}(u) \mid u \text{ is a child of } v_i \text{ in } F\big\}.$$

Notice that, if u is a child of v_i in F, then `compute-STCCs`$(\mathcal{A})$ deactivates u before v_i. Thus, by induction hypothesis, `tr-lowlink`$[u]$ = tr-lowlink(u) holds for every child u of v_i in F that is considered either at line 10 or 26 of `STCCs-visit`$(v_i,\mathcal{A})$. To finish the proof, we need to show that $N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i) = N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$. For this, let's recall the following facts. (i) Any vertex $v \in V$ can be added to `St` only at line 4 of `STCCs-visit`$(v,\mathcal{A})$. (ii) Any vertex can be removed from `St` only at line 31 of `STCCs-visit`$(v,\mathcal{A})$, for some $v \in V$, and only if `tr-lowlink`$[v]$ = `idx`$[v]$ at line 28; this (possibly) happens only after that $N^{\text{in}}_{\mathcal{A}}(v)$ has been fully explored at lines 6-21. With this in mind, we can proceed.

- Firstly, we show $N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i) \subseteq N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$. Let $u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i)$. Then, u and v_i lie within the same tr-palm-tree in F: this is easily seen by induction on the number of tr-palm-trees of F. Then u is a proper ancestor of v_i in F, i.e., $\gamma = u$; thus, $\gamma \in \mathcal{C}_u$. So, the LCA γ of $\{u,v_i\}$ in F exists if u is still active at line 22 of `STCCs-visit`$(v_i,\mathcal{A})$. Otherwise, u has already been deactivated when `STCCs-visit`$(v_i,\mathcal{A})$ reaches line 22 (so, $\gamma \neq u$); in this case, also every ancestor of u that is a proper descendant of γ in F has already been deactivated before. So, by induction hypothesis, `tr-lowlink`$[\hat{v}]$ = tr-lowlink$(\hat{v})$ for every ancestor $\hat{v}$ of u that is also proper descendant of γ in F. On the other hand, since $u \in$ `St` at line 22 of `STCCs-visit`$(v_i,\mathcal{A})$, all those $\hat{v}$ (including u) can't be already been removed from `St` when `STCCs-visit`$(v_i,\mathcal{A})$ reaches line 22. Therefore, by lines 28-33 of `STCCs-visit()`, by Proposition 5.9, and since `tr-lowlink`$[\hat{v}]$ = tr-lowlink$(\hat{v})$ for all of those $\hat{v}$, then none of those $\hat{v}$ can be the root of some STCC of $\mathcal{A}$. Thus, $\gamma \in \mathcal{C}_u$. So, $u \in N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$.

- Secondly, we show $N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i) \subseteq N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i)$. Let $u \in N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$, and let $\gamma \in \mathcal{C}_u$ be the LCA of $\{u,v_i\}$ in F. If u is still active at line 22 of `STCCs-visit`$(v_i,\mathcal{A})$, u is a proper ancestor of v_i in F (i.e., $\gamma = u$); so, $u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i)$. Otherwise, u has already been deactivated when `STCCs-visit`$(v_i,\mathcal{A})$ reaches line 22 (so, $\gamma \neq u$); also, in this case, every ancestor of u that is also a proper descendant of γ in F has already been deactivated before. So, by induction hypothesis, `tr-lowlink`$[\hat{v}]$ = tr-lowlink$(\hat{v})$ for every ancestor $\hat{v}$ of u that is also proper descendant of γ in F; but, then, since $\gamma \in \mathcal{C}_u$, all those $\hat{v}$ (including u) can't be already been removed from `St` when `STCCs-visit`$(v_i,\mathcal{A})$ reaches line 22. Therefore, $u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i)$.

So, $N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i) = N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$. □

Proposition 5.11. *Assume that* `compute-STCCs`$(\mathcal{A})$ *(Algo. 13) is invoked, and that* `STCCs-visit`$(v,\mathcal{A})$ *(Proc. 2) outputs* $\mathcal{C} \subseteq V$ *at line 35, for some* $v \in V$. *Then,* $\mathcal{C} = \mathcal{C}_v$.

Proof. Notice that `STCCs-visit`$(v,\mathcal{A})$ outputs $\mathcal{C}$ at line 35 if and only if `tr-lowlink`$[v] =$ `idx`$[v]$ holds at line 28; assume such a condition holds. Recall `tr-lowlink`$[u] =$ tr-lowlink(u) for every $u \in V$, by Proposition 5.10. We argue that $\mathcal{C} = \mathcal{C}_v$. Indeed, $\mathcal{C} =$ `St`(v) by lines 28-33 of `STCCs-visit`$(v,\mathcal{A})$ (Proc. 2). Firstly, $\mathcal{C}_v \subseteq$ `St`(v): in fact, by Proposition 5.7, $\mathcal{C}_v$ induces a subtree $T_{\mathcal{C}_v}$ in F (i.e., the STCC forest) which is rooted at v, so all vertices in $\mathcal{C}_v$ must have been inserted into `St` at this point; and notice no vertex of $\mathcal{C}_v$ could have been removed earlier, as removals happen only at the root v of $T_{\mathcal{C}_v}$ (by Proposition 5.9 and lines 28-33 of `STCCs-visit()`). Secondly, `St`$(v) \subseteq \mathcal{C}_v$: indeed, assume $u \notin \mathcal{C}_v$ has been inserted into `St` after v, then u is a descendant of v in F, and since $\mathcal{C}_u$ induces a subtree $T_{\mathcal{C}_u}$ in F, then u must have been removed from `St` when the root of $T_{\mathcal{C}_u}$ was visited, i.e., before that `STCCs-visit`$(v,\mathcal{A})$ reaches line 28. Therefore, `St`$(v) = \mathcal{C}_v$; and since $\mathcal{C} =$ `St`(v), then $\mathcal{C} = \mathcal{C}_v$. □

Proposition 5.12. *Let* $\mathcal{C} \subseteq V$ *be some STCC of* $\mathcal{A}$. *Then,* `compute-STCCs`$(\mathcal{A})$ *(Algo. 13) eventually outputs* $\mathcal{C}$ *at line 35.*

Proof. By Proposition 5.7, $\mathcal{C}$ induces a sub-tree $T_{\mathcal{C}}$ in F; then, let v^* be its root. When `STCCs-visit`$(v^*,\mathcal{A})$ is invoked, then:

$$\texttt{tr-lowlink}[v^*] = \texttt{tr-lowlink}(v^*) = \texttt{idx}[v^*]$$

holds at line 28 by Propositions 5.9 and 5.10.

Then, $\mathcal{C}_{v^*}$ is outputted at line 35, by lines 28-33 and Proposition 5.12. □

In summary, we obtain the following result.

Theorem 5.2. *Let* $\mathcal{A}$ *be an arena,* `compute-STCCs`$(\mathcal{A})$ *(Algo. 13) outputs all and only the STCCs of* $\mathcal{A}$.

5.6 Application to Update Games

An *Update Game (UG)* [10,45,46] is played on an arena $\mathcal{A}$ for an infinite number of rounds, a *play* is thus an infinite path $p = v_0 v_1 v_2 \ldots \in V^\omega$ such that $(v_i, v_{i+1}) \in A$ for every $i \in \mathbb{N}$. Let Inf(π) be the set of all and only those vertices $v \in V$ that appear infinitely often in π; namely,

$$\text{Inf}(\pi) \triangleq \{v \in V \mid \forall_{j \in \mathbb{N}} \, \exists_{k \in \mathbb{N}} \text{ such that } k > j \text{ and } v_k = v\}.$$

Player $\Box$ wins the UG $\mathcal{A}$ iff there exists $\sigma_\Box \in \Sigma_\Box^{\mathcal{A}}$ such that, for every $\sigma_\bigcirc \in \Sigma_\bigcirc^{\mathcal{A}}$, every vertex is visited infinitely often in the play consistent with $\sigma_\Box$ and $\sigma_\bigcirc$, independently w.r.t. the starting position $v_s \in V$; namely,

$$\forall_{v_s \in V} \operatorname{Inf}\big(\rho_{\mathcal{A}}(v_s, \sigma_\Box, \sigma_\bigcirc)\big) = V;$$

otherwise, Player $\bigcirc$ wins. When Player $\Box$ wins an UG $\mathcal{A}$, we say that $\mathcal{A}$ is an *Update Network (UN)* [10,45,46].

Proposition 5.13. *An UG $\mathcal{A}$ is UN if and only if V is strongly-trap-connected.*

Proof. If $\mathcal{A}$ is UN, then V is strongly-trap-connected (it follows directly from definitions). Let $i' \triangleq (i+1) \mod |V|$ for every $i \in \{0, \ldots, |V|-1\}$. Conversely, if $V = \{v_i\}_{i=0}^{|V|-1}$ is strongly-trap-connected, for every i there exists $\sigma_\Box(i) \in \Sigma_\Box^M$ such that $\sigma_\Box(i) : v_i \rightsquigarrow v_{i'}$. Starting from any v_i, Player $\Box$ can visit infinitely often all vertices in V simply by playing $\sigma_\Box(i), \sigma_\Box(i'), \sigma_\Box((i')'), \ldots$ in cascade; therefore, $\mathcal{A}$ is UN. $\Box$

So, we obtain the following main result.

Theorem 5.3. *Deciding whether a given UG $\mathcal{A}$ is UN takes $\Theta(|V| + |A|)$ time.*

Proof. On input $\mathcal{A}$, invoke `compute-STCCs`$(\mathcal{A})$ (Algorithm 13), and return `YES` if $\mathcal{A}$ has *only one* STCC; otherwise, $\mathcal{A}$ has at least two STCCs, so return `NO`. By Theorem 5.2 and Proposition 5.13, this correctly decides whether $\mathcal{A}$ is UN. By Proposition 5.6, the decision is made in $\Theta(|V| + |A|)$ time. $\Box$

Also, when the input UG is UN, Algorithm 13 is able to provide a winning strategy as shown next.

Theorem 5.4. *Algorithm 13 can be implemented so that, when `compute-STCCs`$(\mathcal{A})$ halts, and if the UG $\mathcal{A}$ is UN, it is returned a tr-palm-tree encoding routing information that an $O(|V|)$-space agent can consult to win the UG $\mathcal{A}$ in $O(1)$ time per move.*

Proof. During the execution of `compute-STCCs`$(\mathcal{A})$ (Algorithm 13), construct the STCC forest $F = (V_F, A_F)$ explicitly, as follows: $V_F = V$; whenever $\mathcal{D}$.`Union`(u,v) is executed at line 11 or line 27 of `STCCs-visit`$(v,\mathcal{A})$, add (u,v) to A_F (*tree-arcs*); also, if `on_Stack`$[u] =$ `true` holds at line 20 of `STCCs-visit`$(v,\mathcal{A})$, add (u,v) to A_F (*cross-links*). Define $\sigma_\Box \in \Sigma_\Box$ as follows: for each $u \in V_\Box$, the arcs $(u,v) \in A_F$ exiting from u are selected one at a time, one after the other; and when they have all been traversed once, the selection starts again, cyclically. Since $\mathcal{A}$ is UN, then $\mathcal{A}$ has only one single STCC by Proposition 5.13, so F comprises only one single tr-palm-tree T_F. We argue that, if $\mathcal{A}$ is UN, then $\sigma_\Box$ allows Player $\Box$ to win the UG $\mathcal{A}$. Let v_s be any starting position. For any $\sigma_\bigcirc \in \Sigma_\bigcirc$ and $I \triangleq \operatorname{Inf}\big(\rho_{\mathcal{A}}(v_s, \sigma_\Box, \sigma_\bigcirc)\big)$, it is not possible that $I \subsetneq V$: there can be no tree-arc nor cross-link going from some vertex $u \in I \cap V_\Box$ to some vertex in $V \setminus I$ (otherwise such an arc would have eventually been selected by $\sigma_\Box$); and there can be no $u \in I \cap V_{\bigcirc}$ such that $N^{\text{out}}_{\mathcal{A}}(u) \subseteq V \setminus I$. Thus, all vertices in $V \setminus I$ are descendants in T_F of some of those in I; but, since they are all descendants, there must be at least one cross-link going from I to in $V \setminus I$ (because $\mathcal{A}$ has only one single STCC), which is a contradiction. Therefore, $I = V$. Notice the size of T_F is $|T_F| = |V_{T_F}| + |A_{T_F}| = O(|V|)$, and $\sigma_{\Box}$ can be implemented with $O(|V|)$ additional memory (the total number of cross-links in T_F is less than $|V|$); so, $\sigma_{\Box}$ can be implemented with $O(|V|)$ space. By handling pointers in a suitable way, the time spent for each single move of $\sigma_{\Box}$ is $O(1)$. $\Box$

5.7 Application to Explicit McNaughton-Müller Games

McNaughton-Müller Games (MGs) provide a useful model for the synthesis of controllers in reactive systems, but their complexity depends on the representation of the winning conditions [66]. The most straightforward way to represent a (regular; see [66]) winning condition $\mathcal{F} \subseteq 2^V$ is to provide an explicit list of subsets of vertices, i.e., $\mathcal{F} = \{\mathcal{F}_i \subseteq V \mid 1 \leq i \leq \ell\}$, for some $\ell \in \mathbb{N}$. A play $\rho \in V^{\omega}$ is winning for Player $\Box$ if and only if $\text{Inf}(\rho) \in \mathcal{F}$. So, *Explicit MGs (E-MGs)* can be solved in polynomial time, where an effective algorithm is given in [66]. Concerning time complexity, given an input arena $\mathcal{A}$ and explicit winning condition $\mathcal{F}$, there are at most $|\mathcal{F}|$ loops in a run of that algorithm, and the most time consuming operation at each iteration is to decide an UG of size at most $|\mathcal{A}| + |\mathcal{F}|$. By Theorem 5.3, we can decide such an UG in $\Theta(|\mathcal{A}| + |\mathcal{F}|)$ time. So the E-MG algorithm given in [66] is improved by a factor $|\mathcal{A}| + |\mathcal{F}|$ (i.e., from cubic to quadratic time). In summary, we obtain the following result.

Theorem 5.5. *Deciding the winner in a given E-MG $(\mathcal{A}, \mathcal{F})$ takes time:*

$$O\big(|\mathcal{F}| \cdot (|\mathcal{A}| + |\mathcal{F}|)\big).$$

5.8 Conclusion

This work presented an algorithm for solving Update Games in linear time. With this, also the polynomial-time complexity of deciding Explicit McNaughton-Müller Games improves, from cubic to quadratic. The result was obtained by: (a) introducing a refined notion of reachability for arenas, named *trap-reachability*; (b) showing that every arena decomposes into *strongly-trap-connected components (STCCs)*; (c) devising a linear time algorithm for computing this unique decomposition.

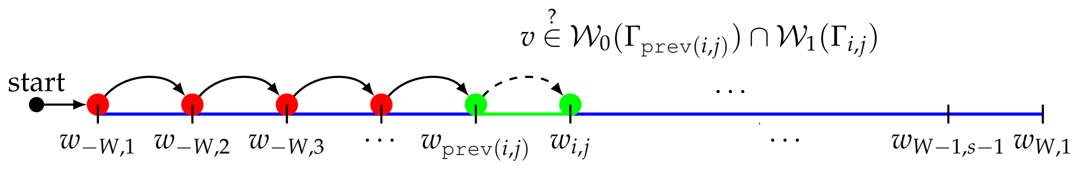
We expect that trap-reachability, and the corresponding linear time STCCs' decomposition, can play a role for speeding up computations in other problems concerning infinite 2-player pebble games.

Future works will likely investigate further on this way.

6 Improved Pseudo-Polynomial Upper Bound for the Value Problem and Optimal Strategy Synthesis in Mean Payoff Games

Chapter Abstract

In this chapter we offer a $\Theta(|V|^2|E|\,W)$ pseudo-polynomial time and $\Theta(|V|)$ space deterministic algorithm for solving the Value Problem and Optimal Strategy Synthesis in Mean Payoff Games. This improves by a factor $\log(|V|\,W)$ the best previously known pseudo-polynomial time upper bound due to Brim, *et al.* The improvement hinges on a suitable characterization of values, and a description of optimal positional strategies, in terms of reweighted Energy Games and Small Energy-Progress Measures.



An illustration of the main MPG algorithm presented in Chapter 6.

This chapter is a revised version of [35, 38].

6.1 Introduction

A *Mean Payoff Game* (MPG) is a two-player infinite game $\Gamma \triangleq (V, E, w, \langle V_0, V_1 \rangle)$, played on a finite weighted directed graph, denoted $G^\Gamma \triangleq (V, E, w)$, the vertices of which are partitioned into two classes, V_0 and V_1, according to the player to which they belong. It is assumed that G^Γ has no sink vertex and that the weights of the arcs are integers, i.e., $w : E \to \{-W, \dots, 0, \dots, W\}$ for some $W \in \mathbb{N}$.

At the beginning of the game a pebble is placed on some vertex $v_s \in V$, and then the two players, named Player 0 and Player 1, move the pebble ad infinitum along the arcs. Assuming the pebble is currently on Player 0's vertex v, then he chooses an arc $(v, v') \in E$ going out of v and moves the pebble to the destination vertex v'. Similarly, assuming the pebble is currently on Player 1's vertex, then it is her turn to choose an outgoing arc. The infinite sequence $v_s, v, v' \dots$ of all the encountered vertices is a *play*. In order to play well, Player 0 wants to maximize the limit inferior of the long-run average weight of the traversed arcs, i.e., to maximize $\liminf_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} w(v_i, v_{i+1})$, whereas Player 1 wants to minimize the $\limsup_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} w(v_i, v_{i+1})$. Ehrenfeucht and Mycielski [49] proved that each vertex v admits a *value*, denoted $\texttt{val}^\Gamma(v)$, which each player can secure by means of a *memoryless* (or *positional*) strategy, i.e., a strategy that depends only on the current vertex position and not on the previous choices.

Solving an MPG consists in computing the values of all vertices (*Value Problem*) and, for each player, a positional strategy that secures such values to that player (*Optimal Strategy Synthesis*). The corresponding decision problem lies in $\mathsf{NP} \cap \mathsf{coNP}$ [123] and it was later shown by Jurdziński [70] to be recognizable with *unambiguous* polynomial time non-deterministic Turing Machines, thus falling within the $\mathsf{UP} \cap \mathsf{coUP}$ complexity class.

The problem of devising efficient algorithms for solving MPGs has been studied extensively in the literature. The first milestone was that of Gurvich, Karzanov and Khachiyan [64], in which they offered an *exponential* time algorithm for solving a slightly wider class of MPGs called *Cyclic Games*. Afterwards, Zwick and Paterson [123] devised the first deterministic procedure for computing values in MPGs, and optimal strategies securing them, within a *pseudo-polynomial* time and polynomial space. In particular, Zwick and Paterson established an $O(|V|^3|E|\,W)$ upper bound for the time complexity of the Value Problem, as well as $O(|V|^4|E|\,W \log(|E|/|V|))$ for that of Optimal Strategy Synthesis [123].

Recently, several research efforts have been spent in studying quantitative extensions of infinite games for modeling quantitative aspects of reactive systems [12, 14, 18]. In this context quantities may represent, for example, the power usage of an embedded component, or the buffer size of a networking element. These studies have brought to light interesting connections with MPGs. Remarkably, they have recently led to the design of faster procedures for solving them. particularly, Brim, *et al.* [14] devised faster deterministic algorithms for solving the Value Problem and Optimal Strategy Synthesis in MPGs in $O(|V|^2|E|W\log(|V|W))$ pseudo-polynomial time and polynomial space.

To the best of our knowledge, this is the tightest pseudo-polynomial upper bound on the time complexity of MPGs which is currently known.

Indeed, a wide spectrum of different approaches have been investigated in the literature. For instance, Andersson and Vorobyov [1] provided a fast *sub-exponential* time *randomized* algorithm for solving MPGs, whose time complexity can be bounded as $O(|V|^2|E|\exp(2\sqrt{|V|\ln(|E|/\sqrt{|V|})} + O(\sqrt{|V|} + \ln|E|)))$. Furthermore, Lifshits and Pavlov [79] devised an $O(2^{|V|}\,|V|\,|E|\,\log W)$ *singly-exponential* time deterministic procedure by considering the *potential theory* of MPGs.

These results are summarized in Table 7.1.

Table 6.1: Complexity of the main Algorithms for solving MPGs.

Algorithm	Value Problem	Optimal Strategy Synthesis	Note
This work	$\Theta(|V|^2|E|W)$	$\Theta(|V|^2|E|W)$	Determ.
[14]	$O(|V|^2|E|W\log(|V|W))$	$O(|V|^2|E|W\log(|V|W))$	Determ.
[123]	$\Theta(|V|^3|E|W)$	$\Theta(|V|^4|E|W\log\frac{|E|}{|V|})$	Determ.
[79]	$O(2^{|V|}\,|V|\,|E|\,\log W)$	n/a	Determ.
[1]	$O\Big(|V|^2|E|e^{2\sqrt{|V|\ln\left(\frac{|E|}{\sqrt{|V|}}\right)}+O(\sqrt{|V|}+\ln|E|)}\Big)$	same complexity	Random.

6.1.1 Contribution

The main contribution of this chapter is to provide a $\Theta(|V|^2|E|W)$ *pseudo-polynomial* time and $\Theta(|V|)$ space deterministic algorithm for solving the Value Problem and Optimal Strategy Synthesis in MPGs. As already mentioned in the introduction, the best previously known procedure has a deterministic time complexity of $O(|V|^2|E|W\log(|V|W))$, which is due to Brim, *et al.* [14]. In this way we improve the best previously known pseudo-polynomial time upper bound by a factor $\log(|V|W)$. This result is summarized in the following theorem.

Theorem 6.1. *There exists a deterministic algorithm for solving the Value Problem and Optimal Strategy Synthesis of MPGs within $\Theta(|V|^2|E|W)$ time and $\Theta(|V|)$ space, on any input MPG $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$. Here, $W = \max_{e\in E}|w_e|$.*

In order to prove Theorem 6.1, this work points out a novel and suitable characterization of values, and a description of optimal positional strategies, in terms of certain reweighting operations that we will introduce later on in Section 6.2.

In particular, we will show that the optimal value $\texttt{val}^{\Gamma}(v)$ of any vertex v is the unique rational number ν for which v *"transits"* from the winning region of Player 0 to that of Player 1, with respect to reweightings of the form $w - \nu$. This intuition will be clarified later on in Section 6.3, where Theorem 6.3 is formally proved.

Concerning strategies, we will show that an optimal positional strategy for each vertex $v \in V_0$ is given by any arc $(v, v') \in E$ which is compatible with certain *Small Energy-Progress Measures* (SEPMs) of the above mentioned reweighted arenas. This fact is formally proved in Theorem 6.4 of Section 6.3.

These novel observations are smooth, simple, and their proofs rely on elementary arguments. We believe that they contribute to clarifying the interesting relationship between values, optimal strategies and reweighting operations (with respect to some previous literature, see e.g. [14, 79]). Indeed, they will allow us to prove Theorem 6.1.

6.1.2 Organization

This chapter is organized as follows. In Section 6.2, we introduce some notation and provide the required background on infinite two-player games and related algorithmic results. In Section 6.3, a suitable relation between values, optimal strategies, and certain reweighting operations is investigated. In Section 6.4, a $\Theta(|V|^2|E|\,W)$ pseudo-polynomial time and $\Theta(|V|)$ space algorithm, for solving the Value Problem and Optimal Strategies Synthesis in MPGs, is designed and analyzed by relying on the results presented in Section 6.3. In this manner, Section 6.4 actually provides a proof of Theorem 6.1 which is our main result in this work.

6.2 Background and Notation

We denote by $\mathbb{N}$, $\mathbb{Z}$, $\mathbf{Q}$ the set of natural, integer, and rational numbers (respectively). It will be sufficient to consider integral intervals, e.g., $[a, b] \triangleq \{z \in \mathbb{Z} \mid a \leq z \leq b\}$ and $[a, b) \triangleq \{z \in \mathbb{Z} \mid a \leq z < b\}$ for any $a, b \in \mathbb{Z}$.

Weighted Graphs. Our graphs are directed and weighted on the arcs. Thus, if $G = (V, E, w)$ is a graph, then every arc $e \in E$ is a triplet $e = (u, v, w_e)$, where $w_e = w(u, v) \in \mathbb{Z}$ is the *weight* of e. The maximum absolute weight is $W \triangleq \max_{e \in E} |w_e|$. Given a vertex $u \in V$, the set of its successors is $\texttt{post}(u) = \{v \in V \mid (u, v) \in E\}$, whereas the set of its predecessors is $\texttt{pre}(u) = \{v \in V \mid (v, u) \in E\}$. A *path* is a sequence of vertices $v_0 v_1 \ldots v_n \ldots$ such that $(v_i, v_{i+1}) \in E$ for every i. We denote by V^* the set of all (possibly empty) finite paths. A *simple path* is a finite path $v_0 v_1 \ldots v_n$ having no repetitions, i.e., for any $i, j \in [0, n]$ it holds $v_i \neq v_j$ whenever $i \neq j$. The *length* of a simple path $\rho = v_0 v_1 \ldots v_n$ equals n and it is denoted by $|\rho|$. A *cycle* is a path $v_0 v_1 \ldots v_{n-1} v_n$ such that $v_0 \ldots v_{n-1}$ is simple and $v_n = v_0$. The *length* of a cycle $C = v_0 v_1 \ldots v_n$ equals n and it is denoted by $|C|$. The *average weight* of a cycle $v_0 \ldots v_n$ is $\frac{1}{n} \sum_{i=0}^{n-1} w(v_i, v_{i+1})$. A cycle $C = v_0 v_1 \ldots v_n$ is *reachable* from v in G if there exists a simple path

$p = vu_1 \ldots u_m$ in G such that $p \cap C \neq \emptyset$.

Arenas. The reader is referred to Chapter 1, section 1.4, in order to recall the notion of *arena*. Here, that of *reweighting* (or *reweighted arena*) is formalized, as follows. For any weight function $w, w' : E \to \mathbb{Z}$, the *reweighting* of $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ w.r.t. w' is the arena $\Gamma^{w'} = (V, E, w', \langle V_0, V_1 \rangle)$. Also, for $w : E \to \mathbb{Z}$ and any $\nu \in \mathbb{Z}$, we denote by $w + \nu$ the weight function w' defined as $w'_e \triangleq w_e + \nu$ for every $e \in E$. Indeed, we shall consider reweighted games of the form Γ^{w-q}, for some $q \in \mathbf{Q}$. Notice that the corresponding weight function $w' : E \to \mathbf{Q} : e \mapsto w_e - q$ is rational, while we required the weights of the arcs to be always integers. To overcome this issue, it is sufficient to re-define Γ^{w-q} by scaling all the weights by a factor equal to the denominator of $q \in \mathbf{Q}$, namely, to re-define: $\Gamma^{w-q} \triangleq \Gamma^{D \cdot w - N}$, where $N, D \in \mathbb{N}$ are such that $q = N/D$ and $\gcd(N, D) = 1$. This re-scaling will not change the winning regions of the corresponding games, and it has the significant advantage of allowing for a discussion (and an algorithmics) which is strictly based on integer weights.

Mean Payoff Games. A *Mean Payoff Game* (MPG) [14,49,123] is a game played on some arena Γ for infinitely many rounds by two opponents, Player 0 gains a payoff defined as the long-run average weight of the play, whereas Player 1 loses that value. Formally, the Player 0's *payoff* of a play $v_0 v_1 \ldots v_n \ldots$ in Γ is defined as follows:

$$\mathtt{MP}_0(v_0 v_1 \ldots v_n \ldots) \triangleq \liminf_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n-1} w(v_i, v_{i+1}).$$

The value *secured* by a strategy $\sigma_0 \in \Sigma_0$ in a vertex v is defined as:

$$\mathtt{val}^{\sigma_0}(v) \triangleq \inf_{\sigma_1 \in \Sigma_1} \mathtt{MP}_0\big(\mathtt{outcome}^{\Gamma}(v, \sigma_0, \sigma_1)\big),$$

Notice that payoffs and secured values can be defined symmetrically for the Player 1 (i.e., by interchanging the symbol *0* with *1* and *inf* with *sup*).

Ehrenfeucht and Mycielski [49] proved that each vertex $v \in V$ admits a unique *value*, denoted $\mathtt{val}^{\Gamma}(v)$, which each player can secure by means of a *memoryless* (or *positional*) strategy. Moreover, *uniform* positional optimal strategies do exist for both players, in the sense that for each player there exist at least one positional strategy which can be used to secure all the optimal values, independently with respect to the starting position v_s. Thus, for every MPG Γ, there exists a strategy $\sigma_0 \in \Sigma_0^M$ such that $\mathtt{val}^{\sigma_0}(v) \geq \mathtt{val}^{\Gamma}(v)$ for every $v \in V$, and there exists a strategy $\sigma_1 \in \Sigma_1^M$ such that $\mathtt{val}^{\sigma_1}(v) \leq \mathtt{val}^{\Gamma}(v)$ for every $v \in V$. Indeed, the *(optimal) value* of a vertex $v \in V$ in the MPG Γ is given by:

$$\mathtt{val}^{\Gamma}(v) = \sup_{\sigma_0 \in \Sigma_0} \mathtt{val}^{\sigma_0}(v) = \inf_{\sigma_1 \in \Sigma_1} \mathtt{val}^{\sigma_1}(v).$$

Thus, a strategy $\sigma_0 \in \Sigma_0$ is *optimal* if $\mathtt{val}^{\sigma_0}(v) = \mathtt{val}^{\Gamma}(v)$ for all $v \in V$. A strategy $\sigma_0 \in \Sigma_0$ is said to be *winning* for Player 0 if $\mathtt{val}^{\sigma_0}(v) \geq 0$, and $\sigma_1 \in \Sigma_1$ is winning for Player 1 if $\mathtt{val}^{\sigma_1}(v) < 0$. Correspondingly, a vertex $v \in V$ is a *winning starting position* for Player 0 if $\mathtt{val}^{\Gamma}(v) \geq 0$, otherwise it is winning for Player 1. The set of all winning starting positions of Player i is denoted by $\mathcal{W}_i$ for $i \in \{0,1\}$.

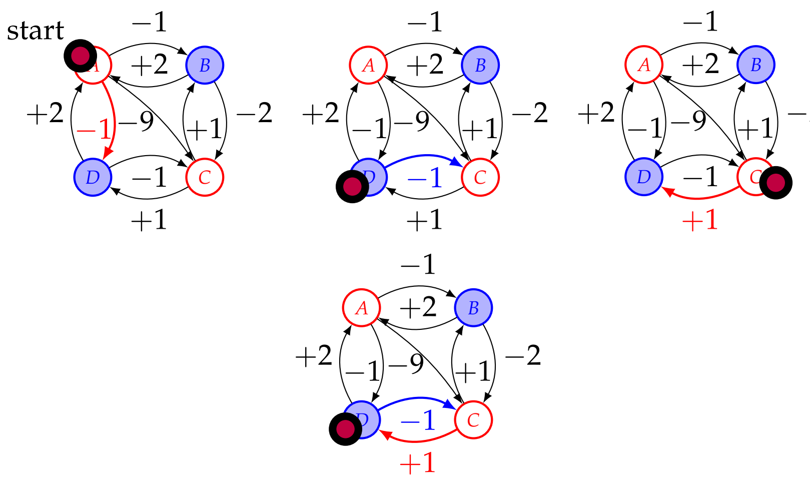


Figure 6.1: An MPG on Γ, played from left to right, whose payoff equals $\frac{-1+1}{2} = 0$.

A finite variant of MPGs is well-known in the literature [14, 49, 123]. Here, the game stops as soon as a cyclic sequence of vertices is traversed (i.e., as soon as one of the two players moves the pebble into a previously visited vertex). It turns out that this finite variant is equivalent to the infinite one [49]. Specifically, the values of an MPG are in relationship with the average weights of its cycles, as stated in the next lemma.

Lemma 6.1 (Brim, *et al.* [14]). *Let $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ be an MPG. For all $\nu \in \mathbf{Q}$, for all positional strategies $\sigma_0 \in \Sigma_0^M$ of Player 0, and for all vertices $v \in V$, the value $\mathtt{val}^{\sigma_0}(v)$ is greater than ν if and only if all cycles C reachable from v in the projection graph $G^{\Gamma}_{\sigma_0}$ have an average weight $w(C)/|C|$ greater than ν.*

The proof of Lemma 6.1 follows from the memoryless determinacy of MPGs. We remark that a proposition which is symmetric to Lemma 6.1 holds for Player 1 as well: for all $\nu \in \mathbf{Q}$, for all positional strategies $\sigma_1 \in \Sigma_1^M$ of Player 1, and for all vertices $v \in V$, the value $\mathtt{val}^{\sigma_1}(v)$ is less than ν if and only if all cycles reachable from v in $G^{\Gamma}_{\sigma_1}$ have an average weight less than ν.

Also, it is well-known [14, 49] that each value $\mathtt{val}^{\Gamma}(v)$ is contained within the following set of rational numbers:

$$S_\Gamma = \left\{ \frac{N}{D} \;\middle|\; D \in [1, |V|], N \in [-DW, DW] \right\}.$$

Notice that $|S_\Gamma| \leq |V|^2 W$.

This chapter tackles on the algorithmics of the following two classical problems:

- *Value Problem.* Compute $\texttt{val}^{\Gamma}(v)$ for each $v \in V$.
- *Optimal Strategy Synthesis.* Compute an optimal $\sigma_0 \in \Sigma_0^M$.

Previously, the asymptotically fastest pseudo-polynomial time algorithm for solving both problems was a deterministic procedure whose time complexity has been bounded as $O(|V|^2|E|W\log(|V|W))$ [14]. This result has been achieved by devising a binary-search procedure that ultimately reduces the Value Problem and Optimal Strategy Synthesis to the resolution of yet another family of games known as the *Energy Games*. Even though we do not rely on binary-search in the present work, and thus we will introduce some truly novel ideas that diverge from the previous solutions, still, we will reduce to solving multiple instances of Energy Games. For this reason, the Energy Games are recalled in the next paragraph.

Energy Games and Small Energy-Progress Measures. An *Energy Game* (EG) is a game that is played on an arena Γ for infinitely many rounds by two opponents, where the goal of Player 0 is to construct an infinite play $v_0v_1\ldots v_n\ldots$ such that for some initial *credit* $c \in \mathbb{N}$ the following holds:

$$c + \sum_{i=0}^{j} w(v_i, v_{i+1}) \geq 0, \text{ for all } j \geq 0. \tag{6.1}$$

Given a credit $c \in \mathbb{N}$, a play $v_0v_1\ldots v_n\ldots$ is *winning* for Player 0 if it satisfies (1), otherwise it is winning for Player 1. A vertex $v \in V$ is a winning starting position for Player 0 if there exists an initial credit $c \in \mathbb{N}$ and a strategy $\sigma_0 \in \Sigma_0$ such that, for every strategy $\sigma_1 \in \Sigma_1$, the play $\texttt{outcome}^{\Gamma}(v, \sigma_0, \sigma_1)$ is winning for Player 0. As in the case of MPGs, the EGs are memoryless determined [14], i.e., for every $v \in V$, either v is winning for Player 0 or v is winning for Player 1, and (uniform) memoryless strategies are sufficient to win the game. In fact, as shown in the next lemma, the decision problems of MPGs and EGs are intimately related.

Lemma 6.2 (Brim, *et al.* [14])**.** *Let $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$ be an arena. For all threshold $\nu \in \mathbf{Q}$, for all vertices $v \in V$, Player 0 has a strategy in the MPG Γ that secures value at least ν from v if and only if, for some initial credit $c \in \mathbb{N}$, Player 0 has a winning strategy from v in the reweighted EG $\Gamma^{w-\nu}$.*

In this work we are especially interested in the *Minimum Credit Problem* (MCP) for EGs: for each winning starting position v, compute the minimum initial credit $c^* = c^*(v)$ such that there exists a winning strategy $\sigma_0 \in \Sigma_0^M$ for Player 0 starting from v. A fast pseudo-polynomial time deterministic procedure for solving MCPs comes from [14].

Theorem 6.2 (Brim, *et al.* [14]). *There exists a deterministic algorithm for solving the MCP within $O(|V|\,|E|\,W)$ pseudo-polynomial time, on any input EG $(V,E,w,\langle V_0,V_1\rangle)$.*

The algorithm mentioned in Theorem 6.2 is the *Value-Iteration* algorithm analyzed by Brim, *et al.* in [14]. Its rationale relies on the notion of *Small Energy-Progress Measures* (SEPMs). These are bounded, non-negative and integer-valued functions that impose local conditions to ensure global properties on the arena, in particular, witnessing that Player 0 has a way to enforce conservativity (i.e., non-negativity of cycles) in the resulting game's graph. Recovering standard notation, see e.g. [14], let us denote $\mathcal{C}_\Gamma = \{n \in \mathbb{N} \mid n \leq |V|\,W\} \cup \{\top\}$ and let $\preceq$ be the total order on $\mathcal{C}_\Gamma$ defined as $x \preceq y$ if and only if either $y = \top$ or $x,y \in \mathbb{N}$ and $x \leq y$.

In order to cast the minus operation to range over $\mathcal{C}_\Gamma$, let us consider an operator $\ominus : \mathcal{C}_\Gamma \times \mathbb{Z} \to \mathcal{C}_\Gamma$ defined as follows:

$$a \ominus b \triangleq \begin{cases} \max(0, a-b) & \text{, if } a \neq \top \text{ and } a - b \leq |V|\,W; \\ a \ominus b = \top & \text{, otherwise.} \end{cases}$$

Given an EG Γ on vertex set $V = V_0 \cup V_1$, a function $f : V \to \mathcal{C}_\Gamma$ is a *Small Energy-Progress Measure* (SEPM) for Γ if and only if the following two conditions are met:

1. if $v \in V_0$, then $f(v) \succeq f(v') \ominus w(v,v')$ for *some* $(v,v') \in E$;
2. if $v \in V_1$, then $f(v) \succeq f(v') \ominus w(v,v')$ for *all* $(v,v') \in E$.

The values of a SEPM, i.e., the elements of the image $f(V)$,are called the *energy levels* of f. It is worth to denote by $V_f = \{v \in V \mid f(v) \neq \top\}$ the set of vertices having finite energy. Given a SEPM f and a vertex $v \in V_0$, an arc $(v,v') \in E$ is said to be *compatible with f* whenever $f(v) \succeq f(v') \ominus w(v,v')$; moreover, a positional strategy $\sigma_0^f \in \Sigma_0^M$ is said to be *compatible with f* whenever for all $v \in V_0$, if $\sigma_0^f(v) = v'$ then $(v,v') \in E$ is compatible with f. Notice that, as mentioned in [14], if f and g are SEPMs, then so is the *minimum function* defined as: $h(v) = \min\{f(v), g(v)\}$ for every $v \in V$. This fact allows one to consider the *least* SEPM, namely, the unique SEPM $f^* : V \to \mathcal{C}_\Gamma$ such that, for any other SEPM $g : V \to \mathcal{C}_\Gamma$, the following holds: $f^*(v) \preceq g(v)$ for every $v \in V$. Also concerning SEPMs, we shall rely on the following lemmata. The first one relates SEPMs to the winning region $\mathcal{W}_0$ of Player 0 in EGs.

Lemma 6.3 (Brim, *et al.* [14]). *Let $\Gamma = (V,E,w,\langle V_0,V_1\rangle)$ be an EG.*

1. *If f is any SEPM of the EG Γ and $v \in V_f$, then v is a winning starting position for Player 0 in the EG Γ. Stated otherwise, $V_f \subseteq \mathcal{W}_0$;*
2. *If f^* is the least SEPM of the EG Γ, and v is a winning starting position for Player 0 in the EG Γ, then $v \in V_{f^*}$. Thus, $V_{f^*} = \mathcal{W}_0$.*

Also notice that the following bound holds on the energy levels of any SEPM (actually by definition of $\mathcal{C}_\Gamma$).

Lemma 6.4. *Let $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ be an EG. Let f be any SEPM of Γ. Then, for every $v \in V$ either $f(v) = \top$ or $0 \leq f(v) \leq |V|W$.*

Value-Iteration Algorithm. The algorithm devised by Brim, *et al.* for solving the MCP in EGs is known as *Value-Iteration* [14]. Given an EG Γ as input, the Value-Iteration aims to compute the least SEPM f^* of Γ. This simple procedure basically relies on a lifting operator δ. Given $v \in V$, the *lifting operator* $\delta(\cdot, v) : [V \to \mathcal{C}_\Gamma] \to [V \to \mathcal{C}_\Gamma]$ is defined by $\delta(f, v) = g$, where:

$$g(u) = \begin{cases} f(u) & \text{if } u \neq v \\ \min\{f(v') \ominus w(v,v') \mid v' \in \texttt{post}(v)\} & \text{if } u = v \in V_0 \\ \max\{f(v') \ominus w(v,v') \mid v' \in \texttt{post}(v)\} & \text{if } u = v \in V_1 \end{cases}$$

We also need the following definition. Given a function $f : V \to \mathcal{C}_\Gamma$, we say that f is *inconsistent* in v whenever one of the following two holds:

1. $v \in V_0$ and *for all* $v' \in \texttt{post}(v)$ it holds $f(v) \prec f(v') \ominus w(v, v')$;

2. $v \in V_1$ and *there exists* $v' \in \texttt{post}(v)$ such that $f(v) \prec f(v') \ominus w(v, v')$.

To start with, the Value-Iteration algorithm initializes f to the constant zero function, i.e., $f(v) = 0$ for every $v \in V$. Furthermore, the procedure maintains a list L of vertices in order to witness the inconsistencies of f. Initially, $v \in V_0 \cap \mathsf{L}$ if and only if all arcs going out of v are negative, while $v \in V_1 \cap \mathsf{L}$ if and only if v is the source of at least one negative arc. Notice that checking the above conditions takes time $O(|E|)$.

As long as the list L is nonempty, the algorithm picks a vertex v from L and performs the following:

1. Apply the lifting operator $\delta(f, v)$ to f in order to resolve the inconsistency of f in v;

2. Insert into L all vertices $u \in \texttt{pre}(v) \setminus \mathsf{L}$ witnessing a new inconsistency due to the increase of $f(v)$.

   (The same vertex can't occur twice in L, i.e., there are no duplicate vertices in L.)

The algorithm terminates when L is empty. This concludes the description of the Value-Iteration algorithm.

As shown in [14], the update of L following an application of the lifting operator $\delta(f, v)$ requires $O(|\texttt{pre}(v)|)$ time. Moreover, a single application of the lifting operator $\delta(\cdot, v)$ takes $O(|\text{post}(v)|)$ time at most. This implies that the algorithm can be implemented so that it will always halt within $O(|V||E|W)$ time (the reader is referred to [14] in order to grasp all the details of the proof of correctness and complexity).

Remark. The Value-Iteration procedure lends itself to the following basic generalization, which turns out to be of a pivotal importance in order to best suit our technical needs. Let f^* be the least SEPM of the EG Γ. Recall that, as a first step, the Value-Iteration algorithm initializes f to be the constant zero function. Here, we remark that it is not necessary to do that really. Indeed, it is sufficient to initialize f to be any function f_0 which bounds f^* from below, that is to say, to initialize f to any $f_0 : V \to \mathcal{C}_\Gamma$ such that $f_0(v) \preceq f^*(v)$ for every $v \in V$. Soon after, L can be initialized in a natural way: just insert v into L if and only if f_0 is inconsistent at v. This initialization still requires $O(|E|)$ time and it doesn't affect the correctness of the procedure.

So, in the rest of this work we shall assume to have at our disposal a procedure named `Value-Iteration()`, which takes as input an EG $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ and an *initial function* f_0 that bounds from below the least SEPM f^* of the EG Γ (i.e., such that $f_0(v) \preceq f^*(v)$ for every $v \in V$). Then, `Value-Iteration()` outputs the least SEPM f^* of the EG Γ within $O(|V|\,|E|\,W)$ time, working with $\Theta(|V|)$ space.

6.3 Values and Optimal Positional Strategies from Reweightings

This section aims to show that values and optimal positional strategies of MPGs admit a suitable description in terms of reweighted arenas, a crux step for solving the Value Problem and Optimal Strategy Synthesis in $\Theta(|V|^2|E|\,W)$ time.

6.3.1 On Optimal Values

A simple representation of values in terms of *Farey sequences* is now observed, then, a characterization of values in terms of reweighted arenas is provided.

Optimal values and Farey sequences. Recall that each value $\texttt{val}^\Gamma(v)$ is contained within the following set of rational numbers:

$$S_\Gamma = \left\{ \frac{N}{D} \;\middle|\; D \in [1, |V|], N \in [-DW, DW] \right\}.$$

Let us introduce some notation in order to handle S_Γ in a way that is suitable for our purposes. Firstly, we write every $\nu \in S_\Gamma$ as $\nu = i + F$, where $i = i_\nu = \lfloor \nu \rfloor$ is the integral and $F = F_\nu = \{\nu\} = \nu - i$ is the fractional part. Notice that $i \in [-W, W]$ and that F is a non-negative rational number having denominator at most $|V|$.

As a consequence, it is worthwhile to consider the *Farey sequence* $\mathcal{F}_n$ of order $n = |V|$. This is the increasing sequence of all irreducible fractions from the (rational) interval $[0,1]$ with denominators less than or equal to n. In the rest of this chapter, $\mathcal{F}_n$ denotes the following sorted set:

$$\mathcal{F}_n = \left\{ \frac{N}{D} \,\middle|\, 0 \leq N \leq D \leq n, \gcd(N,D) = 1 \right\}.$$

Farey sequences have numerous and interesting properties, in particular, many algorithms for generating the entire sequence $\mathcal{F}_n$ in time $O(n^2)$ are known in the literature [62], and these rely on *Stern-Brocot* trees and *mediant* properties. Notice that the above mentioned quadratic running time is optimal, as it is well-known that the sequence $\mathcal{F}_n$ has $s(n) = \frac{3n^2}{\pi^2} + O(n \ln n) = \Theta(n^2)$ terms.

Throughout the article, we shall assume that $F_0, \ldots, F_{s-1}$ is an increasing ordering of $\mathcal{F}_n$, so that $\mathcal{F}_n = \{F_j\}_{j=0}^{s-1}$ and $F_j < F_{j+1}$ for every j.

Also notice that $F_0 = 0$ and $F_{s-1} = 1$.

For example, $\mathcal{F}_5 = \{0, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, 1\}$.

At this point, S_Γ can be represented as follows:

$$S_\Gamma = [-W, W) + \mathcal{F}_{|V|} = \left\{ i + F_j \mid i \in [-W, W), j \in [0, s-1] \right\}.$$

The above representation of S_Γ will be convenient in a while.

Optimal values and reweightings. Two introductory lemmata are shown below, then, a characterization of optimal values in terms of reweightings is provided.

Lemma 6.5. *Let $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ be an MPG and let $q \in \mathbf{Q}$ be a rational number having denominator $D \in \mathbb{N}$.*

Then, $\texttt{val}^\Gamma(v) = \frac{1}{D} \texttt{val}^{\Gamma^{w+q}}(v) - q$ holds for every $v \in V$.

Proof. Let us consider the play $\texttt{outcome}^{\Gamma^{w+q}}(v, \sigma_0, \sigma_1) = v_0 v_1 \ldots v_n \ldots$ By the definition of $\texttt{val}^\Gamma(v)$, and by that of reweighting Γ^{w+q} ($= \Gamma^{D \cdot w + N}$), the following holds:

$$\begin{aligned}
\texttt{val}^{\Gamma^{w+q}}(v) &= \sup_{\sigma_0 \in \Sigma_0} \inf_{\sigma_1 \in \Sigma_1} \texttt{MP}_0(\texttt{outcome}^{\Gamma^{w+q}}(v, \sigma_0, \sigma_1)) \\
&= \sup_{\sigma_0 \in \Sigma_0} \inf_{\sigma_1 \in \Sigma_1} \liminf_{n \to \infty} \frac{1}{n} \textstyle\sum_{i=0}^{n-1} (D \cdot w(v_i, v_{i+1}) + N) \;\; (\text{if } q = N/D) \\
&= D \cdot \sup_{\sigma_0 \in \Sigma_0} \inf_{\sigma_1 \in \Sigma_1} \texttt{MP}_0(\texttt{outcome}^\Gamma(v, \sigma_0, \sigma_1)) + N \\
&= D \cdot \texttt{val}^\Gamma(v) + N.
\end{aligned}$$

Then, $\texttt{val}^\Gamma(v) = \frac{1}{D} \texttt{val}^{\Gamma^{w+q}}(v) - \frac{N}{D} = \frac{1}{D} \texttt{val}^{\Gamma^{w+q}}(v) - q$ holds for every $v \in V$.

□

Lemma 6.6. *Given an MPG $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$, let us consider the reweightings:*

$$\Gamma_{i,j} = \Gamma^{w-i-F_j}, \text{ for any } i \in [-W, W] \text{ and } j \in [0, s-1],$$

where $s = |\mathcal{F}_{|V|}|$ and F_j is the j-th term of the Farey sequence $\mathcal{F}_{|V|}$.
Then, the following propositions hold:

1. *For any $i \in [-W, W]$ and $j \in [0, s-1]$, we have:*

$$v \in \mathcal{W}_0(\Gamma_{i,j}) \text{ if and only if } \texttt{val}^{\Gamma}(v) \geq i + F_j;$$

2. *For any $i \in [-W, W]$ and $j \in [1, s-1]$, we have:*

$$v \in \mathcal{W}_1(\Gamma_{i,j}) \text{ if and only if } \texttt{val}^{\Gamma}(v) \leq i + F_{j-1}.$$

Proof.

1. Let us fix arbitrarily some $i \in [-W, W]$ and $j \in [0, s-1]$.
   Assume that $F_j = N_j / D_j$ for some $N_j, D_j \in \mathbb{N}$.
   Since
   $$\Gamma_{i,j} = (V, E, D_j(w-i) - N_j, \langle V_0, V_1 \rangle),$$
   then by Lemma 6.5 (applyed to $q = -i - F_j$) we have:
   $$\texttt{val}^{\Gamma}(v) = \frac{1}{D_j} \texttt{val}^{\Gamma_{i,j}}(v) + i + F_j.$$
   Recall that $v \in \mathcal{W}_0(\Gamma_{i,j})$ if and only if $\texttt{val}^{\Gamma_{i,j}}(v) \geq 0$.
   Hence, we have $v \in \mathcal{W}_0(\Gamma_{i,j})$ if and only if the following inequality holds:
   $$\begin{aligned} \texttt{val}^{\Gamma}(v) &= \frac{1}{D_j} \texttt{val}^{\Gamma_{i,j}}(v) + i + F_j \\ &\geq i + F_j. \end{aligned}$$
   This proves Item 1.

2. The argument is symmetric to that of Item 1, but with some further observations. Let us fix arbitrarily some $i \in [-W, W]$ and $j \in [1, s-1]$. Assume that $F_j = N_j / D_j$ for some $N_j, D_j \in \mathbb{N}$. Since $\Gamma_{i,j} = (V, E, D_j(w-i) - N_j, \langle V_0, V_1 \rangle)$, then by Lemma 6.5 we have
   $$\texttt{val}^{\Gamma}(v) = \frac{1}{D_j} \texttt{val}^{\Gamma_{i,j}}(v) + i + F_j.$$
   Recall that $v \in \mathcal{W}_1(\Gamma_{i,j})$ if and only if $\texttt{val}^{\Gamma_{i,j}}(v) < 0$.

Hence, we have $v \in \mathcal{W}_1(\Gamma_{i,j})$ if and only if the following inequality holds:

$$\begin{aligned}\texttt{val}^{\Gamma}(v) &= \frac{1}{D_j}\texttt{val}^{\Gamma_{i,j}}(v) + i + F_j \\ &< i + F_j.\end{aligned}$$

Now, recall from Section 6.2 that $\texttt{val}^{\Gamma}(v) \in S_{\Gamma}$, where

$$S_{\Gamma} = \{i + F_j \mid i \in [-W, W), j \in [0, s-1]\}.$$

By hypothesis we have:

$$j \geq 1 \text{ and } 0 \leq F_{j-1} < F_j,$$

thus, at this point, $v \in W_1(\Gamma_{i,j})$ if and only if $\texttt{val}^{\Gamma}(v) \leq i + F_{j-1}$.

This proves Item 2.

□

We are now in the position to provide a simple characterization of values in terms of reweightings.

Theorem 6.3. *Given an MPG $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$, let us consider the reweightings:*

$$\Gamma_{i,j} = \Gamma^{w-i-F_j}, \text{ for any } i \in [-W, W] \text{ and } j \in [1, s-1],$$

where $s = |\mathcal{F}_{|V|}|$ and F_j is the j-th term of the Farey sequence $\mathcal{F}_{|V|}$.

Then, the following holds:

$$\texttt{val}^{\Gamma}(v) = i + F_{j-1} \text{ if and only if } v \in \mathcal{W}_0(\Gamma_{i,j-1}) \cap \mathcal{W}_1(\Gamma_{i,j}).$$

Proof. Let us fix arbitrarily some $i \in [-W, W]$ and $j \in [1, s-1]$.

By Item 1 of Lemma 6.6, we have $v \in \mathcal{W}_0(\Gamma_{i,j-1})$ if and only if $\texttt{val}^{\Gamma}(v) \geq i + F_{j-1}$. Symmetrically, by Item 2 of Lemma 6.6, we have $v \in \mathcal{W}_1(\Gamma_{i,j})$ if and only if $\texttt{val}^{\Gamma}(v) \leq i + F_{j-1}$. Whence, by composition, $v \in \mathcal{W}_0(\Gamma_{i,j-1}) \cap \mathcal{W}_1(\Gamma_{i,j})$ if and only if $\texttt{val}^{\Gamma}(v) = i + F_{j-1}$. □

6.3.2 On Optimal Positional Strategies

The present subsections aims to provide a suitable description of optimal positional strategies in terms of reweighted arenas. An introductory lemma is shown next.

Lemma 6.7. *Let $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$ be an MPG, the following hold:*

1. *If $v \in V_0$, let $v' \in \texttt{post}(v)$. Then $\texttt{val}^{\Gamma}(v') \leq \texttt{val}^{\Gamma}(v)$ holds.*

2. *If $v \in V_1$, let $v' \in \texttt{post}(v)$. Then $\texttt{val}^{\Gamma}(v') \geq \texttt{val}^{\Gamma}(v)$ holds.*

3. *Given any $v \in V_0$, consider the reweighted EG $\Gamma_v = \Gamma^{w-\texttt{val}^{\Gamma}(v)}$.*

   Let $f_v : V \to \mathcal{C}_{\Gamma_v}$ be any SEPM of the EG Γ_v such that $v \in V_{f_v}$ (i.e., $f_v(v) \neq \top$). Let $v'_{f_v} \in V$ be any vertex such that $(v, v'_{f_v}) \in E$ is compatible with f_v in Γ_v.

   Then, $\texttt{val}^{\Gamma}(v'_{f_v}) = \texttt{val}^{\Gamma}(v)$.

Proof. 1. It is sufficient to construct a strategy $\sigma_0^v \in \Sigma_0^M$ securing to Player 0 a payoff at least $\texttt{val}^{\Gamma}(v')$ from v in the MPG Γ. Let $\sigma_0^{v'} \in \Sigma_0^M$ be a strategy securing payoff at least $\texttt{val}^{\Gamma}(v')$ from v' in Γ. Then, let σ_0^v be defined as follows:

$$\sigma_0^v(u) = \begin{cases} \sigma_0^{v'}(u) & \text{, if } u \in V_0 \setminus \{v\}; \\ \sigma_0^{v'}(v) & \text{, if } u = v \text{ and } v \textit{ is} \text{ reachable from } v' \text{ in } G^{\Gamma}_{\sigma_0^{v'}}; \\ v' & \text{, if } u = v \text{ and } v \textit{ is not} \text{ reachable from } v' \text{ in } G^{\Gamma}_{\sigma_0^{v'}}. \end{cases}$$

We argue that σ_0^v secures payoff at least $\texttt{val}^{\Gamma}(v')$ from v in Γ. First notice that, by Lemma 6.1 (applied to v'), all cycles C that are reachable from v' in Γ satisfy:

$$\frac{w(C)}{|C|} \geq \texttt{val}^{\Gamma}(v').$$

The fact is that any cycle reachable from v in $G^{\Gamma}_{\sigma_0^v}$ is also reachable from v' in $G^{\Gamma}_{\sigma_0^{v'}}$ (by definition of σ_0^v), therefore, the same inequality holds for all cycles reachable from v. At this point, the thesis follows again by Lemma 6.1 (applied to v, in the inverse direction). This proves Item 1.

2. The proof of Item 2 is symmetric to that of Item 1.

3. Firstly, notice that $\texttt{val}^{\Gamma}(v'_{f_v}) \leq \texttt{val}^{\Gamma}(v)$ holds by Item 1. To conclude the proof it is sufficient to show $\texttt{val}^{\Gamma}(v'_{f_v}) \geq \texttt{val}^{\Gamma}(v)$. Recall that $(v, v'_{f_v}) \in E$ is compatible with f_v in Γ_v by hypothesis, that is:

$$f_v(v) \succeq f_v(v'_{f_v}) \ominus \big(w(v, v'_{f_v}) - \texttt{val}^{\Gamma}(v)\big).$$

This, together with the fact that $v \in V_{f_v}$ (i.e., $f_v(v) \neq \top$) also holds by hypothesis, implies that $v'_{f_v} \in V_f$ (i.e., $f_v(v'_{f_v}) \neq \top$). Thus, by Item 1 of Lemma 6.3, v'_{f_v} is a winning starting position of Player 0 in the EG Γ_v. Whence, by Lemma 6.2, it holds that $\texttt{val}^{\Gamma}(v'_{f_v}) \geq \texttt{val}^{\Gamma}(v)$. This proves Item 3.

□

We are now in position to provide a sufficient condition, for a positional strategy to be optimal, which is expressed in terms of reweighted EGs and their SEPMs.

Theorem 6.4. *Let $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$ be an MPG.For each $v \in V$, consider the reweighted EG $\Gamma_v = \Gamma^{w - \mathtt{val}^{\Gamma}(v)}$. Let $f_v : V \to \mathcal{C}_{\Gamma_v}$ be any SEPM of Γ_v such that $v \in V_{f_v}$ (i.e., $f_v(v) \neq \top$). Moreover, assume: $f_{v_1} = f_{v_2}$ whenever $\mathtt{val}^{\Gamma}(v_1) = \mathtt{val}^{\Gamma}(v_2)$.*

When $v \in V_0$, let $v'_{f_v} \in V$ be any vertex out of v such that $(v, v'_{f_v}) \in E$ is compatible with f_v in the EG Γ_v, and consider the positional strategy $\sigma_0^* \in \Sigma_0^M$ defined as follows:

$$\sigma_0^*(v) = v'_{f_v}, \textit{for every } v \in V_0.$$

Then, σ_0^* is an optimal positional strategy for Player 0 in the MPG Γ.

Proof. Let us consider the projection graph $G^{\Gamma}_{\sigma_0^*} = (V, E_{\sigma_0^*}, w)$. Let $v \in V$ be any vertex. In order to prove that σ_0^* is optimal, it is sufficient (by Lemma 6.1) to show that every cycle C that is reachable from v in $G^{\Gamma}_{\sigma_0^*}$ satisfies $\frac{w(C)}{|C|} \geq \mathtt{val}^{\Gamma}(v)$.

- *Preliminaries.* Let $v \in V$ and let C be any cycle of length $|C| \geq 1$ that is reachable from v in $G^{\Gamma}_{\sigma_0^*}$. Then, there exists a path ρ of length $|\rho| \geq 1$ in $G^{\Gamma}_{\sigma_0^*}$ and such that: if $|\rho| = 1$, then $\rho = \rho_0\rho_1 = vv$; otherwise, if $|\rho| > 1$, then:
$$\rho = \rho_0 \dots \rho_{|\rho|} = vv_1v_2 \dots v_k u_1 u_2 \dots u_{|C|} u_1,$$
where $vv_1 \dots v_k$ is a simple path, for some $k \geq 0$ and $u_1 \dots u_{|C|} u_1 = C$.

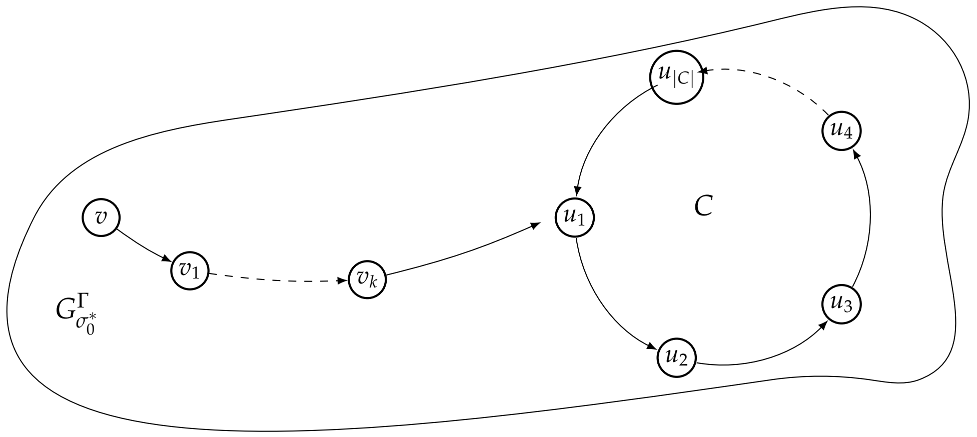


Figure 6.2: A cycle C that is reachable from v through $v_1 \cdots v_k$ in $G^{\Gamma}_{\sigma_0^*}$.

- *Fact 1.* It holds $\mathtt{val}^{\Gamma}(\rho_i) \leq \mathtt{val}^{\Gamma}(\rho_{i+1})$ for every $i \in [0, |\rho|)$.

  Proof of Fact 1. If $\rho_i \in V_0$ then $\mathtt{val}^{\Gamma}(\rho_i) = \mathtt{val}^{\Gamma}(\rho_{i+1})$ by Item 3 of Lemma 6.7; otherwise, if $\rho_i \in V_1$, then $\mathtt{val}^{\Gamma}(\rho_i) \leq \mathtt{val}^{\Gamma}(\rho_{i+1})$ by Item 2 of Lemma 6.7.

This proves Fact 1. In particular, notice that $\texttt{val}^{\Gamma}(v) \leq \texttt{val}^{\Gamma}(u_1)$ when $|\rho| > 1$. □

- *Fact 2.* Assume $C = u_1 \dots u_{|C|} u_1$, then:

$$\texttt{val}^{\Gamma}(u_i) = \texttt{val}^{\Gamma}(u_1) \text{ for every } i \in [0, |C|].$$

Proof of Fact 2. By Fact 1, $\texttt{val}^{\Gamma}(u_{i-1}) \leq \texttt{val}^{\Gamma}(u_i)$ for every $i \in [2, |C|]$, as well as $\texttt{val}^{\Gamma}(u_{|C|}) \leq \texttt{val}^{\Gamma}(u_1)$. Then, the following chain of inequalities holds:

$$\texttt{val}^{\Gamma}(u_1) \leq \texttt{val}^{\Gamma}(u_2) \leq \dots \leq \texttt{val}^{\Gamma}(u_{|C|}) \leq \texttt{val}^{\Gamma}(u_1).$$

Since the first and the last value of the chain are actually the same, i.e., $\texttt{val}^{\Gamma}(u_1)$, then, all these inequalities are indeed equalities. This proves Fact 2. □

- *Fact 3.* The following holds for every $i \in [0, |\rho|)$:

$$f_{\rho_i}(\rho_i), f_{\rho_i}(\rho_{i+1}) \neq \top \text{ and } f_{\rho_i}(\rho_i) \geq f_{\rho_i}(\rho_{i+1}) - w(\rho_i, \rho_{i+1}) + \texttt{val}^{\Gamma}(\rho_i).$$

Proof of Fact 3. Firstly, we argue that any arc $(\rho_i, \rho_{i+1}) \in E$ is compatible with f_{ρ_i} in Γ_{ρ_i}. Indeed, if $\rho_i \in V_0$, then (ρ_i, ρ_{i+1}) is compatible with f_{ρ_i} in Γ_{ρ_i} because $\rho_{i+1} = \sigma_0^*(\rho_i)$ by hypothesis; otherwise, if $\rho_i \in V_1$, then (ρ_i, x) is compatible with f_{ρ_i} in Γ_{ρ_i} for *every* $x \in \texttt{post}(\rho_i)$, in particular for $x = \rho_{i+1}$, by definition of SEPM.

At this point, since (ρ_i, ρ_{i+1}) is compatible with f_{ρ_i} in Γ_{ρ_i}, then:

$$f_{\rho_i}(\rho_i) \succeq f_{\rho_i}(\rho_{i+1}) \ominus \big(w(\rho_i, \rho_{i+1}) - \texttt{val}^{\Gamma}(\rho_i)\big).$$

Now, recall that $\rho_i \in V_{f_{\rho_i}}$ (i.e., $f_{\rho_i}(\rho_i) \neq \top$) holds for every ρ_i by hypothesis. Since $f_{\rho_i}(\rho_i) \neq \top$ and the above inequality holds, then we have $f_{\rho_i}(\rho_{i+1}) \neq \top$. Thus, we can safely write:

$$f_{\rho_i}(\rho_i) \geq f_{\rho_i}(\rho_{i+1}) - w(\rho_i, \rho_{i+1}) + \texttt{val}^{\Gamma}(\rho_i).$$

This proves Fact 3. □

- *Fact 4.* Assume that the cycle $C = u_1 \dots u_{|C|} u_1$ is such that:

$$\texttt{val}^{\Gamma}(u_i) = \texttt{val}^{\Gamma}(u_1) \geq \texttt{val}^{\Gamma}(v), \text{ for every } i \in [1, |C|].$$

Then, provided that $u_{|C|+1} = u_1$, the following holds for every $i \in [1, |C|]$:

$f_{u_1}(u_1), f_{u_{i+1}}(u_{i+1}) \neq \top$ and

$$f_{u_1}(u_1) \geq f_{u_{i+1}}(u_{i+1}) - \sum_{j=1}^{i} w(u_j, u_{j+1}) + i \cdot \mathtt{val}^{\Gamma}(v).$$

Proof of Fact 4. Firstly, notice that $f_{u_1}(u_1), f_{u_{i+1}}(u_{i+1}) \neq \top$ holds by hypothesis.

The proof proceeds by induction on $i \in [1, |C|]$.

– *Base Case.* Assume that $|C| = 1$, so that $C = u_1 u_1$. Then $f_{u_1}(u_1) \geq f_{u_1}(u_1) - w(u_1, u_1) + \mathtt{val}^{\Gamma}(u_1)$ follows by Fact 3. Since $\mathtt{val}^{\Gamma}(u_1) \geq \mathtt{val}^{\Gamma}(v)$ by hypothesis, then the thesis follows.

– *Inductive Step.* Assume by induction hypothesis that the following holds:

$$f_{u_1}(u_1) \geq f_{u_i}(u_i) - \sum_{j=1}^{i-1} w(u_j, u_{j+1}) + (i-1) \cdot \mathtt{val}^{\Gamma}(v).$$

By Fact 3, we have:

$$f_{u_i}(u_i) \geq f_{u_i}(u_{i+1}) - w(u_i, u_{i+1}) + \mathtt{val}^{\Gamma}(u_i).$$

Since $\mathtt{val}^{\Gamma}(u_{i+1}) = \mathtt{val}^{\Gamma}(u_i)$ holds by hypothesis, then we have $f_{u_{i+1}} = f_{u_i}$. Recall that $\mathtt{val}^{\Gamma}(u_i) \geq \mathtt{val}^{\Gamma}(v)$ also holds by hypothesis.

Thus, we obtain the following:

$$f_{u_1}(u_1) \geq f_{u_{i+1}}(u_{i+1}) - \sum_{j=1}^{i} w(u_j, u_{j+1}) + i \cdot \mathtt{val}^{\Gamma}(v).$$

This proves Fact 4.

□

- We are now in position to show that every cycle C that is reachable from v in $G^{\Gamma}_{\sigma^*_0}$ satisfies $w(C)/|C| \geq \mathtt{val}^{\Gamma}(v)$. By Fact 1 and Fact 2, we have $\mathtt{val}^{\Gamma}(v) \leq \mathtt{val}^{\Gamma}(u_1) = \mathtt{val}^{\Gamma}(u_i)$ for every $i \in [1, |C|]$. At this point, we apply Fact 4. Consider the specialization of Fact 4 when $i = |C|$ and also recall that $u_{|C|+1} = u_1$. Then, we have the following:

$$f_{u_1}(u_1) \geq f_{u_1}(u_1) - \sum_{j=1}^{|C|} w(u_j, u_{j+1}) + |C| \cdot \mathtt{val}^{\Gamma}(v).$$

As a consequence, the following lower bound holds on the average weight of C:

$$\frac{w(C)}{|C|} = \frac{1}{|C|}\sum_{j=1}^{|C|} w(u_j, u_{j+1}) \geq \mathtt{val}^{\Gamma}(v),$$

which concludes the proof.

□

Remark 6.1. *Notice that Theorem 6.4 holds, in particular, when f_v is the least SEPM f^*_v of the reweighted EG Γ_v. This follows because $v \in V_{f^*_v}$ always holds for the least SEPM f^*_v of the EG Γ_v, as shown next: by Lemma 6.2 and by definition of Γ_v, then v is a winning starting position for Player 0 in the EG Γ_v (for some initial credit); now, since f^*_v is the least SEPM of the EG Γ_v, then $v \in V_{f^*_v}$ follows by Item 2 of Lemma 6.3.*

6.4 A $\Theta(|V|^2|E|\,W)$ time Algorithm for solving the Value Problem and Optimal Strategy Synthesis in MPGs

This section offers a deterministic algorithm for solving the Value Problem and Optimal Strategy Synthesis of MPGs within $\Theta(|V|^2|E|\,W)$ time and $\Theta(|V|)$ space, on any input MPG $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$.

Let us now recall some notation in order to describe the algorithm in a suitable way. Given an MPG $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$, consider again the following reweightings:

$$\Gamma_{i,j} = \Gamma^{w-i-F_j}, \text{ for any } i \in [-W, W] \text{ and } j \in [0, s-1],$$

where $s = |\mathcal{F}_{|V|}|$ and F_j is the j-th term of $\mathcal{F}_{|V|}$.

Assuming $F_j = N_j/D_j$ for some $N_j, D_j \in \mathbb{N}$, we focus on the following weights:

$$\begin{aligned} w_{i,j} =& w - i - F_j = w - i - \frac{N_j}{D_j}; \\ w'_{i,j} =& D_j\, w_{i,j} = D_j\,(w - i) - N_j. \end{aligned}$$

Recall that $\Gamma_{i,j}$ is defined as $\Gamma_{i,j} \triangleq \Gamma^{w'_{i,j}}$, which is an arena having integer weights. Also notice that, since $F_0 < \ldots < F_{s-1}$ is monotone increasing, then the corresponding weight functions $w_{i,j}$ can be ordered in a natural way, i.e., $w_{-W,1} > w_{-W,2} > \ldots > w_{W-1,s-1} > \ldots > w_{W,s-1}$. In the rest of this section, we denote by $f^*_{w'_{i,j}} : V \to \mathcal{C}_{\Gamma_{i,j}}$ the least SEPM of the reweighted EG $\Gamma_{i,j}$. Moreover, the function $f^*_{i,j} : V \to \mathbf{Q}$, defined as $f^*_{i,j}(v) \triangleq \frac{1}{D_j} f^*_{w'_{i,j}}(v)$ for every $v \in V$, is called the *rational scaling* of $f^*_{w'_{i,j}}$.

6.4.1 Description of the Algorithm

In this section we shall describe a procedure whose pseudo-code is given below in Algorithm 16. It takes as input an arena $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$, and it aims to return a tuple $(\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma_0^*)$ such that: $\mathcal{W}_0$ and $\mathcal{W}_1$ are the winning regions of Player 0 and Player 1 in the MPG Γ (respectively), $\nu : V \to S_\Gamma$ is a map sending each starting position $v \in V$ to its optimal value, i.e., $\nu(v) = \texttt{val}^\Gamma(v)$, and finally, $\sigma_0^* : V_0 \to V$ is an optimal positional strategy for Player 0 in the MPG Γ.

The intuition underlying Algorithm 16 is that of considering the following sequence of weights:

$$w_{-W,1} > w_{-W,2} > \ldots > w_{-W,s-1} > w_{-W+1,1} > w_{-W+1,2} > \ldots > w_{W-1,s-1} > \ldots > w_{W,s-1}$$

where the key idea is that to rely on Theorem 6.3 at each one of these steps, testing whether a *transition of winning regions* has occurred. Stated otherwise,

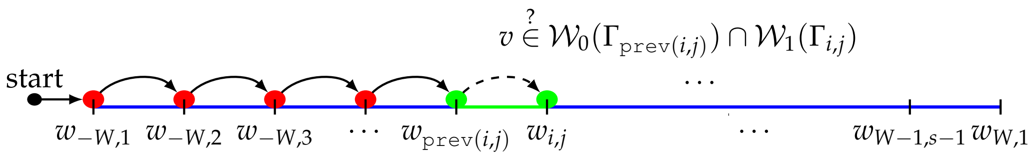


Figure 6.3: An illustration of Algorithm 16.

the idea is to check, for each vertex $v \in V$, whether v is winning for Player 1 with respect to the current weight $w_{i,j}$, meanwhile recalling whether v was winning for Player 0 with respect to the immediately preceding element $w_{\texttt{prev}(i,j)}$ in the weight sequence above.

If such a transition occurs, say for some $\hat{v} \in \mathcal{W}_0(\Gamma_{\texttt{prev}(i,j)}) \cap \mathcal{W}_1(\Gamma_{i,j})$, then one can easily compute $\texttt{val}^\Gamma(\hat{v})$ by relying on Theorem 6.3; Also, at that point, it is easy to compute an optimal positional strategy, provided that $\hat{v} \in V_0$, by relying on Theorem 6.4 and Remark 6.1 in that case.

Each one of these phases, in which one looks at transitions of winning regions, is named *Scan Phase*. A graphical intuition of Algorithm 16 is given in Fig. 7.2.

An in-depth description of the algorithm and of its pseudo-code now follows.

- **Initialization Phase.** To start with, the algorithm performs an initialization phase. At line 1, Algorithm 16 initializes the output variables $\mathcal{W}_0$ and $\mathcal{W}_1$ to be empty sets. Notice that, within the pseudo-code, the variables $\mathcal{W}_0$ and $\mathcal{W}_1$ represent the winning regions of Player 0 and Player 1, respectively; also, the variable ν represents the optimal values of the input MPG Γ, and σ_0^* represents an optimal positional strategy for Player 0 in the input MPG Γ. Secondly, at line 2, an array variable $f : V \to \mathcal{C}_\Gamma$

Algorithm 14: Solving Value Problem and Strategy Synthesis in MPGs.

```
Procedure solve_MPG($\Gamma$)
    input : an MPG $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$.
    output: a tuple $(\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma_0^*)$ such that: $\mathcal{W}_0$ and $\mathcal{W}_1$ are the winning
            regions of Player 0 and Player 1 (respectively) in the MPG $\Gamma$;
            $\nu : V \to S_\Gamma$ is a map sending each starting position $v \in V$ to its
            corresponding optimal value, i.e., $\nu(v) = \texttt{val}^\Gamma(v)$; and $\sigma_0^* : V_0 \to V$
            is an optimal positional strategy for Player 0 in the MPG $\Gamma$.
    // Init Phase
    $\mathcal{W}_0 \leftarrow \emptyset$; $\mathcal{W}_1 \leftarrow \emptyset$;
    $f(v) \leftarrow 0, \forall\, v \in V$;
    $W \leftarrow \max_{e \in E} |w_e|$; $w' \leftarrow w + W$; $D \leftarrow 1$;
    $s \leftarrow$ compute the size $|\mathcal{F}_{|V|}|$ of $\mathcal{F}_{|V|}$; // with the algorithm of [97]
    // Scan Phases
    for $i = -W$ to $W$ do
        $F \leftarrow 0$;
        for $j = 1$ to $s - 1$ do
            prev_$f \leftarrow f$;
            prev_$w \leftarrow \frac{1}{D} w'$;
            prev_$F \leftarrow F$;
            $F \leftarrow$ generate the $j$-th term of $\mathcal{F}_{|V|}$; // with the algorithm of [97]
            $N \leftarrow$ numerator of $F$;
            $D \leftarrow$ denominator of $F$;
            $w' \leftarrow D(w - i) - N$;
            $f \leftarrow \frac{1}{D}\texttt{Value-Iteration}(\Gamma^{w'}, \lceil D\text{prev\_}f \rceil)$;
            for $v \in V$ do
                if $\textit{prev\_}f(v) \neq \top$ and $f(v) = \top$ then
                    $\nu(v) \leftarrow i + \texttt{prev\_}F$; // set optimal value $\nu$
                    if $\nu(v) \geq 0$ then
                        $\mathcal{W}_0 \leftarrow \mathcal{W}_0 \cup \{v\}$; // $v$ is winning for Player 0
                    else
                        $\mathcal{W}_1 \leftarrow \mathcal{W}_1 \cup \{v\}$; // $v$ is winning for Player 1
                    if $v \in V_0$ then
                        for $u \in \texttt{post}(v)$ do
                            if $\textit{prev\_}f(v) \succeq \textit{prev\_}f(u) \ominus \textit{prev\_}w(v,u)$ then
                                $\sigma_0^*(v) \leftarrow u$; break;
    return $(\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma_0^*)$
```

Algorithm 14: The MPG algorithm.

is initialized to $f(v) = 0$ for every $v \in V$; throughout the computation, the variable f represents a SEPM. Next, at line 3, the greatest absolute weight W is assigned as $W = \max_{e \in E} |w_e|$, an auxiliary weight function w' is initialized as $w' = w + W$, and a "denominator" variable is initialized as $D = 1$. Concluding the initialization phase, at line 4 the size (i.e., the total number of terms) of $\mathcal{F}_{|V|}$ is computed and assigned to the variable s. This size can be computed very efficiently with the algorithm devised by Pawlewicz and Pătraşcu [97].

- **Scan Phases.** After initialization, the procedure performs multiple *Scan Phases*. Each one of these is *indexed* by a pair of integers (i,j), where $i \in [-W,W]$ (at line 6) and $j \in [1,s-1]$ (at line 7). Thus, the index i goes from $-W$ to W, and for each i, the index j goes from 1 to $s-1$.

  At each step, we say that the algorithm goes through the (i,j)-th scan phase. For each scan phase, we also need to consider the *previous* scan phase, so that the *previous index* $\texttt{prev}(i,j)$ shall be defined as follows: the predecessor of the first index is $\texttt{prev}(-W,1) \triangleq (-W,0)$; if $j > 1$, then $\texttt{prev}(i,j) \triangleq (i,j-1)$; finally, if $j = 1$ and $i > -W$, then $\texttt{prev}(i,j) \triangleq (i-1,s-1)$.

  At the (i,j)-th scan phase, the algorithm considers the rational number $z_{i,j} \in S_\Gamma$ defined as:
$$z_{i,j} \triangleq i + F[j],$$
  where $F[j] = N_j/D_j$ is the j-th term of $\mathcal{F}_{|V|}$. For each j, $F[j]$ can be computed very efficiently, on the fly, with the algorithm of Pawlewicz and Pătraşcu [97]. Notice that, since $F[0] < \ldots < F[s-1]$ is monotonically increasing, then the values $z_{i,j}$ are scanned in increasing order as well. At this point, the procedure aims to compute the rational scaling $f^*_{i,j}$ of the least SEPM $f^*_{w'_{i,j}}$, i.e.,
$$f \triangleq f^*_{i,j} = \frac{1}{D_j} f^*_{w'_{i,j}}.$$
  This computation is really at the heart of the algorithm and it goes from line 8 to line 13. To start with, at line 8 and line 9, the previous rational scaling $f^*_{\texttt{prev}(i,j)}$ and the previous weight function $w_{\texttt{prev}(i,j)}$ (i.e., those considered during the previous scan phase) are saved into the auxiliary variables $\texttt{prev\_}f$ and $\texttt{prev\_}w$.

  Remark. Since the values $z_{i,j}$ are scanned in increasing order of magnitude, then $\texttt{prev\_}f = f^*_{\texttt{prev}(i,j)}$ bounds from below $f^*_{i,j}$. That is, it holds for every $v \in V$ that:
$$\texttt{prev\_}f(v) = f^*_{\texttt{prev}(i,j)}(v) \preceq f^*_{i,j}.$$
  The underlying intuition, at this point, is that of computing the energy levels of $f = f^*_{i,j}$ firstly by initializing them to the energy levels of the

previous scan phase, i.e., to $\texttt{prev\_}f = f^*_{\texttt{prev}(i,j)}$, and then to update them monotonically upwards by executing the Value-Iteration algorithm for EGs.

Further details of this pivotal step now follow. Firstly, since the Value-Iteration has been designed to work with integer numerical weights only [14], then the weights $w_{i,j} = w - z_{i,j}$ have to be scaled from $\mathbf{Q}$ to $\mathbb{Z}$: this is performed in the standard way, from line 10 to line 13, by considering the numerator N_j and the denominator D_j of $F[j]$, and then by setting:

$$w'_{i,j}(e) \triangleq D_j\big(w(e) - i\big) - N_j, \text{ for every } e \in E.$$

The initial energy levels are also scaled up from $\mathbf{Q}$ to $\mathbb{Z}$ by considering the values: $\lceil D_j \texttt{prev\_}f(v)\rceil$, for every $v \in V$ (line 13). At this point the least SEPM of $\Gamma^{w'_{i,j}}$ is computed, at line 13, by invoking:

$$\texttt{Value-Iteration}(\Gamma^{w'_{i,j}}, \lceil D_j \texttt{prev\_}f\rceil),$$

that is, by executing on input $\Gamma^{w'_{i,j}}$ the Value-Iteration with initial energy levels given by: $\lceil D_j \texttt{prev\_}f(v)\rceil$ for every $v \in V$. Soon after that, the energy levels have to be scaled back from $\mathbb{Z}$ to $\mathbf{Q}$, so that, in summary, at line 13 they becomes:

$$f = f^*_{i,j} = \frac{1}{D_j}\texttt{Value-Iteration}(\Gamma^{w'_{i,j}}, \lceil D_j \texttt{prev\_}f\rceil).$$

The correctness of lines 12-13 will be proved in Lemma 6.8.

Here, let us provide a sketch of the argument:

1. Since $F_0 < \ldots < F_{s-1}$ is monotone increasing, then the sequence $\{w'_{i,j}\}_{(i,j)}$ is monotone decreasing, i.e., for every i, j and $e \in E$, $w'_{\texttt{prev}(i,j)}(e) > w'_{i,j}(e)$. Whence, the sequence of rational scalings $\{f^*_{i,j}\}_{i,j}$ is monotone increasing, i.e., $f^*_{i,j} \preceq f^*_{\texttt{prev}(i,j)}$ holds at the (i,j)-th step. The proof is in Lemma 6.8.
2. At the (i,j)-th iteration of line 8, it holds that $\texttt{prev\_}f = f^*_{\texttt{prev}(i,j)}$.
   This invariant property is also proved as part of Lemma 6.8.
3. Since $\texttt{prev\_}f = f^*_{\texttt{prev}(i,j)}$, then $\texttt{prev\_}f \preceq f^*_{i,j}$.
   Thus, one can prove that $D_j \texttt{prev\_f} \preceq f^*_{w'_{i,j}}$.
4. Since $w'_{i,j}(e) \in \mathbb{Z}$ for every $e \in E$, then $f^*_{w'_{i,j}}(v) \in \mathbb{Z}$ for every $v \in V$, so that
   $$\lceil D_j \texttt{prev\_f}(v)\rceil \preceq f^*_{w'_{i,j}}(v)$$
   holds for every $v \in V$ as well.

5. This implies that it is correct to execute the Value-Iteration, on input $\Gamma^{w'_{i,j}}$, with initial energy levels given by: $\lceil D_j \texttt{prev\_}f(v) \rceil$ for every $v \in V$.

Back to us, once $f = f^*_{i,j}$ has been determined, then for each $v \in V$ the condition:

$$v \overset{?}{\in} \mathcal{W}_0(\Gamma_{\texttt{prev}(i,j)}) \cap \mathcal{W}_1(\Gamma_{i,j}),$$

is checked at line 17: it is not difficult to show that, for this, it is sufficient to test whether both $\texttt{prev\_f}(v) \neq \top$ and $f(v) = \top$ hold on v (it follows by Lemma 6.8).

If $v \in \mathcal{W}_0(\Gamma_{\texttt{prev}(i,j)}) \cap \mathcal{W}_1(\Gamma_{i,j})$ holds, then the algorithm relies on Theorem 6.3 in order to assign the optimal value as follows: $\nu(v) \triangleq \texttt{val}^{\Gamma}(v) = z_{\texttt{prev}(i,j)}$ (line 18). If $\nu(v) \geq 0$, then v is added to the winning region $\mathcal{W}_0$ at line 20. Otherwise, $\nu(v) < 0$ and v is added to $\mathcal{W}_1$ at line 22.

To conclude, from line 23 to line 27, the algorithm proceeds as follows: if $v \in V_0$, then it computes an optimal positional strategy $\sigma^*_0(v)$ for Player 0 in Γ: this is done by testing for each $u \in \texttt{post}(v)$ whether $(v, u) \in E$ is an arc compatible with $\texttt{prev\_}f$ in $\Gamma_{\texttt{prev}(i,j)}$; namely, whether the following holds for some $u \in \texttt{post}(v)$:

$$\text{prev\_}f(v) \overset{?}{\succeq} \text{prev\_}f(u) \ominus \text{prev\_}w(v, u).$$

If $(v, u) \in E$ is found to be compatible with $\texttt{prev\_}f$ at that point, then $\sigma^*_0(v) \triangleq u$ gets assigned and the arc (v, u) becomes part of the optimal positional strategy returned to output. Indeed, the correctness of such an assignment relies on Theorem 6.4 and Remark 6.1.

This concludes the description of the scan phases and also that of Algorithm 16.

6.4.2 Proof of Correctness

Now we formally prove the correctness of Algorithm 16. The following lemma shows some basic invariants that are maintained throughout the computation.

Lemma 6.8. *Algorithm 16 keeps the following invariants throughout the computation:*

1. *For every $i \in [-W, W]$ and every $j \in [1, s-1]$, it holds that:*

$$f^*_{\texttt{prev}(i,j)}(v) \preceq f^*_{i,j}(v), \textit{ for every } v \in V;$$

2. *At the (i, j)-th iteration of line 8, it holds that: $\texttt{prev\_}f = f^*_{\texttt{prev}(i,j)}$;*

3. *At the (i, j)-th iteration of line 8, it holds that: $\lceil D_j \texttt{prev\_}f \rceil \preceq f^*_{w'_{i,j}}$;*

4. *At the (i,j)-th iteration of line 13, it holds that:*

$$\frac{1}{D_j}\texttt{Value-Iteration}(\Gamma^{w'_{i,j}}, \lceil D_j prev\_f \rceil) = f^*_{i,j}.$$

Proof of Item 1. Recall that $w_{i,j} \triangleq w - i - F_j$. Since $F_0 < \ldots < F_{s-1}$ is monotone increasing, then: $w_{i,j}(e) < w_{\texttt{prev}(i,j)}(e)$ holds for every $e \in E$.

In order to prove the thesis, consider the following function:

$$g : V \to \mathbf{Q} \cup \{\top\} : v \mapsto \min\left(f^*_{\texttt{prev}(i,j)}(v), f^*_{i,j}(v)\right).$$

We show that $D_{\texttt{prev}(i,j)}\, g$ is a SEPM of $\Gamma^{w'_{\texttt{prev}(i,j)}}$. There are four cases, according to whether $v \in V_0$ or $v \in V_1$, and $g(v) = f^*_{\texttt{prev}(i,j)}(v)$ or $g(v) = f^*_{i,j}(v)$.

Case: $v \in V_0$. Then, the following holds for *some* $u \in \texttt{post}(v)$:

SubCase: $g(v) = f^*_{prev(i,j)}(v)$:

$$\begin{aligned}
D_{\texttt{prev}(i,j)}\, g(v) &= D_{\texttt{prev}(i,j)}\, f^*_{\texttt{prev}(i,j)}(v) && [\text{by } g(v) = f^*_{\texttt{prev}(i,j)}(v)] \\
&= f^*_{w'_{\texttt{prev}(i,j)}}(v) && [\text{by } D_{\texttt{prev}(i,j)} f^*_{\texttt{prev}(i,j)} = f^*_{w'_{\texttt{prev}(i,j)}}] \\
&\succeq f^*_{w'_{\texttt{prev}(i,j)}}(v) \ominus w'_{\texttt{prev}(i,j)}(v,u) && [f^*_{w'_{\texttt{prev}(i,j)}} \text{ is SEPM of } \Gamma^{w'_{\texttt{prev}(i,j)}}] \\
&= D_{\texttt{prev}(i,j)}\, f^*_{\texttt{prev}(i,j)}(u) \ominus w'_{\texttt{prev}(i,j)}(v,u) && [\text{by } f^*_{w'_{\texttt{prev}(i,j)}} = D_{\texttt{prev}(i,j)} f^*_{\texttt{prev}(i,j)}] \\
&\succeq D_{\texttt{prev}(i,j)}\, g(u) \ominus w'_{\texttt{prev}(i,j)}(v,u) && [\text{by definition of } g(u)]
\end{aligned}$$

SubCase: $g(v) = f^*_{i,j}(v)$:

$$\begin{aligned}
D_{\texttt{prev}(i,j)}\, g(v) &= D_{\texttt{prev}(i,j)}\, f^*_{i,j}(v) && [\text{by } g(v) = f^*_{i,j}(v)] \\
&= \frac{D_{\texttt{prev}(i,j)}}{D_{i,j}} f^*_{w'_{i,j}}(v) && [\text{by } f^*_{i,j} = f^*_{w'_{i,j}}/D_{i,j}] \\
&\succeq \frac{D_{\texttt{prev}(i,j)}}{D_{i,j}} f^*_{w'_{i,j}}(u) \ominus \frac{D_{\texttt{prev}(i,j)}}{D_{i,j}} w'_{i,j}(v,u) && [f^*_{w'_{i,j}} \text{ is SEPM of } \Gamma^{w'_{i,j}}] \\
&= D_{\texttt{prev}(i,j)} f^*_{i,j}(u) \ominus \frac{D_{\texttt{prev}(i,j)}}{D_{i,j}} w'_{i,j}(v,u) && [\text{by } f^*_{i,j} = f^*_{w'_{i,j}}/D_{i,j}] \\
&= D_{\texttt{prev}(i,j)} f^*_{i,j}(u) \ominus D_{\texttt{prev}(i,j)} w_{i,j}(v,u) && [\text{by } w_{i,j}(v,u) = w'_{i,j}(v,u)/D_{i,j}] \\
&\succeq D_{\texttt{prev}(i,j)} f^*_{i,j}(u) \ominus D_{\texttt{prev}(i,j)}\, w_{\texttt{prev}(i,j)}(v,u) && [\text{by } w_{i,j} < w_{\texttt{prev}(i,j)}] \\
&= D_{\texttt{prev}(i,j)} f^*_{i,j}(u) \ominus w'_{\texttt{prev}(i,j)}(v,u) && [\text{by } D_{\texttt{prev}(i,j)} w_{\texttt{prev}(i,j)} = w'_{\texttt{prev}(i,j)}] \\
&\succeq D_{\texttt{prev}(i,j)} g(u) \ominus w'_{\texttt{prev}(i,j)}(v,u) && [\text{by definition of } g(u)]
\end{aligned}$$

This means that (v,u) is an arc compatible with $D_{\texttt{prev}(i,j)} g$ in $\Gamma^{w'_{\texttt{prev}(i,j)}}$.

Case: $v \in V_1$. The same argument shows that $(v,u) \in E$ is compatible with $D_{\texttt{prev}(i,j)} g$ in $\Gamma^{w'_{\texttt{prev}(i,j)}}$, but it holds for *all* $u \in \texttt{post}(v)$ in this case.

This proves that $D_{\texttt{prev}(i,j)}\, g$ is a SEPM of $\Gamma^{w'_{\texttt{prev}(i,j)}}$.

Now, since $f^*_{w'_{\texttt{prev}(i,j)}}$ is the *least* SEPM of $\Gamma^{w'_{\texttt{prev}(i,j)}}$, then:

$$f^*_{w'_{\texttt{prev}(i,j)}}(v) \preceq D_{\texttt{prev}(i,j)}\, g(v), \text{ for every } v \in V.$$

Since $f^*_{w'_{\texttt{prev}(i,j)}} = D_{\texttt{prev}(i,j)}\, f^*_{\texttt{prev}(i,j)}$ and $g = \min(f^*_{\texttt{prev}(i,j)}, f^*_{i,j})$, then:

$$D_{\texttt{prev}(i,j)}\, f^*_{\texttt{prev}(i,j)} \preceq D_{\texttt{prev}(i,j)} \min(f^*_{\texttt{prev}(i,j)}, f^*_{i,j}).$$

Whence $f^*_{\texttt{prev}(i,j)} = \min(f^*_{\texttt{prev}(i,j)}, f^*_{i,j})$.

This proves that $f^*_{\texttt{prev}(i,j)}(v) \preceq f^*_{i,j}(v)$ holds for every $v \in V$. □

Next, we prove that:

Fact 1. If Item 2 holds at the (i,j)-th scan phase, then both Item 3 and Item 4 hold at the (i,j)-th scan phase as well.

Proof of Fact 1. Assume that Item 2 holds. Let us prove Item 3 first. Since $f^*_{\texttt{prev}(i,j)} \preceq f^*_{i,j}$ holds by Item 1, and since $\texttt{prev\_}f = f^*_{\texttt{prev}(i,j)}$ holds by hypothesis, then $\texttt{prev\_}f(v) \preceq f^*_{i,j}(v)$ holds for every $v \in V$. Since $w'_{i,j} = D_j w_{i,j}$ and $f^*_{w'_{i,j}} = D_j f^*_{i,j}$, then $D_j \texttt{prev\_f}(v) \preceq f^*_{w'_{i,j}}(v)$ holds for every $v \in V$. Since $w'_{i,j}(e) \in \mathbb{Z}$ for every $e \in E$, then $f^*_{w'_{i,j}}(v) \in \mathbb{Z}$ for every $v \in V$, so that

$$\lceil D_j \texttt{prev\_f}(v) \rceil \preceq f^*_{w'_{i,j}}(v)$$

holds for every $v \in V$ as well. This proves Item 3.

We show Item 4 now. Since Item 3 holds, at line 13 it is correct to initialize the starting energy levels of `Value-Iteration()` to $\lceil D_j \texttt{prev\_}f(v) \rceil$ for every $v \in V$, in order to execute the Value-Iteration on input $\Gamma^{w'_{i,j}}$.

This implies the following:

$$\texttt{Value-Iteration}(\Gamma^{w'_{i,j}}, \lceil D_j \texttt{prev\_}f \rceil) = f^*_{w'_{i,j}}.$$

But we also know that $\frac{1}{D_j} f^*_{w'_{i,j}} = f^*_{i,j}$.

This proves that Item 4 holds and concludes the proof of Fact 1. □

Fact 2. We now prove that Item 2 holds at each iteration of line 8.

Proof of Fact 2. The proof proceeds by induction on (i,j).

Base Case. Let us consider the first iteration of line 8; i.e., the iteration indexed by $i = -W$ and $j = 1$. Recall that, at line 2 of Algorithm 16, the function f is initialized as $f(v) = 0$ for every $v \in V$. Notice that f is really the least SEPM $f^*_{-W,0}$ of $\Gamma_{-W,0} = \Gamma^{w+W}$, because every arc $e \in E$ has a non-negative weight in Γ^{w+W}, i.e., $w_e + W \geq 0$ for every $e \in E$.

Hence, at the first iteration of line 8, the following holds:

$$\texttt{prev\_}f = \mathbf{0} = f^*_{-W,0} = f^*_{\texttt{prev}(-W,1)}.$$

Inductive Step. Let us assume that Item 2 holds for the $\texttt{prev}(i,j)$-th iteration, and let us prove it for the (i,j)-th one. Hereafter, let us denote $(i_p, j_p) = \texttt{prev}(i,j)$ for convenience. Since Item 2 holds for the (i_p, j_p)-th iteration by induction hypothesis, then, by Fact 1, the following holds at the (i_p, j_p)-th iteration of line 13:

$$\frac{1}{D_{j_p}} \texttt{Value-Iteration}(\Gamma^{w'_{i_p,j_p}}, \lceil D_{j_p} \texttt{prev\_}f \rceil) = f = f^*_{i_p,j_p}.$$

Thus, at the (i,j)-th iteration of line 8:

$$\texttt{prev\_}f = f = f^*_{i_p,j_p} = f^*_{\texttt{prev}(i,j)}.$$

This concludes the proof of Fact 2. □

At this point, by Fact 1 and Fact 2, Lemma 6.8 follows.

We are now in the position to show that Algorithm 16 is correct.

Proposition 6.1. *Assume that Algorithm 16 is invoked on input $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ and, whence, that it returns $(\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma_0)$ as output.*

Then, $\mathcal{W}_0$ and $\mathcal{W}_1$ are the winning sets of Player 0 and Player 1 in Γ (respectively), $\nu : V \to S$ is such that $\nu(v) = \texttt{val}^\Gamma(v)$ for every $v \in V$, and $\sigma_0 : V_0 \to V$ is an optimal positional strategy for Player 0 in the MPG Γ.

Proof. At the (i,j)-th iteration of line 17, the following holds by Lemma 6.8:

$$\texttt{prev\_}f = f^*_{\texttt{prev}(i,j)} \text{ and } f = f^*_{i,j}.$$

Our aim now is that to apply Theorem 6.3. For this, firstly observe that one can safely write $\texttt{prev\_}f = f^*_{i,j-1}$. In fact, since $F_0 = 0$ and $F_{s-1} = 1$, then:

$$w_{\texttt{prev}(i,1)} = w_{i-1,s-1} = w - i = w_{i,0}, \quad \text{for every } i \in [-W, W].$$

This implies that $w_{\texttt{prev}(i,j)} = w_{i,j-1}$ for every $i \in [-W, W]$ and $j \in [1, s-1]$.

Whence, $\texttt{prev\_f} = f^*_{\texttt{prev}(i,j)} = f^*_{i,j-1}$.

So, at the (i,j)-th iteration of line 17, the following holds for every $v \in V$:

$$\begin{aligned}
\texttt{prev\_}f(v) \neq \top \text{ and } f(v) = \top \quad &\textit{iff} \quad f^*_{i,j-1}(v) \neq \top \text{ and } f^*_{i,j}(v) = \top \text{ [by Lemma 6.8]} \\
&\textit{iff} \quad v \in \mathcal{W}_0(\Gamma_{i,j-1}) \cap \mathcal{W}_1(\Gamma_{i,j}) \text{ [by Item 1-2 of Lemma 6.3]} \\
&\textit{iff} \quad \texttt{val}^\Gamma(v) = i + F_{j-1} \text{ [by Theorem 6.3]}
\end{aligned}$$

This implies that, at the (i,j)-th iteration of line 18, Algorithm 16 correctly assigns the value $\nu(v) = i + F[j-1] = i + F_{j-1}$ to the vertex v.

Since for every vertex $v \in V$ we have $\texttt{val}^{\Gamma}(v) \in S_{\Gamma}$ (recall that S_{Γ} admits the following representation $S_{\Gamma} = \{ i + F_j \mid i \in [-W, W), j \in [0, s-1] \}$), then, as soon as Algorithm 16 halts, $\nu(v) = \texttt{val}^{\Gamma}(v)$ correctly holds for every $v \in V$. In turn, at line 20 and at line 22, the winning sets $\mathcal{W}_0$ and $\mathcal{W}_1$ are correctly assigned as well.

Now, let us assume that $\nu(v) = i + F_{j-1}$ holds at the (i,j)-th iteration of line 18, for some $v \in V$. Then, the following holds on $\texttt{prev\_}w$ at line 9:

$$\texttt{prev\_}w = w_{\texttt{prev}(i,j)} = w_{i,j-1} = w - i - F_{j-1} = w - \nu(v) = w - \texttt{val}^{\Gamma}(v).$$

Thus, at the (i,j)-th iteration of line 25, for every $v \in V_0$ and $u \in \texttt{post}(v)$:

$$\begin{aligned} \texttt{prev\_}f(v) \succeq \texttt{prev\_}f(u) \ominus \texttt{prev\_}w(v,u) \text{ \emph{iff} } & f^*_{\texttt{prev}(i,j)}(v) \succeq f^*_{\texttt{prev}(i,j)}(u) \ominus \big(w - \texttt{val}^{\Gamma}(v)\big) \\ \text{\emph{iff} } & (v,u) \text{ is compatible with } f^*_{\texttt{prev}(i,j)} \text{ in } \Gamma^{w - \texttt{val}^{\Gamma}(v)} \end{aligned}$$

Recall that $f^*_{\texttt{prev}(i,j)}$ is the least SEPM of $\Gamma^{w - \texttt{val}^{\Gamma}(v)}$, thus by Theorem 6.4 the following implication holds: if (v,u) is compatible with $f^*_{\texttt{prev}(i,j)}$ in $\Gamma^{w - \texttt{val}^{\Gamma}(v)}$, then $\sigma_0(v) = u$ is an optimal positional strategy for Player 0, at v, in the MPG Γ.

This implies that line 26 of Algorithm 16 is correct and concludes the proof. $\square$

6.4.3 Complexity Analysis

The present section aims to show that Algorithm 16 always halts in $\Theta(|V|^2|E|\,W)$ time. This upper bound is established in the next proposition.

Proposition 6.2. *Algorithm 16 always halts within $\Theta(|V|^2|E|\,W)$ time and it works with $\Theta(|V|)$ space, on any input MPG $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$. Here, $W = \max_{e \in E} |w_e|$.*

Proof. (Time Complexity of the Init Phase) The initialization of $\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma_0$ (at line 1) and that of f (at line 2) takes time $O(|V|)$. The initialization of W at line 3 takes $O(|E|)$ time. To conclude, the size $s = |\mathcal{F}_{|V|}|$ of the Farey sequence (i.e., its total number of terms) can be computed in $O(n^{2/3}\log^{1/3} n)$ time as shown by Pawlewicz and Pătraşcu in [97]. Whence, the Init phase of Algorithm 16 takes $O(|E|)$ time overall.

(Time Complexity of the Scan Phases) To begin, notice that there are $O(|V|^2 W)$ scan phases overall. In fact, at line 6 the index i goes from $-W$ to W, while at line 7 the index j goes from 0 to $s-1$ where $s = |\mathcal{F}_{|V|}| = \Theta(|V|^2)$. Observe that, at each iteration, it takes $O(|E|)$ time to go from line 8 to line 12 and then from line 14 to line 27. In particular, at line 5, the j-th term F_j of the Farey sequence $\mathcal{F}_{|V|}$ can be computed in $O(n^{2/3}\log^{4/3} n)$ time as shown by Pawlewicz and Pătraşcu in [97].

Now, let us denote by $T^{13}_{i,j}$ the time taken by the (i,j)-th iteration of line 13, that is the time it takes to execute the Value-Iteration algorithm on input $\Gamma^{w'_{i,j}}$ with initial energy levels: $\lceil D_j f^*_{\texttt{prev}(i,j)} \rceil$. Then, the (i,j)-th scan phase always completes within the following time bound: $O(|E|) + T^{13}_{i,j}$.

We now focus on $T^{13}_{i,j}$ and argue that the (aggregate) total cost $\sum_{i,j} T^{13}_{i,j}$ of executing the Value-Iteration algorithm for EGs at line 13 (throughout all scan phases) is only $\Theta(|V|^2|E|\,W)$. Stated otherwise, we aim to show that the *amortized cost* of executing the (i,j)-th scan phase is only $O(|E|)$.

Recall that the Value-Iteration algorithm for EGs consists, as a first step, into an *initialization* (which takes $O(|E|)$ time) and, then, in the continuous iteration of the following two operations: (1) the application of the *lifting* operator $\delta(f,v)$ (which takes $O(|\texttt{post}(v)|)$ time) in order to resolve the inconsistency of f in v, where $f(v)$ represents the current energy level and $v \in V$ is any vertex at which f is inconsistent; and (2) the *update* of the list L (which takes $O(|\texttt{pre}(v)|)$ time), in order to keep track of all the vertices that witness an inconsistency. Recall that L contains no duplicates.

At this point, since at the (i,j)-th iteration of line 13 the Value-Iteration is executed on input $\Gamma^{w'_{i,j}}$, then a scaling factor on the maximum absolute weight W must be taken into account. Indeed, it holds that:

$$W' \triangleq \max\Big\{ |w'_{i,j}(e)| \;\Big|\; e \in E,\, i \in [-W, W],\, j \in [0, s-1] \Big\} = O(|V|\,W).$$

Remark. Actually, since $w'_{i,j} \triangleq D_j(w - i) - N_j$ (where $N_j/D_j = F_j \in \mathcal{F}_{|V|}$), then the scaling factor D_j *changes* from iteration to iteration. Still, $D_j \leq |V|$ holds for every j.

At each application of the lifting operator $\delta(f,v)$ the energy level $f(v)$ increases by at least one unit with respect to the scaled-up maximum absolute weight W'. Stated otherwise, at each application of $\delta(f,v)$, the energy level $f(v)$ increases by at least $1/|V|$ units with respect to the original weight W.

Throughout the whole computation, the rational scalings of the energy levels never decrease by Lemma 6.8: in fact, at the (i,j)-th scan phase, Algorithm 16 executes the Value-Iteration with initial energy levels: $\lceil D_j f^*_{\texttt{prev}(i,j)} \rceil$. Whence, at line 13, the (i,j)-th execution of the Value-Iteration starts from the (carefully scaled-up) energy levels of the $\texttt{prev}(i,j)$-th execution; roughly speaking, no energy gets ever lost during this process. Then, by Lemma 6.4, each energy level $f(v)$ can be lifted-up at most $|V|\,W' = O(|V|^2 W)$ times.

The above observations imply that the (aggregate) total cost of executing the Value-Iteration at line 13 (throughout all scan phases) can be bounded as follows:

$$\sum_{\substack{-W\le i\le W\\ 1\le j\le s-1}} T^{13}_{i,j} = \left(\sum_{\substack{-W\le i\le W\\ 1\le j\le s-1}} \underbrace{O(|E|)}_{\textit{init cost}} \right) + \left(\sum_{v\in V} O\big(\underbrace{|\texttt{post(v)}|}_{\textit{lifting } \delta} + \underbrace{|\texttt{pre}(v)|}_{\textit{update } \mathsf{L}}\big) \underbrace{O(|V|W')}_{\textit{Lemma } 6.4} \right)$$
$$= \Theta(|V|^2|E|W) + O(|V|^2W) \sum_{v\in V} O\big(|\texttt{post(v)}| + |\texttt{pre}(v)|\big)$$
$$= \Theta(|V|^2|E|W)$$

Whence, Algorithm 16 always halts within the following time bound:

$$\textsc{Time}\Big(\texttt{solve\_MPG}(\Gamma)\Big) = \sum_{\substack{-W\le i\le W\\ 1\le j\le s-1}} \Big(O(E) + T^{13}_{i,j}\Big) = \Theta(|V|^2|E|W).$$

This concludes the proof of the time complexity bound.

We now turn our attention to the space complexity.

(Space Complexity) First of all, although the Farey sequence $\mathcal{F}_{|V|}$ has $|\mathcal{F}_{|V|}| = \Theta(|V|^2)$ many elements, still, Algorithm 16 works fine assuming that every next element of the sequence is generated *on the fly* at line 5. This computation can be computed in $O(|V|^{2/3}\log^{4/3}|V|)$ *sub-linear* time and space as shown by Pawlewicz and Pătraşcu [97]. Secondly, given i and j, it is not necessary to actually store all weights $w'_{i,j}(e) \triangleq D_j(w(e) - i) - N_j$ for every $e \in E$, as one can compute them on the fly provided that N_j, D_j, w and e are given. Finally, Algorithm 16 needs to store in memory the two SEPMs f and $\texttt{old\_}f$, but this requires only $\Theta(|V|)$ space. Finally, at line 13, the Value-Iteration algorithm employs only $\Theta(|V|)$ space. In fact the list L, which it maintains in order to keep track of the inconsistencies, doesn't contain duplicate vertices and, therefore, its length is at most $|\mathsf{L}| \le |V|$. These facts imply altogether that Algorithm 16 works with $O(|V|)$ space. $\square$

6.5 Conclusions

In this work we proved an $O(|V|^2|E|W)$ pseudo-polynomial time upper bound for the Value Problem and Optimal Strategy Synthesis in Mean Payoff Games. The result was achieved by providing a suitable description of values and positional strategies in terms of reweighted Energy Games and Small Energy-Progress Measures.

On this way we ask whether further improvements are not too far away.

7 Faster $O(|V|^2|E|W)$-Time Energy Algorithm for Optimal Strategy Synthesis in Mean Payoff Games

Chapter Abstract

In this chapter we further strengthen the links between Mean Payoff Games (MPGs) and Energy Games (EGs). We offer a faster $O(|V|^2|E|W)$ pseudo-polynomial time and $\Theta(|V|+|E|)$ space deterministic algorithm for solving the Value Problem and Optimal Strategy Synthesis in MPGs. This improves the estimates on the pseudo-polynomial time complexity to:

$$O(|E|\log|V|) + \Theta\Big(\sum_{v\in V} \mathtt{deg}_\Gamma(v)\cdot \ell_\Gamma(v)\Big) = O(|V|^2|E|W),$$

where $\ell_\Gamma(v)$ counts the number of times that an energy-lifting operator $\delta(\cdot,v)$ is applied to any $v\in V$, along a certain sequence of Value-Iterations on reweighted EGs; and $\mathtt{deg}_\Gamma(v)$ is the degree of v. This improves significantly over the pseudo-polynomial time bound shown in Chapter 6 [35,38], i.e., $\Theta\big(|V|^2|E|W + \sum_{v\in V}\mathtt{deg}_\Gamma(v)\cdot\ell_\Gamma(v)\big) = \Theta(|V|^2|E|\,W)$, as the pseudo-polynomiality is now confined to depend solely on ℓ_Γ. The actual improvement in performance is also confirmed experimentally.

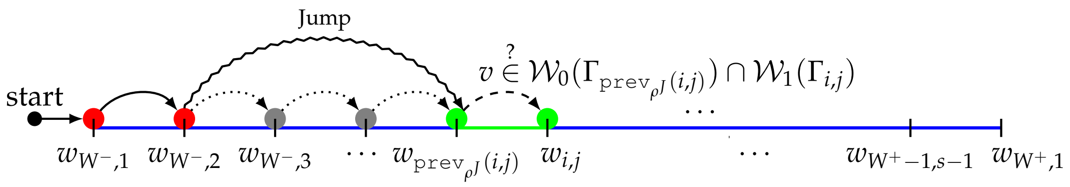

Jump
$v \overset{?}{\in} \mathcal{W}_0(\Gamma_{\mathtt{prev}_{\rho^J}(i,j)}) \cap \mathcal{W}_1(\Gamma_{i,j})$
start
$w_{W^-,1}$
$w_{W^-,2}$
$w_{W^-,3}$
$\cdots$
$w_{\mathtt{prev}_{\rho^J}(i,j)}$
$w_{i,j}$
$\cdots$
$w_{W^+-1,s-1}$
$w_{W^+,1}$

Figure 7.1: An illustration of the MPG algorithm offered in Chapter 7.

This chapter is a revised version of [37].

7.1 Introduction

As already seen in Chapter 6, a *Mean Payoff Game* (MPG) is a 2-player infinite game $\Gamma \triangleq (V, E, w, \langle V_0, V_1 \rangle)$ played on a finite weighted directed graph. Some other related models have been studied widely in the literature. For instance, several research efforts have been spent in studying quantitative extensions of infinite games for modeling quantitative aspects of reactive systems, like *Energy Games (EGs)* [12, 14, 18]. In this context quantities may represent, e.g., the power usage of an embedded component, or the buffer size of a networking element. These studies unveiled interesting connections with MPGs, and they have recently led to the design of faster procedures for solving them. In particular, [14] devised a faster deterministic algorithm for solving the Value Problem and Optimal Strategy Synthesis in MPGs within $O\big(|V|^2|E|\,W\log(|V|\,W)\big)$ pseudo-polynomial time and polynomial space. Essentially, a binary search is directed by the resolution of multiple reweighted EGs. The determination of EGs comes from repeated applications of energy-lifting operators $\delta(\cdot,v)$ for $v \in V$. These are all monotone functions defined on a complete lattice (the Energy-Lattice of a reweighted EG), so the correct termination is ensured by the Knaster–Tarski's fixed point theorem [111].

In Chapter 6, the worst-case time complexity of the Value Problem and Optimal Strategy Synthesis was given an improved pseudo-polynomial upper bound [35, 38]. That chapter focused on offering a simple proof of the improved time complexity bound. The algorithm there proposed, Algorithm 16, had the advantage of being very simple; its existence made it possible to discover and to analyze some of the underlying fundamental ideas, that ultimately led to the improved upper bound, more directly; it was shown appropriate to supersede (at least in the perspective of obtaining sharpened bounds) the above mentioned binary search by sort of a linear search that succeeds at amortizing all the energy-liftings throughout the computation. However, its running time turns out to be also $\Omega(|V|^2|E|W)$, the actual time complexity being $\Theta\big(|V|^2|E|\,W + \sum_{v\in V} \mathtt{deg}_\Gamma(v)\cdot \ell^0_\Gamma(v)\big)$, where $\ell^0_\Gamma(v) \leq (|V|-1)|V|W$ denotes the total number of times that the energy-lifting operator $\delta(\cdot,v)$ is applied to any $v \in V$ by Algorithm 16. After the appearance of those works, a way to overcome this issue was found.

7.1.1 Contribution

This chapter aims at further strenghtening the relationship between MPGs and EGs.

Our results are summarized as follows.

Faster $O(|V|^2|E|W)$-Time Algorithm for MPGs by Jumping through Reweighted EGs.

We introduce a novel algorithmic scheme, named *Jumping* (Algorithm 15), which tackles on some further regularities of the problem, thus reducing the estimate on the pseudo-polynomial time complexity of MPGs to:

$$O(|E|\log|V|) + \Theta\Big(\sum_{v\in V} \texttt{deg}_\Gamma(v)\cdot \ell^1_\Gamma(v)\Big),$$

where $\ell^1_\Gamma(v)$ is the total number of applications of $\delta(\cdot, v)$ to $v \in V$ that are made by Algorithm 1; $\ell^1_\Gamma \leq (|V|-1)|V|W$ (worst-case; but experimentally, $\ell^1_\Gamma \ll \ell^0_\Gamma$), and the working space is $\Theta(|V|+|E|)$. Overall the worst-case complexity is still $O(|V|^2|E|W)$, but the pseudo-polynomiality is now confined to depend solely on the total number ℓ^1_Γ of required energy-liftings; this is not known to be $\Omega(|V|^2|E|W)$ generally, and future theoretical or practical advancements concerning the Value-Iteration framework for EGs could help reducing this metric. Under this perspective, the computational equivalence between MPGs and EGs seems now like a bit more unfolded and subtle. In practice, Algorithm 15 allows us to reduce the magnitude of ℓ_Γ considerably, w.r.t. [35, 38], and therefore the actual running time; our experiments suggest that $\ell^1_\Gamma \ll \ell^0_\Gamma$ holds for wide families of MPGs (see SubSection 7.3.4).

In summary, the present work offers a *faster* pseudo-polynomial time algorithm; theoretically the pseudo-polynomiality now depends only on ℓ^1_Γ, and in practice the actual performance also improves considerably w.r.t. [35,38]. With hindsight, Algorithm 16 turned out to be a high-level description and the tip of a more technical and efficient underlying procedure. This is the first truly $O(|V|^2|E|W)$ time deterministic algorithm, for solving the Value Problem and Optimal Strategy Synthesis in MPGs, that can be effectively applied in practice (optionally, in interleaving with some other known sub-exponential time algorithms).

Indeed, a wide spectrum of different approaches have been investigated in the literature. For instance, [1] provided a fast *randomized* algorithm for solving MPGs in *sub-exponential* time $O\big(|V|^2|E|\exp\big(2\sqrt{|V|\ln(|E|/\sqrt{|V|})} + O(\sqrt{|V|} + \ln|E|)\big)\big)$. [79] devised a deterministic $O(2^{|V|}|V||E|\log W)$ *singly-exponential* time procedure by considering a so called potential-theory of MPGs, that is akin to EGs.

Table 7.1 offers a summary of past and present results.

7.1.2 Organization

In Section 7.2, we introduce some notation and provide the required background on infinite 2-player pebble games and related algorithmic results. In Section 7.3, it is presented an $O(|E|\log|V|) + \Theta\big(\sum_{v\in V}\texttt{deg}_\Gamma(v)\cdot\ell^1_\Gamma(v)\big) = O(|V|^2|E|W)$ pseudo-polynomial time and $\Theta(|V|+|E|)$ space deterministic algorithm for solving the Value Problem and Optimal Strategy Synthesis in MPGs; SubSection 7.3.4 offers an experimental comparison between Algorithm 15 and Algorithm 16 [35,38].

Table 7.1: Time Complexity of the main Algorithms for solving MPGs.

Algorithm	Optimal Strategy Synthesis	Value Problem
This work	$O(\|E\|\log\|V\|)+\Theta\big(\sum_{v\in V}\mathtt{deg}_\Gamma(v)\cdot\ell^1_\Gamma(v)\big)$	*same complexity*
CRStrat15-16	$\Theta\big(\|V\|^2\|E\|W+\sum_{v\in V}\mathtt{deg}_\Gamma(v)\cdot\ell^0_\Gamma(v)\big)$	*same complexity*
BC11	$O(\|V\|^2\|E\|W\log(\|V\|W))$	*same complexity*
LP07	n/a	$O(\|V\|\|E\|2^{\|V\|}\log W)$
AV06	$O\Big(\|V\|^2\|E\|e^{2\sqrt{\|V\|\ln\big(\frac{\|E\|}{\sqrt{\|V\|}}\big)}+O(\sqrt{\|V\|}+\ln\|E\|)}\Big)$	*same complexity*
ZP96	$\Theta(\|V\|^4\|E\|W\log\frac{\|E\|}{\|V\|})$	$\Theta(\|V\|^3\|E\|W)$

7.2 Background and Notation

We refer the reader to Section 6.2 of Chapter 6 for the background notation on MPGs. Also recall that the *Farey sequence* $\mathcal{F}_n$ is the increasing sequence of all irreducible fractions from the (rational) interval $[0,1]$ with denominators less than or equal to n. In Section 6.4 we will be interested in generating the Farey sequence $F_0,\dots,F_{s-1}$, one term after another, iteratively and efficiently. As mentioned in [97], combining several properties satisfied by the Farey sequence, one can get a trivial iterative algorithm, which generates the next term $F_j = N_j/D_j$ of $\mathcal{F}_n$ based on the previous two:

$$N_j \leftarrow \left\lfloor \frac{D_{j-2}+n}{D_{j-1}} \right\rfloor \cdot N_{j-1} - N_{j-1}; \qquad D_j \leftarrow \left\lfloor \frac{D_{j-2}+n}{D_{j-1}} \right\rfloor \cdot D_{j-1} - D_{j-1}.$$

Given F_{j-2}, F_{j-1} as input, this computes F_j in $O(1)$ time and space.

7.3 A Faster $O(|V|^2|E|W)$-Time Algorithm for MPGs by Jumping through Reweighted EGs

This section offers an $O(|E|\log|V|)+\Theta\big(\sum_{v\in V}\mathtt{deg}_\Gamma(v)\cdot\ell^1_\Gamma(v)\big) = O(|V|^2|E|W)$ time algorithm for solving the Value Problem and Optimal Strategy Synthesis in MPGs $\Gamma=(V,E,w,\langle V_0,V_1\rangle)$, where $W\triangleq\max_{e\in E}|w_e|$; it works with $\Theta(|V|+|E|)$ space. Its name is Algorithm 15.

In order to describe it in a suitable way, let us firstly recall some notation. Given an MPG Γ, we shall consider the following reweightings:

$$\Gamma_{i,j} \cong \Gamma^{w-i-F_j}, \text{ for any } i\in[-W,W] \text{ and } j\in[1,s-1],$$

where $s\triangleq|\mathcal{F}_{|V|}|$, and F_j is the j-th term of $\mathcal{F}_{|V|}$.

Assuming $F_j=N_j/D_j$ for some (co-prime) $N_j,D_j\in\mathbb{N}$, we work with the following weights:

$$w_{i,j} \triangleq w - i - F_j = w - i - N_j/D_j; \qquad w'_{i,j} \triangleq D_j w_{i,j} = D_j(w-i) - N_j.$$

Recall $\Gamma_{i,j} \triangleq \Gamma^{w'_{i,j}}$ and $\forall^{e\in E} w'_{i,j}(e) \in \mathbb{Z}$. Notice, since $F_1 < \ldots < F_{s-1}$ is monotone increasing, $\{w_{i,j}\}_{i,j}$ can be ordered (inverse)-lexicographically w.r.t. (i,j); i.e., $w_{(i,j)} > w_{(i',j')}$ *iff* either: $i < i'$, or $i = i'$ and $j < j'$; e.g., $w_{W^-,1} > w_{W^-,2} > \ldots > w_{W^-,s-1} > \ldots > w_{W^+-1,s-1} > w_{W^+,s-1}$. Also, we denote the least-SEPM of the reweighted EG $\Gamma_{i,j}$ by $f^*_{w'_{i,j}} : V \to \mathcal{C}_{\Gamma_{i,j}}$. In addition, $f^*_{i,j} : V \to \mathbf{Q}$ denotes the *rational-scaling* of $f^*_{w'_{i,j}}$, which is defined as: $\forall^{v\in V} f^*_{i,j}(v) \triangleq \frac{1}{D_j} \cdot f^*_{w'_{i,j}}(v)$. Finally, if f is any SEPM of the EG $\Gamma_{i,j}$, then $\text{Inc}(f,i,j) \triangleq \{v \in V \mid v \text{ is inconsistent w.r.t. } f \text{ in } \Gamma_{i,j}\}$.

7.3.1 Description of Algorithm 15

Outline. Given an input arena $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$, Algorithm 15 aims at returning a tuple $(\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma^*_0)$ where: $\mathcal{W}_0$ is the winning set of Player 0 in the MPG Γ, and $\mathcal{W}_1$ is that of Player 1; $\nu : V \to S_\Gamma$ maps each starting position $v_s \in V$ to $\texttt{val}^\Gamma(v_s)$; finally, $\sigma^*_0 : V_0 \to V$ is an optimal positional strategy for Player 0 in the MPG Γ.

Let $W^- \triangleq \min_{e\in E} w_e$ and $W^+ \triangleq \max_{e\in E} w_e$. The first aspect underlying Algorithm 15 is that of ordering $[W^-, W^+] \times [1, s-1]$ lexicographically, by considering the already mentioned (decreasing) sequence of weights:

$$\rho : [W^-, W^+] \times [1, s-1] \to \mathbb{Z}^E : (i,j) \mapsto w_{i,j},$$

$$\rho : w_{W^-,1} > w_{W^-,2} > \ldots > w_{W^-,s-1} > w_{W^-+1,1} > w_{W^-+1,2} > \ldots > w_{W^+-1,s-1} > \ldots > w_{W^+,s-1},$$

then, to rely on Theorem 6.3, at each step of ρ, testing whether some *transition of winning regions* occurs. At the generic (i,j)-th step of ρ, we run a Value-Iteration [14] in order to compute the least-SEPM of $\Gamma_{i,j}$, and then we check for every $v \in V$ whether v is winning for Player 1 w.r.t. the *current* weight $w_{i,j}$ (i.e., w.r.t. $\Gamma_{i,j}$), meanwhile recalling whether v was winning for Player 0 w.r.t. the (immediately, inverse-lex) *previous* weight $w_{\texttt{prev}_\rho(i,j)}$ (i.e., w.r.t. $\Gamma_{\texttt{prev}_\rho(i,j)}$). This step relies on Lemma 6.3, as in fact $\mathcal{W}_0(\Gamma_{\texttt{prev}_\rho(i,j)}) = V_{f^*_{\texttt{prev}_\rho(i,j)}}$ and $\mathcal{W}_1(\Gamma_{i,j}) = V \setminus V_{f^*_{i,j}}$.

If a transition occurs, say for some $\hat{v} \in \mathcal{W}_0(\Gamma_{\texttt{prev}_\rho(i,j)}) \cap \mathcal{W}_1(\Gamma_{i,j})$, then $\texttt{val}^\Gamma(\hat{v})$ can be computed easily by relying on Theorem 6.3, i.e., $\nu(\hat{v}) \leftarrow i + F[j-1]$; also, an optimal positional strategy can be extracted from $f^*_{\texttt{prev}_\rho(i,j)}$ thanks to Theorem 6.4 and Remark 6.1, provided that $\hat{v} \in V_0$.

Each phase, in which one does a Value-Iteration and looks at transitions of winning regions, it is named *Scan-Phase*. Remarkably, for every $i \in [W^-, W^+]$ and $j \in [1, s-1]$, the *(i,j)-th* Scan-Phase performs a Value-Iteration [14] on the reweighted EG $\Gamma_{i,j}$ by initializing all the energy-levels to those computed by the previous Scan-Phase (subject to a suitable re-scaling and a rounding-up, i.e., $\lceil D_j \cdot f^*_{\texttt{prev}_\rho(i,j)} \rceil$). As described in [38], the main step of computation that is

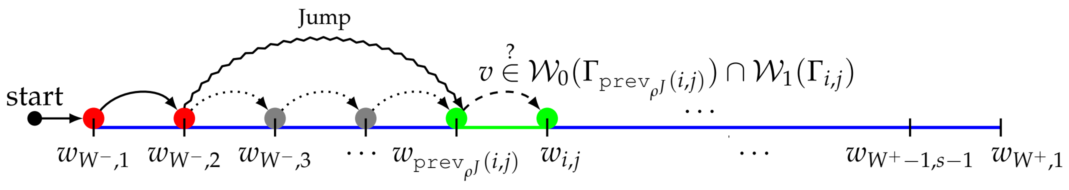

Jump
$v \overset{?}{\in} \mathcal{W}_0(\Gamma_{\texttt{prev}_{\rho^J}(i,j)}) \cap \mathcal{W}_1(\Gamma_{i,j})$
start
$w_{W^-,1}$
$w_{W^-,2}$
$w_{W^-,3}$
$w_{\texttt{prev}_{\rho^J}(i,j)}$
$w_{i,j}$
$w_{W^+-1,s-1}$
$w_{W^+,1}$

Figure 7.2: An illustration of Algorithm 15.

carried on at the (i,j)-th Scan-Phase goes therefore as follows:

$$f_{i,j} \leftarrow \frac{1}{D_j}\texttt{Value-Iteration}\Big(\Gamma_{i,j}, \lceil D_j \cdot f^*_{\texttt{prev}_\rho(i,j)} \rceil\Big),$$

where D_j is the denominator of F_j. Then, one can prove that $\forall (i,j)\, f_{i,j} = f^*_{i,j}$ (see Chapter 6, Lemma 6.8, Item 4). Indeed, by Lemma 6.2 and Lemma 6.3, $\mathcal{W}_0(\Gamma_{\texttt{prev}_\rho(i,j)}) = V_{f^*_{\texttt{prev}_\rho(i,j)}}$ and $\mathcal{W}_1(\Gamma_{i,j}) = V \setminus V_{f^*_{i,j}}$. And since ρ is monotone decreasing, the sequence of energy-levels $\psi_\rho : (i,j) \mapsto f^*_{i,j}$ is monotone non-decreasing (see Chapter 6, Lemma 6.8, Item 1):

$$\psi_\rho : f^*_{W^-,1} \preceq f^*_{W^-,2} \preceq \cdots \preceq f^*_{W^-,s-1} \preceq f^*_{W^-+1,1} \preceq f^*_{W^-+1,2} \preceq \cdots \preceq f^*_{W^+-1,s-1} \preceq \cdots \preceq f^*_{W^+,s-1};$$

Our algorithm will succeed at *amortizing* the cost of the corresponding sequence of Value-Iterations for computing ψ_ρ. Recall that a similar amortization takes place already in Algorithm 16. However, Algorithm 16 performs exactly one Scan-Phase (i.e., one Value-Iteration, plus the tests $v \in^? \mathcal{W}_0(\Gamma_{\texttt{prev}_\rho(i,j)}) \cap \mathcal{W}_1(\Gamma_{i,j})$) for each term of ρ –without making any *Jump* in ρ–. Thus, Algorithm 16 performs $\Theta(|V|^2 W)$ Scan-Phases overall, each one costing $\Omega(|E|)$ time (i.e., the cost of initializing the Value-Iteration as in [14]). This brings an overall $\Omega(|V|^2|E|W)$ time complexity, which turns out to be also $O(|V|^2|E|W)$; leading us to an improved pseudo-polynomial time upper bound for solving MPGs [35,38].

The present work shows that it is instead possible, and actually very convenient, to perform many *Jumps* in ρ; thus introducing "gaps" between the weights that are considered along the sequence of Scan-Phases. The corresponding sequence of weights is denoted by ρ^J. This is Algorithm 15. In Fig. 7.2, a graphical intuition of Algorithm 15 and ρ^J is given, in which a Jump is depicted with an arc going from $w_{W^-,2}$ to $w_{\texttt{prev}_{\rho^J}(i,j)}$, e.g., $w_{\texttt{prev}_{\rho^J}(\texttt{prev}_{\rho^J}(i,j))} = w_{W^-,2}$.

Two distinct kinds of Jumps are employed: *Energy-Increasing-Jumps (EI-Jumps)* and *Unitary-Advance-Jumps (UA-Jumps)*. Briefly, EI-Jumps allow us to satisfy a suitable invariant:

[*Inv-EI*] Whenever a Scan-Phase is executed (each time that a Value-Iteration is invoked), an energy-level $f(v)$ strictly increases for at least one $v \in V$. There will be no *vain* Scan-Phase (i.e., such that all the energy-levels stand still); so, δ will be applied (successfully) at least once per Scan-Phase. Therefore, ψ_{ρ^J} will be monotone increasing (except at the steps of backtracking introduced next, but there will be at most $|V|$ of them). □

Indeed, the UA-Jumps are employed so to scroll through $\mathcal{F}_{|V|}$ only *when* and *where* it is really necessary. Consider the following facts.

– Suppose that Algorithm 15 came at the end of the $(i,s-1)$-th Scan-Phase, for some $i \in [W^-, W^+]$; recall that $F_{s-1} = 1$, so $w_{i,s-1} = w'_{i,s-1}$ is integral. Then, Algorithm 16 would scroll through $\mathcal{F}_{|V|}$ entirely, by invoking one Scan-Phase per each term, going from the $(i+1,1)$-th to the $(i+1,s-1)$-th, meanwhile testing whether a transition of winning regions occurs; notice, $w_{i+1,s-1}$ is integral again. Instead, to UA-Jump means to jump in advance (proactively) from $w_{i,s-1}$ to $w_{i+1,s-1}$, by making a Scan-Phase on input $\Gamma_{i+1,s-1}$, thus skipping all those from the $(i+1,1)$-th to the $(i+1,s-2)$-th one. After that, Algorithm 15 needs to *backtrack* to $w_{i,s-1}$, and to scroll through $\mathcal{F}_{|V|}$, if and only if $\mathcal{W}_0(\Gamma_{w_{i,s-1}}) \cap \mathcal{W}_1(\Gamma_{w_{i+1,s-1}}) \neq \emptyset$. Otherwise, it is safe to keep the search going on, from $w_{i+1,s-1}$ on out, making another UA-Jump to $w_{i+2,s-1}$. The backtracking step may happen at most $|V|$ times overall, because some value $\nu(v)$ is assigned to some $v \in V$ at each time. So, Algorithm 15 scrolls entirely through $\mathcal{F}_{|V|}$ at most $|V|$ times; i.e., only *when* it is really necessary.

– Remarkably, when scrolling through $\mathcal{F}_{|V|}$, soon after the above mentioned backtracking step, the corresponding sequence of Value-Iterations really need to lift-up again (more slowly) *only* the energy-levels of the sub-arena of Γ that is induced by $S \triangleq \mathcal{W}_0(\Gamma_{w_{i,s-1}}) \cap \mathcal{W}_1(\Gamma_{w_{i+1,s-1}})$. All the energy-levels of the vertices in $V \setminus S$ can be confirmed and left unchanged during the UA-Jump's backtracking step; and they will all stand still, during the forthcoming sequence of Value-Iterations (at least, until a new EI-Jump will occur), as they were computed just *before* the occurence of the UA-Jump's backtracking step. This is why Algorithm 15 scrolls through $\mathcal{F}_{|V|}$ only *where* it is really necessary.

– Also, Algorithm 15 succeeds at *interleaving* EI-Jumps and UA-Jumps, thus making only one single pass through ρ^J (plus the backtracking steps).

Altogether these facts are going to reduce the running time considerably.

Definition 7.1 (ℓ^1_Γ). *Given an input MPG Γ, let $\ell^1_\Gamma(v)$ be the total number of times that the energy-lifting operator $\delta(\cdot, v)$ is applied to any $v \in V$ by Algorithm 15 (notice that it will be applied only at line 3 of `J-VI()`, see SubProcedure 6).*

Then, the following remark holds on Algorithm 15.

Remark 7.1. *Jumping is not heuristic, the theoretical running time of the procedure improves exactly, from:*

$$\Theta(|V|^2|E|W + \sum_{v\in V} deg_\Gamma(v) \cdot \ell^0_\Gamma(v)) \text{ (Algorithm 16)}$$

to:

$$O(|E|\log|V|) + \Theta\big(\sum_{v\in V} deg_\Gamma(v) \cdot \ell^1_\Gamma(v)\big) \text{ (Algorithm 15),}$$

where $\ell^1_\Gamma \leq (|V|-1)|V|W$; which is still $O(|V|^2|E|W)$ in the worst-case, but it isn't known to be $\Omega(|V|^2|E|W)$ generally. In practice, this reduces the magnitude of ℓ_Γ significantly, i.e., $\ell^1_\Gamma \ll \ell^0_\Gamma$ is observed in our experiments (see SubSection 7.3.4).

To achieve this, we have to overcome some subtle technical issues. Firstly, we show that it is unnecessary to re-initialize the Value-Iteration at each Scan-Phase (this would cost $\Omega(|E|)$ each time otherwise), even when making wide jumps in ρ. Instead, it will be sufficient to perform an initialization phase only at the beginning, paying only $O(|E|\log|V|)$ total time and a linear space in pre-processing. For this, we will provide a suitable readjustment of the Value-Iteration; it is named `J-VI()` (SubProcedure 6). Briefly, the Value-Iteration of [14] employs an array of counters, $\texttt{cnt} : V_0 \to \mathbb{N}$, in order to check in time $O(|N^{\text{in}}_\Gamma(v)|)$ which vertices $u \in N^{\text{in}}_\Gamma(v) \cap V_0$ have become inconsistent (soon after that the energy-level $f(v)$ was increased by applying $\delta(f,v)$ to some $v \in V$), and should therefore be added to the list L^{inc} of inconsistent vertices. One subtle issue here is that, when going from the $\texttt{prev}_{\rho^J}(i,j)$-th to the (i,j)-th Scan-Phase, the *coherency* of `cnt` can break (i.e., `cnt` may provide false-positives, thus classifying a vertex as consistent when it isn't really so). This may happen when $w_{\texttt{prev}_{\rho^J}(i,j)} > w_{(i,j)}$ (which is always the case, except for the UA-Jump's backtracking steps). This is even amplified by the EI-Jumps, as they may lead to wide jumps in ρ. The algorithm in [38] recalculates `cnt` from scratch, at the beginning of each Scan-Phase, thus paying $\Omega(|E|)$ time per each. In this work, we show how to keep `cnt` coherent throughout the Jumping Scan-Phases, efficiently. Actually, even in Algorithm 15 the coherency of `cnt` can possibly break, but Algorithm 15 succeeds at *repairing* all the incoherencies that may happen during the whole computation in $\Theta(|E|)$ *total* time – just by paying $O(|E|\log|V|)$ time in pre-processing. This is a very convenient trade-off. At this point we should begin entering into the details of Algorithm 15.

Jumper. We employ a container data-structure, which is denoted by J. It comprises a bunch of arrays, maps, plus an integer variable $J.i$. Concerning maps, the key universe is V or E; i.e., keys are restricted to a narrow range of integers ($[1,|V|]$ or $[1,|E|]$, depending on the particular case).

We suggest direct addressing: the value binded to a key $v \in V$ (or $(u,v) \in E$) is stored at $A[v]$ (resp., $A[(u,v)]$); if there is no binding for key v (resp., (u,v)), the cell stores a sentinel, i.e., $A[v] = \bot$. Also, we would need to iterate efficiently through A (i.e., without having to scroll entirely through A). This is easy to implement by handling pointers in a suitable way; one may also keep a list L_A associated to A, explicitly, storing one element for each $(k,v) \neq \bot$ of A; every time that an item is added to or removed from A, then L_A is updated accordingly, in time $O(1)$ (by handling pointers). The *last* entry inserted into A (the key of which isn't already binded at insertion time) goes in *front* of L_A. We say that $L \triangleq (A, L_A)$ is an *array-list*, and we dispose of the following operations: $\texttt{insert}((k,\texttt{val}),L)$, which binds `val` to k by inserting $(k,\texttt{val})$ into L (if any $(k,\texttt{val}')$ is already in L, then $\texttt{val}'$ gets overwritten by `val`); $\texttt{remove}(k,L)$ deletes an entry $(k,\texttt{val})$ from L; $\texttt{pop\_front}(L)$, removes from

L the *last* $(k,\texttt{val})$ that was inserted (and whose key was not already binded at the time of the insertion, i.e., the *front*) also returning it; $\texttt{for\_each}\big((k,\texttt{val}) \in L\big)$ iterates through the entries of L efficiently (i.e., skipping the sentinels). Notice, any sequence of `insert` and `pop_front` on L implements a LIFO policy.

So, J comprises: an integer variable $J.i$; an array $J.f : V \to \mathbf{Q}$; an array $J.\texttt{cnt} : V_0 \to \mathbb{N}$; an array $J.\texttt{cmp} : \{(u,v) \in E \mid u \in V_0\} \to \{\texttt{T},\texttt{F}\}$; a bunch of array-lists, $L_f : V \to \mathbb{N}$, and $L^{\text{inc}}, L^{\text{inc}}_{\text{nxt}}, L^{\text{inc}}_{\text{cpy}}, L_\top : V \to \{*\}$; finally, a special array-list L_ω indexed by $\{w_e \mid e \in E\}$, whose values are in turn (classical, linked) lists of arcs, denoted L_α; L_ω is filled in pre-processing as follows: $(\hat{w}, L_\alpha) \in L_\omega$ *iff* $L_\alpha = \{e \in E \mid w_e = \hat{w}\}$. The subprocedure `init_jumper()` (SubProcedure 3) takes care of initializing J.

SubProcedure 3: Init Jumper J

```
SubProcedure init_jumper($J$,$\Gamma$)
    input : Jumper $J$, an MPG $\Gamma$.
1   $L_f, L^{\text{inc}}, L^{\text{inc}}_{\text{nxt}}, L^{\text{inc}}_{\text{cpy}}, L_\top, L_\omega \leftarrow \emptyset$;
2   foreach $v \in V$ do
3       $J.f[v] \leftarrow 0$;
4       if $v \in V_0$ then
5           $J.\texttt{cnt}[v] \leftarrow |N^{\text{out}}_\Gamma(v)|$;
6   foreach $(u,v,w) \in E$ do
7       if $v \in V_0$ then
8           $J.\texttt{cmp}[(u,v)] \leftarrow \texttt{T}$;
9       if $L_\omega[w] = \perp$ then
10          insert$((w,\emptyset), L_\omega)$;
11      insert$((u,v), L_\omega[w])$;
12  Sort $L_\omega$ in increasing order w.r.t. the keys $w$;
```

At the beginning, all array-lists are empty (line 1). For every $v \in V$ (line 2), we set $J.f[u] = 0$ and, if $v \in V_0$, then $J.\texttt{cnt}[v] \leftarrow |N^{\text{out}}_\Gamma(v)|$ (lines 3-5). Then, each arc $(u,v,w) \in E$ is flagged as *compatible*, i.e., $J.\texttt{cmp}[(u,v)] \leftarrow \texttt{T}$ (lines 6-8); also, if L_ω doesn't contain an entry already binded to $w(u,v)$, then an empty list of arcs is inserted into L_ω as an entry $(w,\emptyset)$ (lines 9-10); then, in any case, the arc (u,v) is added to the unique L_α which is binded to $w = w(u,v)$ in L_ω (line 11). At the end (line 12), all the elements of L_ω are sorted in increasing order w.r.t. their weight keys, w_e for $e \in E$ (e.g., (W^-, L_α) goes in front of L_ω). This concludes the initialization of J; it takes $O(|E|\log|V|)$ time and $\Theta(|V| + |E|)$ space.

Main Procedure: `solve_MPG()`. The main procedure of Algorithm 15 is organized as follows. Firstly, the algorithm performs an initialization phase; which includes $\texttt{init\_jumper}(J,\Gamma)$.

The variables $\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma^*_0$ are initially empty (line 1). Also, $W^- \leftarrow \min_{e\in E} w_e$, $W^+ \leftarrow \max_{e\in E} w_e$ (line 2). And F is a reference to the Farey's terms, say $\{F[j] \mid j \in [0, s-1]\} = \mathcal{F}_{|V|}$, and $s \leftarrow |\mathcal{F}_{|V|}|$ (line 3). At line 4, J is initialized by $\texttt{init\_jumper}(J,\Gamma)$ (SubProcedure 3).

Then the Scan-Phases start.

After setting $i \leftarrow W^- - 1$, $j \leftarrow 1$ (line 5), Algorithm 15 enters into a `while` loop (line 6), which lasts until both `ei-jump`$(i,J) = \texttt{T}$ at line 7, *and* $L^{\text{inc}} = \emptyset$ at line 8, hold; in which case $(\mathcal{W}_0,\mathcal{W}_1,\nu,\sigma_0^*)$ is returned (line 8) and Algorithm 15 halts. Inside the `while` loop, `ei-jump`(i,J) (SubProcedure 8) is invoked (line 7). This checks whether or not to make an EI-Jump; if so, the ending point of the EI-Jump (the new value of i) is stored into $J.i$. This will be the starting point for making a sequence of UA-Jumps, which begins by invoking `ua-jumps`$(J.i,s,F,J,\Gamma)$ at line 9. When the `ua-jumps()` halts, it returns $(\hat{\imath},S)$, where: $\hat{\imath}$ is the new value of i (line 9), for some $\hat{\imath} \geq J.i$; and S is a set of vertices such that $S = \mathcal{W}_0(\Gamma_{w_{\hat{\imath}-1,s-1}}) \cap \mathcal{W}_1(\Gamma_{w_{\hat{\imath},s-1}})$.

Algorithm 15: Main Procedure

```
Procedure solve_MPG(Γ)
    input  : An MPG $\Gamma = (V,E,w,\langle V_0,V_1\rangle)$.
    output: $(\mathcal{W}_0,\mathcal{W}_1,\nu,\sigma_0^*)$.
1   $\mathcal{W}_0,\mathcal{W}_1,\nu,\sigma_0^* \leftarrow \emptyset$; /*Init Phase*/
2   $W^- \leftarrow \min_{e\in E} w_e$; $W^+ \leftarrow \max_{e\in E} w_e$;
3   $F \leftarrow$ reference to $\mathcal{F}_{|V|}$; $s \leftarrow |\mathcal{F}_{|V|}|$;
4   init_jumper$(J,\Gamma)$;
5   $i \leftarrow W^- - 1$; $j \leftarrow 1$; /*Jumping Scan-Phases*/
6   while T do
7       if ei-jump$(i,J)$ then
8           if $L^{inc} = \emptyset$ then return $(\mathcal{W}_0,\mathcal{W}_1,\nu,\sigma_0^*)$;
9           $(i,S) \leftarrow$ ua-jumps$(J.i,s,F,J,\Gamma)$;
10          $j \leftarrow 1$;
11      J-VI$(i,j,F,J,\Gamma[S])$;
12      set_vars$(\mathcal{W}_0,\mathcal{W}_1,\nu,\sigma_0^*,i,j,F,J,\Gamma[S])$;
13      scl_back_f$(j,F,J)$;
14      $j \leftarrow j+1$;
```

Next, $j \leftarrow 1$ is set (line 10), as Algorithm 15 is now completing the backtracking from $w_{\hat{\imath},s-1}$ to $w_{\hat{\imath},1}$, in order to begin scrolling through $\mathcal{F}_{|V|}$ by running a sequence of `J-VI()` at line 11. Such a sequence of `J-VI()`s will last until the occurence of another EI-Jump at line 7, that in turn will lead to another sequence of UA-Jumps at line 9, and so on. So, a `J-VI()` (SubProcedure 6) is executed on input $(\hat{\imath},j,F,J,\Gamma[S])$ at line 11. We remark that, during the `J-VI`$(i,j,F,J,\Gamma[S])$, the energy-levels are scaled up, from $\mathbf{Q}$ to $\mathbb{N}$; actually, from $J.f$ to $\lceil D_j \cdot J.f \rceil$, where D_j is the denominator of F_j. Also, `J-VI`$(i,j,F,J,\Gamma[S])$ (SubProcedure 6) is designed so that, when it halts, $L_\top = \mathcal{W}_0(\Gamma_{\texttt{prev}_{\rho^J}(i,j)}) \cap \mathcal{W}_1(\Gamma_{i,j})$. Then, `set_vars()` is invoked on input $(\mathcal{W}_0,\mathcal{W}_1,\nu,\sigma_0^*,i,j,F,J,\Gamma[S])$ (line 12): this checks whether some value and optimal strategy needs to be assigned to ν and σ_0^* (respectively). Next, all of the energy-levels are scaled back, from $\mathbb{N}$ to $\mathbf{Q}$, and stored back into $J.f$: this is done by invoking `scl_back_`$f(j,F,J)$ (line 13). Finally, $j \leftarrow j+1$ (line 14) is assigned (to step through the sequence $\mathcal{F}_{|V|}$ during the `while` loop at line 7). This concludes `solve_MPG()`, which is the main procedure of Algorithm 15.

Set Values and Optimal Strategy. Let us provide the details of `set_vars()` (SubProcedure 4). It takes $(\mathcal{W}_0,\mathcal{W}_1,\nu,\sigma_0^*,i,j,F,\Gamma)$ in input, where $i \in [W^-,W^+]$ and $j \in [1,s-1]$. At line 1, $D = D_{j-1}$ is the denominator of F_{j-1}. Then, all of the following operations are repeated while $L_\top \neq \emptyset$ (line 2). Firstly, the front element u of $L_\top$ is popped (line 3); recall, it will turn out that $u \in \mathcal{W}_0(\Gamma_{i,j-1}) \cap \mathcal{W}_1(\Gamma_{i,j})$, thanks to the specs of `J-VI()` (SubProcedure 6). For this reason, the optimal value of u in the MPG Γ is set to $\nu(u) \leftarrow i + F[j-1]$ (line 4); and, if $\nu(u) \geq 0$, u is added to the winning region $\mathcal{W}_0$; else, to $\mathcal{W}_1$ (line 5). The correctness of lines 4-5 relies on Theorem 6.3. If $u \in V_0$ (line 6), it is searched an arc $(u,v) \in E$ that is compatible w.r.t. $D_{j-1} \cdot J.f$ in $\Gamma_{i,j-1}$ (line 8), i.e., it is searched some $v \in N^{\text{out}}_\Gamma(u)$ such that:

$$(D \cdot J.f[u]) \succeq (D \cdot J.f[v]) \ominus \texttt{get\_scl\_}w\big(w(u,v), i, j-1, F\big) \text{ (line 8);}$$

By Theorem 6.4, setting $\sigma^*_0(u) \leftarrow v$ (line 9) brings an optimal positional strategy for Player 0 in the MPG Γ. Here, $\texttt{get\_scl\_}w\big(w,i,j-1,F\big)$ simply returns $D_{j-1} \cdot (w(u,v) - i) - N_{j-1}$, where: N_{j-1} is the numerator of F_{j-1}, and D_{j-1} is its denominator. Thanks to how `J-VI()` (SubProcedure 6) is designed, at this point $J.f$ still stores the energy-levels as they were just *before* the last invocation of `J-VI()` made at line 11 of Algorithm 15; instead, the new energy-levels, those lifted-up during that same `J-VI()`, are stored into L_f. So, at this point, it will turn out that $\forall^{u\in V}\, J.f[u] = f^*_{i,j-1}(u)$.

SubProcedure 4: Set Values and Optimal Strategy

```
Procedure set_vars($\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma^*_0, i, j, F, J, \Gamma$)
    input : Winning sets $\mathcal{W}_0, \mathcal{W}_1$, values $\nu$, opt. strategy $\sigma^*_0$, $i \in [W^-, W^+]$, $j \in [1, s-1]$, ref. $F$ to
            $\mathcal{F}_{|V|}$, MPG $\Gamma$
1   $D \leftarrow$ denominator of $F[j-1]$;
2   while $L_\top \neq \emptyset$ do
3       $u \leftarrow$ pop_front($L_\top$);
4       $\nu(u) \leftarrow i + F[j-1]$;
5       if $\nu(u) \geq 0$ then $\mathcal{W}_0 \leftarrow \mathcal{W}_0 \cup \{v\}$; else $\mathcal{W}_1 \leftarrow \mathcal{W}_1 \cup \{v\}$;
6       if $u \in V_0$ then
7           for $v \in N^{out}_\Gamma(u)$ do
8               if $(D \cdot J.f[u]) \succeq (D \cdot J.f[v]) \ominus$ get_scl_w$(w(u,v), i, j-1, F)$ then
9                   $\sigma^*_0(u) \leftarrow v$; break;
```

This actually concludes the description of `set_vars()` (SubProcedure 4).

Indeed, the role of L_f is precisely that to allow the `J-VI()` to lift-up the energy-levels during the (i,j)-th Scan-Phase, meanwhile preserving (inside $J.f$) those computed at the $(i,j-1)$-th one (because `set_vars()` needs them in order to rely on Theorem 6.4). As mentioned, when `set_vars()` halts, all the energy-levels are scaled back, from $\mathbb{N}$ to $\mathbf{Q}$, and stored back from L_f into $J.f$ (at line 13 of Algorithm 15, see `scl_back_`f`()` in SubProcedure 5).

We remark at this point that all the arithmetics of Algorithm 15 can be done in $\mathbb{Z}$.

Now, let us detail the remaining subprocedures, those governing the Jumps and those concerning the energy-levels and the `J-VI()`. Since the details of the former rely significantly on those of the latter two, we proceed by discussing firstly how the energy-levels are handled by the `J-VI()` (see SubProcedure 6 and 5).

J-Value-Iteration. `J-VI()` is similar to the Value-Iteration of [14]. Still, there are some distinctive features. The `J-VI`$()$ takes in input two indices $i \in [W^-, W^+]$ and $j \in [1, s-1]$, a reference F to $\mathcal{F}_{|V|}$, (a reference to) the Jumper J, (a reference to) the input arena Γ. Basically, `J-VI`(i, j, F, J, Γ) aims at computing the least-SEPM of the reweighted EG $\Gamma_{i,j}$. For this, it relies on a (slightly revisited) *energy-lifting* operator $\delta : [V \to \mathcal{C}_\Gamma] \times V \to [V \to \mathcal{C}_\Gamma]$. The array-list employed to keep track of the inconsistent vertices is L^{inc}. It is assumed, as a pre-condition, that L^{inc} is already initialized when `J-VI`$()$ starts. We will show that this pre-condition holds thanks to how $L^{\text{inc}}_{\text{nxt}}$ is managed. Recall, Algorithm 15 is going to perform a sequence of invocations to `J-VI`$()$. During the execution of any such invocation of `J-VI`$()$, the role of $L^{\text{inc}}_{\text{nxt}}$ is precisely that of collecting, in advance, the initial list of inconsistent vertices for the *next*[1] `J-VI`$()$. Rephrasing, the k-th invocation of `J-VI`$()$ takes care of initializing L^{inc} for the $k+1$-th invocation of `J-VI`$()$, and this is done thanks to $L^{\text{inc}}_{\text{nxt}}$.

Also, the energy-levels are managed in a special way. The *inital* energy-levels are stored inside $J.f$ (as a pre-condition). Again, the k-th invocation of `J-VI`$()$ takes care of initializing the initial energy-levels for the $k+1$-th one: actually, those computed at the end of the k-th `J-VI`$()$ will become the initial energy-levels for the $k+1$-th one (subject to a rescaling). In this way, Algorithm 15 will succeed at amortizing the cost of all invocations of `J-VI`$()$. As mentioned, since $J.f$ stores rational-scalings, and $\Gamma_{i,j}$ is weighted in $\mathbb{Z}$, the `J-VI`$()$ needs to scale everything up, from $\mathbf{Q}$ to $\mathbb{N}$, when it reads the energy-levels out from $J.f$. So, $J.f$ is accessed *read-only* during the `J-VI`$()$: we want to update the energy-levels by applying δ, but still we need a back-up copy of the initial energy-levels (because they are needed at line 8 of `set_vars`$()$, SubProcedure 4). Therefore, a special subprocedure is employed for accessing energy-levels during `J-VI`$()$, it is named `get_scl_`$f()$ (SubProcedure 5); moreover, an array-list L_f is employed, whose aim is that to store the current energy-levels, those lifted-up during the `J-VI`$()$. SubProcedure 5 shows `get_scl_`$f()$, it takes: $u \in V$, some $j \in [1, s-1]$, a reference F to $\mathcal{F}_{|V|}$, and (a reference to) J.

`get_scl_`$f()$ goes as follows. If $L_f[u] = \perp$ (line 1), the denominator D of F_j is taken (line 2), and $f \leftarrow \lceil D \cdot J.f[u] \rceil$ is computed (line 3); a (new) entry (v, f) is inserted into L_f (line 4). Finally, in any case, $L_f[v]$ is returned (line 5).

[1] i.e., the subsequent invocation (in the above mentioned sequence of `J-VI`$()$) that will be performed, either at line 12 of `solve_MPG`$()$ (Algorithm 15), or at line 3 of `ua-jumps`$()$ (SubProcedure 9).

SubProcedure 5: Energy-Levels

```
SubProcedure get_scl_f$(v,j,F,J)$
    input: $v \in V$, $j \in [1,s-1]$,
        $F$ is a ref. to $\mathcal{F}_{|V|}$, $J$ is Jumper.
1   if $L_f[v] = \bot$ then
2       $D \leftarrow$ denominator of $F[j]$;
3       $f \leftarrow \lceil D \cdot J.f[v] \rceil$;
4       insert$\big((v,f),L_f\big)$;
5   return $L_f[v]$;
SubProcedure scl_back_f$(j,F,J)$
    input: $j \in [0,s-1]$, $F$ is a ref. to Farey's terms,
        $J$ is Jumper.
1   $D \leftarrow$ denominator of $F[j]$;
2   while $L_f \neq \emptyset$ do
3       $(v,f) \leftarrow$ pop_front$(L_f)$;
4       $J.f[v] \leftarrow f/D$;
```

As mentioned, at line 13 of Algorithm 15, $J.f$ will be overwritten by scaling back the values that are stored in L_f. This is done by `scl_back_f()` (SubProcedure 5): at line 1, D is the denominator of F_j; then, L_f is emptied, one element at a time (line 2); for each $(v,f) \in L_f$ (line 3), the rational f/D is stored back to $J.f[v]$ (line 4). This concludes `scl_back_f()`.

Next, `J-VI()` takes in input: $i \in [W^-,W^+]$, $j \in [1,s-1]$, a reference F to $\mathcal{F}_{|V|}$, (a reference to) the Jumper J, and (a reference to) the input MPG Γ. At line 1, `J-VI()` enters into a `while` loop which lasts while $L^{\text{inc}} \neq \emptyset$. The front vertex $v \leftarrow$ `pop_front`(L^{inc}) is popped from L^{inc} (line 2). Next, the energy-lifting operator δ is applied to v by invoking `apply_`$\delta(v,i,j,F,J,\Gamma)$ (line 3).

There inside (at line 1 of `apply_`$\delta()$), the energy-level of v is lifted-up as follows:

$$f_v \leftarrow \begin{cases} \min\big\{\texttt{get\_scl\_}f\big(v',j,F,J\big) \ominus \texttt{get\_scl\_}w\big(w(v,v'),i,j,F\big) \mid v' \in N_\Gamma^{\text{out}}(v)\big\}, & \text{if } v \in V_0; \\ \max\big\{\texttt{get\_scl\_}f\big(v',j,F,J\big) \ominus \texttt{get\_scl\_}w\big(w(v,v'),i,j,F\big) \mid v' \in N_\Gamma^{\text{out}}(v)\big\}, & \text{if } v \in V_1. \end{cases}$$

Then, f_v is stored inside L_f (notice, not in $J.f$), where it is binded to the key v (line 2). The control turns back to `J-VI()`. The current energy-level of v is retrieved by $f_v \leftarrow$ `get_scl_`$f(v,j,F,J)$ (line 4). If $f_v \neq \top$ (line 5), then v is inserted into $L^{\text{inc}}_{\text{nxt}}$ (if it isn't already in there) (line 6); moreover, if $v \in V_0$, then $J.$`cnt`$[v]$ and $\{J.$`cmp`$[(v,v')] \mid v' \in N_\Gamma^{\text{out}}(v)\}$ are recalculated from scratch, by invoking `init_cnt_cmp`(v,i,j,F,J,Γ) (line 7, see SubProcedure 7). Else, if $f_v = \top$ (line 8), then v is stored into $L_\top$ (line 9); and if $L^{\text{inc}}_{\text{nxt}}[v] \neq \bot$ in addition, then v is removed from $L^{\text{inc}}_{\text{nxt}}$ (line 10).

At this point it is worth introducing the following notation concerning energy-levels.

Definition 7.2. *For any step of execution ι and for any variable x of Algorithm 15, the state of x at step ι is denoted by x^ι. Then, the* current energy-levels *at step ι are* *defined as follows:*

$$\forall^{v \in V} f^{c:\iota}(v) \triangleq \begin{cases} L^{\iota}_f[v], & \textit{if } L^{\iota}_f[v] \neq \bot; \\ \lceil D_{j^{\iota}} \cdot J.f^{\iota}[v] \rceil, & \textit{otherwise.} \end{cases}$$

where D_{j^ι} is the denominator of F_{j^ι}. If ι is implicit, the current energy-levels *are denoted by f^c.*

SubProcedure 6: J-Value-Iteration

```
Procedure J-VI($i,j,F,J,\Gamma$)
    input : $i \in [W^-, W^+]$ and $j \in [1, s-1]$, $F$ is a ref. to Farey's terms, $J$ is Jumper, $\Gamma$ is an MPG.
1   while $L^{inc} \neq \emptyset$ do
2       $v \leftarrow$ pop_front($L^{\text{inc}}$);
3       apply_$\delta$($v,i,j,F,J,\Gamma$);
4       $f_v \leftarrow$ get_scl_$f$($v,j,F,J$);
5       if $f_v \neq \top$ then
6           if $L^{inc}_{nxt}[v] = \bot$ then insert($v, L^{\text{inc}}_{\text{nxt}}$);
7           if $v \in V_0$ then init_cnt_cmp($v,i,j,F,J,\Gamma$);
8       else
9           insert($v, L_\top$);
10          if $L^{inc}_{nxt}[v] \neq \bot$ then remove($v, L^{\text{inc}}_{\text{nxt}}$);
11      foreach $u \in N^{in}_\Gamma(v)$ do
12          $f_u \leftarrow$ get_scl_$f$($u,j,F,J$);
13          $\Delta_{u,v} \leftarrow f_v \ominus$ get_scl_$w$($w(u,v),i,j,F$);
14          if $L^{inc}[u] = \bot$ and $f_u < \Delta_{u,v}$ then
15              if $u \in V_0$ and $J.cmp[(u,v)] =$ T then
16                  $J$.cnt[$u$] $\leftarrow J$.cnt[$u$] $- 1$;
17                  $J$.cmp[$(u,v)$] $\leftarrow$ F;
18              if $u \in V_1$ OR $J.cnt[u] = 0$ then insert($u, L^{\text{inc}}$);
19  swap($L^{\text{inc}}, L^{\text{inc}}_{\text{nxt}}$);

SubProcedure apply_$\delta$($v,i,j,F,J,\Gamma$)
    input : $v \in V$, $i \in [W^-, W^+]$, $j \in [1, s-1]$, $F$ is a ref. to Farey, $J$ is Jumper, $\Gamma$ is an MPG.
1   $f_v \leftarrow \begin{cases} \min\{\text{get\_scl\_}f(v',j,F,J) \ominus \text{get\_scl\_}w(w(v,v'),i,j,F) \mid v' \in N^{\text{out}}_\Gamma(v)\}, & \text{if } v \in V_0; \\ \max\{\text{get\_scl\_}f(v',j,F,J) \ominus \text{get\_scl\_}w(w(v,v'),i,j,F) \mid v' \in N^{\text{out}}_\Gamma(v)\}, & \text{if } v \in V_1. \end{cases}$
2   insert($(v, f_v), L_f$);
```

Remark 7.2. *Recall, the role of L^{inc}_{nxt} and that of the* `swap`() *(line 19) is precisely that of initializing, in advance, the list of inconsistent vertices L^{inc} for the* next `J-VI`(); *because the* `J-VI`() *assumes a correct initialization of L^{inc} as a pre-condition.*

We argue in Proposition 7.2 and Lemma 7.1 that, when `J-VI`() *halts –say at step h– it is necessary to initialize $J.L^{inc}$ for the* next `J-VI`() *by including (at least) all the $v \in V$ such that: $0 < f^{c:h}(v) \neq \top$.*

Notice, if $L^{\text{inc}}_{\text{nxt}} = \emptyset$ holds just before the `swap()` at line 19, then $L^{\text{inc}} = \emptyset$ holds soon after; therefore, in that case yet another *EI-Jump* will occur (at line 7 of Algorithm 15) and eventually some other vertices will be inserted into L^{inc} (see the details of SubProcedure 8). We shall provide the details of `init_cnt_inc`(v,i,j,F,J) (line 7) hereafter. But let us first discuss the role played by J.`cnt` and J.`cmp` during `J-VI()`.

SubProcedure 7: Counters and Cmp Flags

```
SubProcedure $init\_cnt\_cmp(u,i,j,F,J,\Gamma)$
    input : $u \in V_0$, $i \in [W^-,W^+]$, $j \in [1,s-1]$, $F$ is a ref. to Farey, $J$ is Jumper, $\Gamma$ is an MPG.
    $c_u \leftarrow 0$;
    foreach $v \in N^{out}_\Gamma(u)$ do
        $f_u \leftarrow \texttt{get\_scl\_}f(u,j,F,J)$;
        $f_v \leftarrow \texttt{get\_scl\_}f(v,j,F,J)$;
        if $f_u \succeq f_v \ominus get\_scl\_w(w(u,v),i,j,F)$ then
            $c_u \leftarrow c_u + 1$;
            $J.\texttt{cmp}[(u,v)] \leftarrow \texttt{T}$;
        else $J.\texttt{cmp}[(u,v)] \leftarrow \texttt{F}$;
    $J.\texttt{cnt}[u] \leftarrow c_u$;
```

From line 11 to line 18, `J-VI()` explores $N^{\text{in}}_\Gamma(v)$ in order to find all the $u \in N^{\text{in}}_\Gamma(v)$ that may have become inconsistent soon after the energy-lifting δ that was applied to v (before, at line 3). For each $u \in N^{\text{in}}_\Gamma(v)$ (line 11), the energy-level $f_u \leftarrow \texttt{get\_scl\_}f(u,j,F,J)$ is considered (line 12), also, $\Delta_{u,v} \leftarrow f_v \ominus w'_{i,j}(u,v)$ is computed (line 13), where $f_v \leftarrow \texttt{get\_scl\_}f(v,j,F,J)$; if $f_u < \Delta_{u,v}$ (i.e., in case (u,v) is now incompatible w.r.t. f^c in $\Gamma_{i,j}$) *and* $L^{\text{inc}}[u] = \perp$ holds (line 14), then:

– If $u \in V_0$ *and* (u,v) was not already incompatible *before* (i.e., if $J.\texttt{cmp}[(u,v)] = \texttt{T}$ at line 15, then: $J.\texttt{cnt}[u]$ is decremented (line 16), and $J.\texttt{cmp}[(u,v)] \leftarrow \texttt{F}$ is assigned (line 17). (This is the role of the $J.\texttt{cnt}$ and $J.\texttt{cmp}$ flags).

– After that, if $u \in V_1$ *or* $J.\texttt{cnt}[u] = 0$, then u is inserted into L^{inc} (line 18).

When the `while` loop (at line 1) ends, the (references to) L^{inc} and $L^{\text{inc}}_{\text{nxt}}$ are *swapped* (line 19) (one is assigned to reference the other and vice-versa, in $O(1)$ time by interchanging pointers).

The details of $\texttt{init\_cnt\_cmp}(u,i,j,F,J,\Gamma)$ (line 7), where $u \in V_0$, are given in SubProcedure 7. At line 1, $c_u \leftarrow 0$ is initialized. For each $v \in N^{\text{out}}_\Gamma(u)$ (line 2), it is checked whether (u,v) is compatible with respect to the current energy-levels; i.e., whether or not $f_u \succeq f_v \ominus w'_{i,j}(u,v)$, holds for $f_u \leftarrow \texttt{get\_scl\_}f(u,j,F,J) = f^c(u)$ and $f_v \leftarrow \texttt{get\_scl\_}f(v,j,F,J) = f^c(v)$ (lines 3-5); if (u,v) is found to be compatible, then c_u is incremented (line 6) and $J.\texttt{cmp}[(u,v)] \leftarrow \texttt{T}$ is assigned (line 7); otherwise, (c_u stands still and) it is set $J.\texttt{cmp}[(u,v)] \leftarrow \texttt{F}$ (line 8). At the very end, it is finally set $J.\texttt{cnt}[u] \leftarrow c_u$ (line 9).

Concerning $J.\texttt{cmp}$ and $J.\texttt{cnt}$, it is now worth defining a formal notion of *coherency*.

Definition 7.3. *Let ι be any step of execution of Algorithm 15. Let $i \in [W^-,W^+]$, $j \in [0,s-1]$, $u \in V_0$ and $v \in N^{out}_\Gamma(u)$. We say that $J.cmp^\iota[(u,v)]$ is* coherent *w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$ when it holds:*

$$J.cmp^\iota[(u,v)] = \texttt{T} \text{ iff } f^{c:\iota}(u) \succeq f^{c:\iota}(v) \ominus w'_{i,j}(u,v).$$

Also, we say that $J.cnt^\iota[u]$ is coherent *w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$ when:*

$$J.cnt^\iota[u] = \big|\{(u,v) \in E \mid f^{c:\iota}(u) \succeq f^{c:\iota}(v) \ominus w'_{i,j}(u,v)\}\big|.$$

We say that $J.\texttt{cmp}^\iota$ is coherent *when* $\forall\,(u \in V_0 \setminus L^{inc^\iota})\,\forall\,(v \in N^{out}_\Gamma(u))$ $J.\texttt{cmp}^\iota[(u,v)]$ *is coherent;*

and we say that $J.\texttt{cnt}^\iota$ is coherent *when* $\forall(u \in V_0 \setminus L^{inc^\iota})$ $J.\texttt{cnt}^\iota[u]$ *is coherent.*

Finally, when something is not coherent, it is incoherent. *Remark: the step ι can be implicit.*

Remark 7.3. *In the Value-Iteration [14], the consistency checking of $(u,v) \in E$ (line 14) is explicit: an inequality like "$f(u) \succeq f(v) \ominus w(u,v)$" is tested; thus, neither the* `cmp` *flags nor an explicit notion of coherency are needed. So, why we introduced* `cmp` *flags and coherency? Observe, at line 14 of* `J-VI`$()$*, it doesn't make much sense to check "$f(u) \succeq f(v) \ominus w(u,v)$" in our setting. Consider the following facts: (1) of course the values of $w'_{i,j}$ depend on the index (i,j) of the current Scan-Phase; (2) therefore, going from one Scan-Phase to the next one (possibly, by Jumping), some counters may become incoherent, because $w_{i',j'} < w_{i,j}$ if $(i',j') > (i,j)$; but in the Value-Iteration [14] the only possible source of incoherency was the application of $\delta(\cdot,v)$; in Algorithm 15, going from one Scan-Phase to the next, we have an additional source of incoherency. (3) still,* `J-VI`$()$ *can't afford to re-initialize* `cnt` $: V \to \mathbb{N}$ *each time that it is needed, as this would cost $\Omega(|E|)$. So, if $(u,v) \in E$ is found incompatible (at line 14 of* `J-VI`$()$*) after the application of $\delta(\cdot,v)$ (line 3), how do we know whether or not (u,v) was already incompatible* before *the (last) application of $\delta(\cdot,v)$? We suggest to adopt the* `cmp` *flags, one bit per arc is enough.*

To show correctness and complexity, we firstly assume that whenever `J-VI`(i,j,F,J,Γ) is invoked the following three pre-conditions are satisfied:

(PC-1) $L_f = \emptyset$ and $\forall^{v\in V} f^c(v) \preceq f^*_{w'_{i,j}}(v)$;

(PC-2) $L^{\text{inc}} = \text{Inc}(f^c, i, j)$;

(PC-3) $J.$`cnt` and $J.$`cmp` are *coherent* w.r.t. f^c in $\Gamma_{i,j}$.

After having described the internals of the EI-Jumps, we'll show how to ensure (a slightly weaker, but still sufficient formulation of) (PC-1), (PC-2), (PC-3).

Assuming the pre-conditions, similar arguments as in [[14], Theorem 4] show that `J-VI`$()$ computes the least-SEPM of the EG $\Gamma_{i,j}$ in time $O(|V|^2|E|W)$ and linear space.

Proposition 7.1. *Assume that* `J-VI`$()$ *is invoked on input (i,j,F,J,Γ), and that (PC-1), (PC-2), (PC-3) hold at invocation time. Then,* `J-VI`$()$ *halts within the following time bound:*

$$\Theta\Big(\sum_{v\in V} deg_\Gamma(v) \cdot \ell^1_{\Gamma_{i,j}}(v)\Big) = O(|V|^2|E|W),$$

where $0 \leq \ell^1_{\Gamma_{i,j}}(v) \leq (|V|-1)|V|W$ is the number of times that the energy-lifting operator δ is applied to any $v \in V$, at line 3 of `J-VI`$()$ *on input (i,j,F,J,Γ). The working space is $\Theta(|V|+|E|)$.*

When `J-VI`$()$ *halts, f^c coincides with the* least-*SEPM of the reweighted EG $\Gamma_{i,j}$.*

Proof. The argument is very similar to that of [[14], Theorem 4], but there are some subtle differences between the `J-VI`() and the Value-Iteration of Brim, *et al.*:

(1) `J-VI`() employs $J.f$ and L_f to manage the energy-levels; however, one can safely argue by always referring to the current energy-levels f^{c}.

(2) `J-VI`() has no initialization phase; however, notice that the pre-conditions (PC-1), (PC-2), (PC-3) ensure a correct initialization of it.

(3) `J-VI`() employs $J.$`cmp` in order to test the consistency state of the arcs (see line 15 and 17 of `J-VI`()); but it is easy to see that, assuming (PC-3), this is a correct way to go.

Let us provide a sketch of the proof of correctness. As already observed in [[14], Lemma 7], the energy-lifting operator δ is $\sqsubseteq$*-monotone* (i.e., $\delta(f,v) \sqsubseteq \delta(g,v)$ for all $f \sqsubseteq g$). Next, the following invariant is maintained by `J-VI`() (Subprocedure 6) at line 1.

Inv-JVI. $\forall$(iteration ι of line 1 of `J-VI`(i,j,F,J,Γ)) $\forall(u \in V \setminus J.L^{\text{inc}^\iota})$ $\forall(v \in N^{\text{out}}_\Gamma(u))$:

(*i*) $\delta(f^{\text{c}:\iota},u) = f^{\text{c}:\iota}$;

(*ii*) if $u \in V_0 \setminus J.L^{\text{inc}^\iota}$, then $J.$`cnt`$^\iota[u]$ and $J.$`cmp`$^\iota[(u,v)]$ are both coherent w.r.t. $f^{\text{c}:\iota}$ in $\Gamma_{i,j}$.

It is not difficult to prove that *Inv-JVI* holds. The argument is almost the same as in [[14], Lemma 8]; the only noticeable variations are: (a) the `J-VI`() employs $J.$`cmp` in order to flag the compatibility status of the arcs; (b) the reference energy-level is f^{c}; (c) at the first iteration of line 1 of `J-VI`(), the *Inv-JVI* holds thanks to (PC-2) and (PC-3).

Termination is enforced by three facts: (i) every application of the energy-lifting operator (line 3) strictly increases the energy-level of one vertex v; (ii) the co-domain of SEPMs is finite.

Correctness follows by applying the Knaster-Tarski's fixed point theorem [111]. Indeed, at halting time, since δ is $\sqsubseteq$-monotone, and since (PC-1) and *Inv-JVI* hold, then we can apply Knaster-Tarski's fixed point theorem [111] to conclude that, when `J-VI`() halts at step h (say), then $f^{\text{c}:h}$ is the unique least fixpoint of (simultaneously) all operators $\delta(\cdot,v)$ for all $v \in V$, i.e., $f^{\text{c}:h}$ is the least-SEPM of the EG $\Gamma_{i,j}$.

So, when `J-VI`() halts, it holds that $\forall^{v\in V} f^{\text{c}:h}(v) = f^*_{w'_{i,j}}(v)$.

Concerning the time and space complexity, $\delta(\cdot,v)$ can be computed in time $O(|N^{\text{out}}_\Gamma(v)|)$ (line 3) (see `apply_`δ() in SubProcedure 6); the updating of $J.$`cnt` and $J.$`cmp`, which is performed by `init_cnt_cmp`() (line 7), also takes $O(|N^{\text{out}}_\Gamma(v)|)$ time. Soon after that $\delta(\cdot,v)$ has been applied to $v \in V$ (line 3), the whole $N^{\text{in}}_\Gamma(v)$ is explored for repairing incoherencies and for finding new inconsistent vertices (which is done from line 11 to line 18): this process takes $O(|N^{\text{in}}_\Gamma(v)|)$ time. Therefore, if $\delta(\cdot,v)$ is applied $\ell^1_{\Gamma_{i,j}}(v)$ times to (any) $v \in V$ during the `J-VI`(i,j,F,J,Γ), the total time is $\Theta\big(\sum_{v\in V} \texttt{deg}_\Gamma(v) \cdot \ell^1_{\Gamma_{i,j}}(v)\big)$. The codomain of any SEPM of $\Gamma_{i,j}$ is at most $(|V|-1)W'$, for $W' = D_j W \leq |V|W$, where the additional factor $D_j \leq |V|$ comes from the scaled weights of $\Gamma_{i,j}$; thus,

$\forall^{v \in V} 0 \leq \ell^1_{\Gamma_{i,j}}(v) \leq (|V|-1)D_j W \leq (|V|-1)|V|W$. As already mentioned in Section 6.3, the Farey's term $F[j]$ can be computed at the beginning of `J-VI()` in $O(1)$ time and space, from $F[j-1]$ and $F[j-2]$. Since $\sum_{v \in V} \texttt{deg}_\Gamma(v) = 2|E|$, the running time is also $O(|V|^2|E|W)$. We check that `J-VI()` works with $\Theta(|V|+|E|)$ space: L^{inc}, $L^{\text{inc}}_{\text{nxt}}$, L_f, and $L_\top$ contain no duplicates, so they take $\Theta(|V|)$ space; the size of $J.f$ and $J.$`cnt` is $|V|$, that of $J.$`cmp` plus L_ω is $\Theta(|E|)$. □

Indeed, the `J-VI()` keeps track of two additional array-lists, $L^{\text{inc}}_{\text{nxt}}$ and $L_\perp$. The role of $L^{\text{inc}}_{\text{nxt}}$ is to ensure (a slightly weaker formulation of) (PC-2): during the execution of Algorithm 15, the $\texttt{prev}_{\rho^J}(i,j)$-th invocation of `J-VI()` handles $L^{\text{inc}}_{\text{nxt}}$ so to ensure that (a slightly weaker, but still sufficient form of) (PC-2) holds for the (i,j)-th invocation. However, the way in which this happens also relies on the internals of the EI-Jumps. Also, the EI-Jumps take care of repairing $J.$`cnt` and $J.$`cmp` so to ensure (a weaker) (PC-3). The weaker formulation of (PC-2), (PC-3) is discussed in SubSection 7.3.2. From this perspective, the functioning of `J-VI()` and that of the EI-Jumps is quite braided. In order to detail these aspects, we need to observe the following fact.

Proposition 7.2. *Let $i \in [W^-, W^+]$ and $j \in [1, s-1]$. Assume that `J-VI`(i,j,F,J,Γ) is invoked at some step ι, suppose that $J.L^{inc\,\iota}_{nxt} = \emptyset$, and that (PC-1), (PC-2), (PC-3) hold at step ι. Then, the following two facts hold:*

1. *At each step $\hat{\iota} \geq \iota$ of `J-VI()` that is done* before *the `swap()` at line 19, it holds that: $J.L^{inc\,\hat{\iota}} \subseteq Inc(f^{c:\hat{\iota}}, i, j)$.*
2. *When `J-VI()` halts,* after *the `swap()` at line 19, say at step h, then:*

$$J.L^{inc^h} = \{v \in V \mid 0 < f^{c:h}(v) \neq \top\}.$$

Proof of (1) When `J-VI()` is invoked, Item 1 holds by (PC-2). Then, `J-VI()` can insert any $u \in V$ into L^{inc} only at line 18, when exploring $N^{\text{in}}_\Gamma(v)$ (from line 11 to line 18), for some $v \in V$. At line 18, $u \in V$ is inserted into L^{inc} *iff* $f_u < \Delta_{u,v}$ (line 14) and either $u \in V_1$ or $J.$`cnt`$[u] = 0$; i.e., *iff* u is inconsistent w.r.t. f^c in $\Gamma_{i,j}$ (indeed, $J.$`cnt` is coherent by (PC-3) and the fact that lines 15-17 of `J-VI()` preserve coherency). As $f^c(u)$ stands still while u is inside L^{inc}, and $f^c(v)$ for any $v \in N^{\text{out}}_\Gamma(u)$ can only increase during the `J-VI()`, then Item 1 holds. □

Proof of (2) Let us focus on the state of $L^{\text{inc}}_{\text{nxt}}$. Initially, $L^{\text{inc}}_{\text{nxt}} = \emptyset$ by hypothesis. During the `J-VI()`, $L^{\text{inc}}_{\text{nxt}}$ is modified only at line 6 or 10: some $v \in V$ is inserted into $L^{\text{inc}}_{\text{nxt}}$, say at step $\hat{\iota}$, (line 6) *iff* $f_v \neq \top$ (where f_v is the energy-level of v at the time of the insertion $\hat{\iota}$). We argue that $f_v > 0$ holds at $\hat{\iota}$ (line 6): since v was extracted from L^{inc} (line 2), and since all vertices in L^{inc} are inconsistent w.r.t. $f^{c:\hat{\iota}}$ in $\Gamma_{i,j}$ by Item 1, then $\delta(\cdot, v)$ had really increased $f^c(v)$ (at line 3); thus, it really holds $f_v > 0$ at $\hat{\iota}$. After the insertion, in case $f^c(v)$ becomes $\top$ at some subsequent execution of line 3, v is removed from $L^{\text{inc}}_{\text{nxt}}$ (and inserted into $L_\top$), see lines 8-10. Finally, at line 19 of `J-VI()`, $L^{\text{inc}}_{\text{nxt}}$ and L^{inc} are swapped (line 19). Therefore, at that point, Item 2 holds. □

When `J-VI()` halts, it is necessary to initialize L^{inc} for the *next* `J-VI()` by including all the $v \in V$ such that $0 < f^{\text{c}}(v) \neq \top$, because they are all inconsistent; this is shown by Lemma 7.1.

Lemma 7.1. *Let $i \in [W^-, W^+]$ and $j \in [1, s-1]$, where $s \triangleq |\mathcal{F}_{|V|}|$. Assume that `J-VI()` is invoked on input (i, j, F, J, Γ), and that all the pre-conditions (PC-1), (PC-2), (PC-3) are satisfied. Assume that `J-VI`(i, j, F, J, Γ) halts at step h. Let $i' \in [W^-, W^+]$ and $j' \in [1, s-1]$ be any two indices such that $(i', j') > (i, j)$. If $v \in V$ satisfies $0 < f^{\text{c}:h}(v) \neq \top$, then $v \in Inc(f^{\text{c}:h}, i', j')$.*

Proof. Let $\hat{v} \in V$ be any vertex such that $0 < f^{\text{c}:h}(\hat{v}) \neq \top$. By Proposition 7.1, $\forall^{v \in V} f^{\text{c}:h}(v) = f^*_{w'_{i,j}}(v)$. Being them monotonic, all operators $\{\delta(\cdot, v)\}_{v \in V}$ have least fixed point by Knaster-Tarski's theorem [111]. Since $f^*_{w'_{i,j}}(\hat{v})$ is the least-SEPM of $\Gamma_{i,j}$, then it is the unique least fixed point of simultaneously all operators $\{\delta(\cdot, v)\}_{v \in V}$; therefore, the following holds:

$$f^{\text{c}:h}(\hat{v}) = f^*_{w'_{i,j}}(\hat{v}) = \begin{cases} \min\{f^*_{w'_{i,j}}(v') \ominus w'_{i,j}(\hat{v}, v') \mid v' \in N^{\text{out}}_\Gamma(\hat{v})\}, & \text{if } \hat{v} \in V_0 \\ \max\{f^*_{w'_{i,j}}(v') \ominus w'_{i,j}(\hat{v}, v') \mid v' \in N^{\text{out}}_\Gamma(\hat{v})\}, & \text{if } \hat{v} \in V_1 \end{cases}$$

Since $0 < f^{\text{c}:h}(\hat{v}) \neq \top$, it is safe to discard the $\ominus$ operator in the equality above. Moreover, since $(i', j') > (i, j)$, then $w'_{i,j} > w'_{i',j'}$. Therefore, the following inequality holds:

$$\begin{aligned} f^{\text{c}:h}(\hat{v}) &= \begin{cases} \min\{f^{\text{c}:h}(v') - w'_{i,j}(\hat{v}, v') \mid v' \in N^{\text{out}}_\Gamma(\hat{v})\}, & \text{if } \hat{v} \in V_0 \\ \max\{f^{\text{c}:h}(v') - w'_{i,j}(\hat{v}, v') \mid v' \in N^{\text{out}}_\Gamma(\hat{v})\}, & \text{if } \hat{v} \in V_1 \end{cases} \\ &< \begin{cases} \min\{f^{\text{c}:h}(v') - w'_{i',j'}(\hat{v}, v') \mid v' \in N^{\text{out}}_\Gamma(\hat{v})\}, & \text{if } \hat{v} \in V_0 \\ \max\{f^{\text{c}:h}(v') - w'_{i',j'}(\hat{v}, v') \mid v' \in N^{\text{out}}_\Gamma(\hat{v})\}, & \text{if } \hat{v} \in V_1 \end{cases} \end{aligned}$$

So, restoring the $\ominus$ operator, we have:

$$f^{\text{c}:h}(\hat{v}) \prec \begin{cases} f^{\text{c}:h}(v') \ominus w'_{i',j'}(\hat{v}, v') \text{ for all } v' \in N^{\text{out}}_\Gamma(\hat{v}), & \text{if } \hat{v} \in V_0 \\ f^{\text{c}:h}(v') \ominus w'_{i',j'}(\hat{v}, v') \text{ for some } v' \in N^{\text{out}}_\Gamma(\hat{v}), & \text{if } \hat{v} \in V_1 \end{cases}$$

Therefore, $v \in \text{Inc}(f^{\text{c}:h}, i', j')$. □

Although, when the `prev`$_{\rho^J}(i, j)$-th `J-VI()` halts, it is correct –and necessary– to initialize L^{inc} for the (i, j)-th `J-VI()` by including all those $v \in V$ such that $0 < f^{\text{c}}(v) \neq \top$ (because they are all inconsistent w.r.t. to f^{c} in $\Gamma_{i,j}$ by Lemma 7.1), still, we observe that this is not sufficient. Indeed, consider the following two facts (I-1) and (I-2):

(I-1) It may be that, when the $\texttt{prev}_{\rho^J}(i,j)$-th J-VI() halts, it holds for all $v \in V$ that either $f^{c}(v) = 0$ or $f^{c}(v) = \top$. In that case, L^{inc} would be empty (if nothing more than what prescribed by Proposition 7.2 is done). We need to prevent this from happening, so to avoid vain Scan-Phases.

(I-2) When going, say, from the $(i-1,j)$-th to the $(i,1)$-th Scan-Phase, there might be some $(u,v) \in E$ such that: $f^{c}(u) = 0 = f^{c}(v)$ and $w(u,v) = i$; those (u,v) may become incompatible w.r.t. f^{c} in $\Gamma_{i,1}$ (because $i-1$ had been increased to i), possibly breaking the compatibility (and thus the coherency) of (u,v). These incompatible arcs are not taken into account by Proposition 7.2, nor by Lemma 7.1. Thus a special care is needed in order to handle them.

Energy-Increasing-Jumps. To resolve the issues raised in I-1 and I-2, the *EI-Jumps* will come into play. The pseudocode of the EI-Jumps is provided in SubProcedure 8. The ei-jump(i,J) really makes a jump only when $L^{\text{inc}} = \emptyset$ holds invocation. Basically, if $L^{\text{inc}} = \emptyset$ (line 1) we aim at avoiding *vain* Scan-Phases, i.e., (I-1); still, we need to take care of some additional (possibly) incompatible arcs, i.e., (I-2). Recall, L^{inc} is initialized by the J-VI() itself according to Proposition 7.2. Therefore, at line 1, $L^{\text{inc}} = \emptyset$ *iff* either $J.f(v) = 0$ *or* $J.f(v) = \top$ for every $v \in V$.

SubProcedure 8: EI-Jump

```
Procedure ei-jump($i$,$J$)
    input : Jumper $J$.
    output: T if an EI-Jump occurs; else, F.
1   if $L^{inc} = \emptyset$ then
2       $L^{\text{inc}} \leftarrow L^{\text{inc}}_{\text{cpy}}$; $L^{\text{inc}}_{\text{cpy}} \leftarrow \emptyset$;
3       $J.i \leftarrow i+1$;
4       if $L_\omega \neq \emptyset$ then
5           $(w,L_\alpha) \leftarrow$ read_front($L_\omega$);
6           if $w = J.i$ then
7               pop_front($L_\omega$);
8               repair($L_\alpha$,$J$);
9       while $L^{inc} = \emptyset$ and $L_\omega \neq \emptyset$ do
10          $(w,L_\alpha) \leftarrow$ pop_front($L_\omega$);
11          $J.i \leftarrow w$;
12          repair($L_\alpha$,$J$);
13      return T;
14  else return F;

SubProcedure repair($L_\alpha$,$J$)
    input : A list of arcs $L_\alpha$, reference to Jumper $J$.
1   foreach $(u,v) \in L_\alpha$ do
2       if $J.f[u] = 0$ and $J.f[v] = 0$ and $L^{inc}[u] = \bot$ then
3           if $u \in V_0$ then
4               $J$.cnt[$u$] $\leftarrow J$.cnt[$u$] $- 1$;
5               $J$.cmp[$(u,v)$] $\leftarrow$ F;
6               if $J$.cnt[$u$] $= 0$ then
7                   insert($u$,$L^{\text{inc}}$);
8           if $u \in V_1$ then insert($u$,$L^{\text{inc}}$);
```

To begin with, if $L^{\text{inc}} = \emptyset$ (line 1), copy $L^{\text{inc}} \leftarrow L^{\text{inc}}_{\text{cpy}}$, then, erase $L^{\text{inc}}_{\text{cpy}} \leftarrow \emptyset$ (line 2): this is related to the steps of backtracking that are performed by the UA-Jumps, we will give more details on this later on. Next, we increment i to $J.i \leftarrow i + 1$ (line 3). Then, if $L_\omega \neq \emptyset$ at line 4, we read (read-only) the front entry $(\hat{w}, L_{\hat{\alpha}})$ of L_ω (line 5); only if $\hat{w} = J.i$ (line 6), we pop $(\hat{w}, L_{\hat{\alpha}})$ out of L_ω (line 7), and we invoke `repair`$(L_{\hat{\alpha}}, J)$ (line 8) to repair the coherency state of all those arcs (i.e., all and only those in $L_{\hat{\alpha}}$) that we mentioned in (I-2). We will detail `repair()` shortly, now let us proceed with `ei-jump()`. At line 9, *while* $L^{\text{inc}} = \emptyset$ *and* $L_\omega \neq \emptyset$: the front $(\bar{w}, L_{\bar{\alpha}})$ is popped from L_ω (line 10) and $J.i \leftarrow \bar{w}$ is assigned (line 11). The ending-point of the EI-Jump will now reach $\bar{w}$ (at least). A moment's reflection reveals that, jumping up to $\bar{w}$, some arcs $(u,v) \in E$ such that $f^{c}(u) = 0 = f^{c}(v)$ (which were compatible w.r.t. the (i,j)-th Scan-Phase, just *before* the jump) may become incompatible for the $(\bar{w}, 1)$-th Scan-Phase (which is now candidate to happen), because $\bar{w} > i$. What are these new incompatible arcs? Since L_ω was sorted in increasing order, they're all *and only* those of weight $w(u,v) = \bar{w} = J.i$; i.e., those in the $L_{\bar{\alpha}}$ that is binded to $\bar{w}$ in L_ω. To repair coherency, `repair`$(L_{\bar{\alpha}}, J)$ (line 12) is invoked. This repeats until $L^{\text{inc}} \neq \emptyset$ *or* $L_\omega = \emptyset$. Then, `ei-jump()` returns `T` (at line 13).

If $L^{\text{inc}} \neq \emptyset$ at line 1, then `F` is returned (line 14); so, in that case, *no* EI-Jump will occur.

Let us detail the `repair`(L_α, J). On input (L_α, J), for each arc $(u,v) \in L_\alpha$ (line 1), if $J.f[u] = 0 = J.f[v]$ and $L^{\text{inc}}[u] = \perp$ (line 2), the following happens. If $u \in V_1$, then u is promptly inserted (in front of) L^{inc} (line 8); else, if $u \in V_0$, $J.$`cnt`$[u]$ is decremented by one unit (line 4); also, it is flagged $J.$`cmp`$[(u,v)] \leftarrow$ `F` (line 5). After that, if $J.$`cnt`$[u] = 0$ (line 6), then u is inserted in front of L^{inc} (line 7). The following proposition holds for the `ei-jump()` (SubProcedure 8).

Proposition 7.3. *The* `ei-jump()` *(SubProcedure 8) halts in finite time. The* total *time spent for all invocations of* `ei-jump()` *(that are made, at line 7, during the main* `while` *loop of Algorithm 15) is $\Theta(t_{\ell_7} + |E|)$, where t_{ℓ_7} is the total number of iterations of line 7 that are made by Algorithm 15. The* `ei-jump()` *works with $\Theta(|V| + |E|)$ space.*

Proof. The `for-each` loop in `repair()` is bounded: each arc (u,v) of L_α is visited exactly once, spending $O(1)$ time per each. The `while` loop in `ei-jump()` (lines 9-12) is also bounded: it consumes the elements (w, L_α) of L_ω, spending $O(|L_\alpha|)$ time per cycle. There are no other loops in `ei-jump()`, so it halts in finite time. Now, consider the following three facts: (i) `ei-jump()` is invoked by `solve_MPG()` (Algorithm 15) once per each iteration of the main `while` loop at line 7. Assume there are t_{ℓ_7} such iterations overall. (ii) either `ei-jump()` returns immediately or it visits k arcs $(u,v) \in E$ in time $\Theta(k)$, for some $1 \leq k \leq |E|$; (iii) each arc $(u,v) \in E$ is visited by `ei-jump()` at most once during the whole execution of Algorithm 15, because the elements of L_ω are consumed and there are no duplicates in there. Altogether, (i), (ii) and (iii) imply the $\Theta(t_{\ell_7} + |E|)$ total running time. Moreover, `ei-jump()` works with $\Theta(|V| + |E|)$ space. Indeed L^{inc} contains no duplicated vertices, so: $|L^{\text{inc}}| \leq |V|$,

$|L_\omega| = |E|$, the size of $J.f$ and that of J.`cnt` is $|V|$, and the size of J.`cmp` is $|E|$. □

The description of Algorithm 15 ends by detailing the UA-Jumps.

Unitary-Advance-Jumps. Recall, UA-Jumps are adopted so to scroll through $\mathcal{F}_{|V|}$ only *when* (and *where*) it is really necessary; that is only $|V|$ times at most, because each time at least one vertex will take a value. The pseudocode is shown in Fig. 9.

SubProcedure 9: UA-Jumps

```
SubProcedure ua-jumps($i,s,F,J,\Gamma$)
    input : $i \in [W^-, W^+]$, $s = |\mathcal{F}_{|V|}|$, $F$ is a ref. to $\mathcal{F}_{|V|}$, Jump $J$, input MPG $\Gamma$.
    repeat
        J-VI($i, s-1, F, J, \Gamma$); /* UA-Jump */
        if $L_\top = \emptyset$ then
            $i \leftarrow i + 1$;
            rejoin_ua-jump($i,s,F,J$);
    until $L_\top \neq \emptyset$
    $S \leftarrow$ backtrack_ua-jump($i,s,F,J,\Gamma$);
    return ($i,S$);
SubProcedure rejoin_ua-jump($i,s,F,J$)
    input : $i \in [W^-, W^+]$, $F$ is a ref. to $\mathcal{F}_{|V|}$, Jump $J$.
    scl_back_f($s-1, F, J$);
    if $L_\omega \neq \emptyset$ then
        $(w, L_\alpha) \leftarrow$ read_front($L_\omega$);
        if $w = i$ then
            pop_front($L_\omega$);
            repair($L_\alpha, J$); // see SubProc. 8
SubProcedure backtrack_ua-jump($i,s,F,J,\Gamma$)
    input : $i \in [W^-, W^+]$, $s = |\mathcal{F}_{|V|}|$, Jump $J$, MPG $\Gamma$.
    $L^{\text{inc}}_{\text{cpy}} \leftarrow L^{\text{inc}}$; $L^{\text{inc}} \leftarrow \emptyset$;
    $L_f[u] \leftarrow \begin{cases} \perp & \text{, if } u \in L_\top; \\ L_f[u] & \text{, if } u \in V \setminus L_\top. \end{cases}$
    scl_back_f($s-1, F, J$);
    $S \leftarrow L_\top$;
    while $L_\top \neq \emptyset$ do
        $u \leftarrow$ pop_front($L_\top$)
        if $u \in V_0$ then
            init_cnt_cmp($u, i, 1, F, J, \Gamma[S]$);
            if $J.cnt[u] = 0$ then
                insert($u, L^{\text{inc}}$);
        if $u \in V_1$ then
            foreach $v \in N^{out}_{\Gamma[S]}(u)$ do
                $f_u \leftarrow$ get_scl_f($u, 1, F, J$);
                $f_v \leftarrow$ get_scl_f($v, 1, F, J$);
                $w' \leftarrow$ get_scl_w($w(u,v), i, 1, F$);
                if $f_u \prec f_v \ominus w'$ then
                    insert($u, L^{\text{inc}}$); break;
    return $S$ ;
```

The UA-Jumps begin soon after that `ei-jump()` returns `T` at line 8 of Algorithm 15. The starting point of the UA-Jumps (i.e., the initial value of i) is provided by `ei-jump()` (line 7 of Algorithm 15): it is stored into $J.i$

and passed in input to `ua-jumps`$(J.i,s,F,J,\Gamma)$ (at line 9 of Algorithm 15). Starting from $i = J.i$, basically the `ua-jumps`$()$ repeats a sequence of invocations to `J-VI`$()$, on input $(i,s-1),(i+1,s-1),(i+2,s-1),\cdots,(\hat{\imath},s-1)$; until $L_\top \triangleq \mathcal{W}_0(\Gamma_{\hat{\imath}-1,s-1}) \cap \mathcal{W}_1(\Gamma_{\hat{\imath},s-1}) \neq \emptyset$ holds for some $\hat{\imath} \geq i$. When $L_\top \neq \emptyset$, the `ua-jumps`$()$ *backtracks* the Scan-Phases from the $(\hat{\imath},s-1)$-th to the $(\hat{\imath},1)$-th one, by invoking `backtrack_ua-jump`(i,s,F,J,Γ), and then it halts; soon after, Algorithm 15 will begin scrolling through $\mathcal{F}_{|V|}$ by invoking another sequence of `J-VI`$()$ (this time at line 11 of Algorithm 15) on input $(\hat{\imath},1),(\hat{\imath},2),(\hat{\imath},3),\ldots$ (which is controlled by the `while` loop at line 6 of Algorithm 15). More details concerning the UA-Jumps now follow.

So, `ua-jumps`$()$ (SubProcedure 9) performs a sequence of UA-Jumps (actually, at least one). The invocation to `J-VI`$(\hat{\imath},s-1,F,J,\Gamma)$ repeats for $\hat{\imath} \geq i$ (lines 1-2), until $L_\top \neq \emptyset$ (line 6). There, $L_\top$ contains all and only those $v \in V$ whose energy-level became $f(v) = \top$ during the last performed `J-VI`$()$ (line 2); so, at line 3, it is $L_\top = \mathcal{W}_0(\Gamma_{\hat{\imath}-1,s-1}) \cap \mathcal{W}_1(\Gamma_{\hat{\imath},s-1})$. At this point, if $L_\top = \emptyset$ (line 3), the procedure prepares itself to make another UA-Jump: $\hat{\imath} \leftarrow \hat{\imath}+1$ is set (line 4), and then `rejoin_ua-jump`$(\hat{\imath},s,F,J)$ is invoked (line 5). Else, if $L_\top \neq \emptyset$ (line 6), it is invoked `backtrack_ua-jump`$(\hat{\imath},s,F,J,\Gamma)$ (line 7), and then (i,S) is returned (line 8), where $S \triangleq L_\top = \mathcal{W}_0(\Gamma_{\hat{\imath}-1,s-1}) \cap \mathcal{W}_1(\Gamma_{\hat{\imath},s-1})$ was assigned at line 4 of `backtrack_ua-jump`$()$.

The `rejoin_ua-jump`(i,s,F,J) firstly copies the energy-levels stored in L_f back to $J.f$, by invoking `scl_back_`$f(s-1,F,J)$ (line 1). Secondly, at lines 4-6, by operating in the same way as `ei-jump`$()$ does (see lines 4-8 of `ei-jump`$()$, SubProcedure 8), it repairs the coherency state of $J.$`cnt` and $J.$`cmp` w.r.t. all those arcs $(u,v) \in E$ such that $w(u,v) = i$ and $J.f[u] = 0 = J.f[v]$.

Let us detail the `backtrack_ua-jump`$()$. Basically, it aims at preparing a correct state so to allow Algorithm 15 to step through $\mathcal{F}_{|V|}$. Stepping through $\mathcal{F}_{|V|}$ essentially means to execute a sequence of `J-VI`$()$ at line 11 of Algorithm 15, until $L^{\text{inc}} = \emptyset$. A moment's reflection reveals that this sequence of `J-VI`$()$ can run just on the sub-arena of Γ that is induced by $S \triangleq L_\top = \mathcal{W}_0(\Gamma_{i-1,s-1}) \cap \mathcal{W}_1(\Gamma_{i,s-1})$ (see line 4 of `backtrack_ua-jump`$()$); there is no real need to lift-up again (actually, slowly than before) all the energy-levels of the component induced by $V \setminus L_\top$: those energy-levels can all be confirmed now that the UA-Jumps are finishing, and they can all stand still while Algorithm 15 is stepping through $\mathcal{F}_{|V|}$ at line 11, until another EI-Jump occurs.

For this reason, `backtrack_ua-jump`$(\hat{\imath},s,F,J,\Gamma)$ works as follows.

Firstly, we copy $L^{\text{inc}}_{\text{cpy}} \leftarrow L^{\text{inc}}$, then we erase $L^{\text{inc}} \leftarrow \emptyset$ (line 1). This is sort of a back-up copy, notice that $L^{\text{inc}}_{\text{cpy}}$ will be restored back to L^{inc} at line 2 of `ei-jump`$()$ (SubProcedure 8): when Algorithm 15 will finish to step through $\mathcal{F}_{|V|}$, it will hold $L^{\text{inc}} = \emptyset$ at line 1 of `ei-jump`$()$ (SubProcedure 8), so at that point the state of L^{inc} will need to be restored by including (at least) all those vertices that are now assigned to $L^{\text{inc}}_{\text{cpy}}$ at line 1 of `backtrack_ua-jump`$()$. Next, all the energy-levels of $V \setminus L_\top$ are confirmed and saved back to $J.f$; this is done: (i) by setting,

$$L_f[u] \leftarrow \begin{cases} \bot & \text{, if } u \in L_\top; \\ L_f[u] & \text{, if } u \in V \setminus L_\top. \end{cases} \quad \text{(line 2)}$$

and (ii) by invoking `scl_back_`$f(s-1,F,J)$ (line 3). The energy-levels of all $v \in L_\top$ are thus restored as they were at the end of the $(\hat{\imath}-1,s-1)$-th invocation of `J-VI()` at line 2 of `ua-jumps()`. Next, it is assigned $S \leftarrow L_\top$ at line 4. Then, `backtrack_ua-jump()` takes care of preparing a correct state of L^{inc}, J.`cnt`, J.`cmp` for letting Algorithm 15 stepping through $\mathcal{F}_{|V|}$.

While $L_\top \neq \emptyset$ (line 5), we pop the front element of $L_\top$, i.e., $u \leftarrow$ `pop_front`$(L_\top)$ (line 6):

– If $u \in V_0$ (line 7), then we compute J.`cnt`$[u]$ and we also compute for every $v \in N^{\text{out}}_{\Gamma[S]}(u)$ a coherent J.`cmp`$[(u,v)]$ w.r.t. f^{c} in $\Gamma[S]_{\hat{\imath},1}$, by `init_cnt_cmp`$(u,\hat{\imath},1,F,J,\Gamma[S])$ (line 8); finally, if J.`cnt`$[u]=0$ (line 9), we insert u into L^{inc} (line 10).

– Else, if $u \in V_1$ (line 11), we explore $N^{\text{out}}_{\Gamma[S]}(u)$ looking for some incompatible arc (lines 12-17). For each $v \in N^{\text{out}}_{\Gamma[S]}(u)$ (line 12), if $f_u \prec f_v \ominus w'_{\hat{\imath},1}(u,v)$ (i.e., if (u,v) is incompatible w.r.t. f^{c} in $\Gamma_{\hat{\imath},1}$), where $f_u \leftarrow$ `get_scl_`$f(u,1,F,J)$ and $f_v \leftarrow$ `get_scl_`$f(v,1,F,J)$, then, we insert u into L^{inc} at line 17 (also breaking the `for-each` cycle).

This concludes the description of the UA-Jumps. Algorithm 15 is completed.

7.3.2 Correctness of Algorithm 15

This subsection presents the proof of correctness for Algorithm 15. It is organized as follows. Firstly, we show that `J-VI()` (SubProcedure 6) works fine even when assuming a relaxed form of the pre-conditions (PC-2) and (PC-3). Secondly, we identify an additional set of pre-conditions under which the `ei-jump()` (SubProcedure 8) is correct. Thirdly, we prove that under these pre-conditions `ua-jumps()` (SubProcedure 9) is also correct. Finally, we show that these pre-conditions are all satisfied during the execution of Algorithm 15, and that the latter is thus correct.

Correctness of `J-VI()` (SubProcedure 6)

To prove the correctness of `J-VI()`, the (PC-1), (PC-2), (PC-3) have been assumed in Lemma 7.1. It would be fine if they were met whenever Algorithm 16 invokes `J-VI()`. Unfortunately, (PC-2) and (PC-3) may not hold. Still, we shall observe that a weaker formulation of them, denoted by (w-PC-2) and (w-PC-3), really hold; and these will turn out to be enough for proving correctness.

Definition 7.4. *Let $i \in [W^-,W^+]$ and $j \in [1,s-1]$. Fix some step of execution ι of Algorithm 15.*

The pre-conditions (w-PC-2) and (w-PC-3) are defined at step ι as follows.

(w-PC-2) $L^{inc^\iota} \subseteq Inc(f^{c:\iota},i,j)$.

(w-PC-3) $\forall(u \in V \setminus L^{inc^\iota})\ \forall(v \in N^{out}_\Gamma(u))$:

If $u \in V_0$, the following three properties hold on J.`cnt`$^\iota$ and J.`cmp`$^\iota$:

1. *If $J.\texttt{cmp}^\iota[(u,v)] = \texttt{F}$, then (u,v) is incompatible w.r.t. f^c in $\Gamma_{i,j}$;*
2. *If $J.\texttt{cmp}^\iota[(u,v)] = \texttt{T}$ and (u,v) is incompatible w.r.t. f^c in $\Gamma_{i,j}$, then $v \in L^{inc^\iota}$.*
3. *$J.\texttt{cnt}^\iota[u] = \big|\{v \in N^{out}_\Gamma(u) \mid J.\texttt{cmp}^\iota[(u,v)] = \texttt{T}\}\big|$ and $J.\texttt{cnt}^\iota[u] > 0$.*

If $u \in V_1$, and $(u,v) \in E$ is incompatible w.r.t. f^c in $\Gamma_{i,j}$, then $v \in L^{inc^\iota}$.

If (w-PC-3) holds on $J.\texttt{cnt}^\iota$ and $J.\texttt{cmp}^\iota$, they are said weak-coherent *w.r.t. f^c in $\Gamma_{i,j}$.*

We will also need the following Lemma 7.2, it asserts that $\psi_\rho : (i,j) \to f^*_{i,j}$ is monotone non-decreasing; the proof already appears in [[38], Lemma 8, Item 1].

Lemma 7.2. *Let $i,i' \in [W^-,W^+]$ and $j,j' \in [1,s-1]$ be any two indices such that $(i,j) < (i',j')$.*
Then, $\forall^{v\in V} f^*_{i,j}(v) \preceq f^*_{i',j'}(v)$.

Proposition 7.4 shows that (PC-1), (w-PC-2), (w-PC-3) suffices for the correctness of $\texttt{J-VI}()$.

Proposition 7.4. *The $\texttt{J-VI}()$ (SubProcedure 6) is correct (i.e., Propositions 7.1 and 7.2 still hold) even if (PC-1), (w-PC-2), (w-PC-3) are assumed instead of (PC-1), (PC-2), (PC-3).*

In particular, suppose that $\texttt{J-VI}()$ is invoked on input (i,j,F,J,Γ), say at step ι, and that all of the pre-conditions (PC-1), (w-PC-2), (w-PC-3) hold at ι. When $\texttt{J-VI}(i,j,F,J,\Gamma)$ halts, say at step h, then all of the following four propositions hold:

1. *$f^{c:h}$ is the least-SEPM of the EG $\Gamma_{i,j}$;*
2. *$J.\texttt{cnt}^h$, $J.\texttt{cmp}^h$ are both coherent w.r.t. $f^{c:h}$ in $\Gamma_{i,j}$;*
3. *$L^{inc^h} = \{v \in V \mid 0 < f^{c:h}(v) \neq \top\}$;*
4. *$L^h_\top = V_{f^{c:\iota}} \cap V \setminus V_{f^{c:h}}$.*

Proof. Basically, we want to prove that Propositions 7.1 and 7.2 still hold.

Suppose $L^{\text{inc}^\iota} = \emptyset$. Let $u \in V_0$. By (w-PC-3) and $L^{\text{inc}^\iota} = \emptyset$, for every $v \in N^{\text{out}}_\Gamma(u)$, $J.\texttt{cmp}^\iota[(u,v)]$ is coherent w.r.t. f^ι in $\Gamma_{i,j}$; thus, $J.\texttt{cnt}^\iota[u]$ is also coherent w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$. Therefore, (PC-3) holds. Now, let $u \in V_1$. By (w-PC-3) and $L^{\text{inc}^\iota} = \emptyset$, for every $v \in N^{\text{out}}_\Gamma(u)$ it holds that (u,v) is compatible w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$; thus, u is consistent w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$. In addition, by (w-PC-3) again, $J.\texttt{cnt}^\iota[u] > 0$ holds for every $u \in V_0$. Therefore, every $u \in V$ is consistent w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$; so, (PC-2) holds. Since (PC-1,2,3) hold, then Propositions 7.1 and 7.2 hold.

Now, suppose $L^{\text{inc}^\iota} \neq \emptyset$. Since $J.\texttt{cnt}^\iota$ and $J.\texttt{cmp}^\iota$ may be incoherent – *at time ι* –, there might be some $\hat{u} \in V \setminus L^{\text{inc}^\iota}$ which is already inconsistent w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$ (i.e., even if $u \notin L^{\text{inc}^\iota}$).

Still, we claim that, during `J-VI()`'s execution (say at some steps ι', ι'', i.e., eventually), for every $u \in V_0$ and $v \in N^{\text{out}}_{\Gamma}(u)$, both $J.\texttt{cmp}^{\iota'}[(u,v)]$ and $J.\texttt{cnt}^{\iota''}[u]$ will *become* coherent (at ι', ι'' respectively); and we also claim that any $u \in V_1$ which was inconsistent at ι will be (eventually, say at step ι''') inserted into L^{inc}. Indeed, at that point (say, at $\hat{\iota} = \max\{\iota', \iota'', \iota'''\}$), *all* (and only those) $\hat{u} \in V$ that were already inconsistent at invocation time ι, or that became inconsistent during `J-VI()`'s execution (until step $\hat{\iota}$), they will be really inserted into L^{inc}.

To prove it, let $\hat{u} \in V \setminus L^{\text{inc}^{\iota}}$ and $\hat{v} \in N^{\text{out}}_{\Gamma}(\hat{u})$ be any two (fixed) vertices such that either:

$\hat{u} \in V_0$ and $J.\texttt{cmp}^{\iota}[(\hat{u},\hat{v})] = \texttt{F}$: Then, by (w-PC-3), $(\hat{u},\hat{v})$ is incompatible w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$.

$\hat{u} \in V_0$ and $J.\texttt{cmp}^{\iota}[(\hat{u},\hat{v})] = \texttt{T}$ but $(\hat{u},\hat{v})$ is incompatible w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$:

Then, by (w-PC-3), $\hat{v} \in L^{\text{inc}^{\iota}}$. Since `J-VI()` aims precisely at emptying L^{inc}, $\hat{v}$ is popped from $L^{\text{inc}^{\iota'}}$ (line 2 of SubProcedure 6) – say at some step ι' of `J-VI()`'s execution. Soon after that, $N^{\text{in}}_{\Gamma}(\hat{v})$ is explored (lines 11-18 of SubProcedure 6); so $\hat{u}$ is visited, then $(\hat{u},\hat{v})$ is found incompatible (i.e., $f_{\hat{u}} < \Delta_{\hat{u},\hat{v}}$ at line 14, after ι'). Since $\hat{u} \in V_0 \setminus L^{\text{inc}^{\iota'}}$, and $J.\texttt{cmp}^{\iota'}[(\hat{u},\hat{v})] = \texttt{T}$, then at some step $\iota'' > \iota'$ the counter $J.\texttt{cnt}^{\iota''}$ is decremented by one unit and therefore $J.\texttt{cmp}^{\iota''}[(u,v)] \leftarrow \texttt{F}$ is assigned (at lines 16-17). This proves that $J.\texttt{cmp}[(\hat{u},\hat{v})]$ becomes coherent eventually (i.e., at ι''). Now, given $\hat{u}$, the same argument holds for any other $v \in N^{\text{out}}_{\Gamma}(\hat{u})$; therefore, when $J.\texttt{cmp}[(\hat{u},v)]$ will finally become coherent for every $v \in N^{\text{out}}_{\Gamma}(\hat{u})$, then $J.\texttt{cnt}[\hat{u}]$ will be coherent as well by (w-PC-3). Thus, by (w-PC-3), coherency of both $J.\texttt{cnt}$ and $J.\texttt{cmp}$ holds eventually, say at $\hat{\iota}$. At that point, all $u \in V_0$ that were inconsistent at ι, or that have become inconsistent during the execution (up to $\hat{\iota}$), they necessarily have had to be inserted into L^{inc} (at line 18 of `J-VI()`, SubProcedure 6), because their (coherent) counter $J.\texttt{cnt}[u]$ must reach 0 (at $\hat{\iota}$), which allows `J-VI()` to recognize u as inconsistent at lines 14-18. Notice that the coherency of $J.\texttt{cnt}$ and $J.\texttt{cmp}$ is kept satisfied from $\hat{\iota}$ onwards: when some $v \in V$ is popped out of L^{inc} (line 2), then $J.\texttt{cnt}$ and $J.\texttt{cmp}$ are recalculated from scratch (line 7), and it is easy to check that `init_cnt_cmp()` (SubProcedure 7) is correct; then $J.\texttt{cnt}$, $J.\texttt{cmp}$ may be modified subsequently, at lines 16-17 (SubProcedure 6), but it's easy to check that lines 14-17 preserve coherency; so, coherency will be preserved until `J-VI()` halts.

$\hat{u} \in V_1$ and $(\hat{u},\hat{v})$ is incompatible w.r.t. $f^{c:\iota}$ in $\Gamma_{i,j}$:

Then, by (w-PC-3), $\hat{v} \in L^{\text{inc}^{\iota}}$. As before, since the `J-VI()` aims precisely at emptying L^{inc}, $\hat{v}$ is popped from L^{inc} (line 2 of SubProcedure 6); at some step of `J-VI()`'s execution. Soon after that, $N^{\text{in}}_{\Gamma}(\hat{v})$ is explored (lines 11-18 of SubProcedure 6). As soon as $\hat{u}$ is visited, $(\hat{u},\hat{v})$ is found incompatible (i.e., $f_{\hat{u}} < \Delta_{\hat{u},\hat{v}}$ at line 14). Since $\hat{u} \in V_1 \setminus L^{\text{inc}}$, then $\hat{u}$ is promptly inserted into L^{inc} (line 18). In this way, all those $u \in V_1$ that were inconsistent at the time of `J-VI()`'s invocation, or that become inconsistent during the execution, they necessarily have had to be inserted into L^{inc} (line 18 of SubProcedure 6).

This analysis is already sufficient for asserting that Proposition 7.1 holds, even assuming only (PC-1), (w-PC-3): indeed, the `Inv-JVI` invariant mentioned in its proof will hold, eventually, and then the Knaster-Tarski's fixed point theorem applies. This also proves Items (1) and (2).

Moreover, by (w-PC-2) and by arguments above, at each step $\bar{\iota}$ of `J-VI()`, if $v \in L^{\text{inc}^{\bar{\iota}}}$ then v is really inconsistent w.r.t. $f^{\text{c}:\bar{\iota}}$ in $\Gamma_{i,j}$, i.e., $L^{\text{inc}^{\bar{\iota}}} \subseteq \text{Inc}(f^{\text{c}:\bar{\iota}},i,j)$. Thus, every time that some v is popped from L^{inc} at line 2, then $\delta(f^{\text{c}},v)$ really increases $f^{\text{c}}(v)$ at line 3; therefore, $f^{\text{c}}(v) > 0$ holds whenever v is inserted into $L^{\text{inc}}_{\text{nxt}}$ at line 6 of `J-VI()` (SubProcedure 6); this implies that Proposition 7.2 holds, assuming (PC-1), (w-PC-2), (w-PC-3), and proves Item (3). To conclude, we show Item (4). Notice, $L_\top$ is modified only at line 9 of `J-VI()` (SubProcedure 6); in particular, some $v \in V$ is inserted into $L_\top$ at line 9, say at step $\hat{\iota}$, if and only if $f^{\text{c}:\hat{\iota}}(v) = \top$. Since the energy-levels can only increase during the execution of `J-VI()`, then $L^h_\top \subseteq V \setminus V_{f^{\text{c}:h}} = \{u \in V \mid f^{\text{c}:h}(u) = \top\}$. Since at each step $\bar{\iota}$ of `J-VI()` it holds $L^{\text{inc}^{\bar{\iota}}} \subseteq \text{Inc}(f^{\text{c}:\bar{\iota}},i,j)$, then whenever some $v \in V$ is inserted into $L_\top$ at line 9, it must be that $f^{\text{c}:\iota}(v) < \top$ where ι is the invocation time (otherwise, v would not have been inconsistent at step $\bar{\iota}$); thus, $L^h_\top \subseteq V_{f^{\text{c}:\iota}} = \{v \in V \mid f^{\text{c}:\iota}(v) < \top\}$. Therefore, $L^h_\top \subseteq V_{f^{\text{c}:\iota}} \cap V \setminus V_{f^{\text{c}:h}}$. Vice versa, let $v \in V_{f^{\text{c}:\iota}} \cap V \setminus V_{f^{\text{c}:h}}$; the only way in which `J-VI()` can increase the energy-level of v from step ι to step h is by applying $\delta(f^{\text{c}},v)$ at line 3; as soon as $f^{\text{c}}(v) = \top$ (and this will happen, eventually, since $v \in V_{f^{\text{c}:\iota}} \cap V \setminus V_{f^{\text{c}:h}}$), then v is inserted into $L_\top$ at line 9. Thus, $V_{f^{\text{c}:\iota}} \cap V \setminus V_{f^{\text{c}:h}} \subseteq L^h_\top$. Therefore, $L^h_\top = V_{f^{\text{c}:\iota}} \cap V \setminus V_{f^{\text{c}:h}}$; and this proves Item (4). □

Correctness of EI-Jump (SubProcedure 8)

To begin, it is worth asserting some preliminary properties of `ei-jump()` (SubProcedure 8).

Lemma 7.3. *Assume* `ei-jump`(i,J) *(SubProcedure 8) is invoked by Algorithm 15 at line 7, say at step ι, and for some $i \in [W^- - 1, W^+]$ (i.e., for $i = i^\iota$). Assume $L^{inc^\iota} = \emptyset$ and $L^\iota_\omega \neq \emptyset$; and say that* `ei-jump`$(i,J)$ *halts at step h. Then, the following two properties hold.*

1. *The front element $(\bar{w}, L_\alpha)$ of L^ι_ω satisfies $\bar{w} = \min\{w_e \mid e \in E, w_e > i\}$;*
2. *It holds that $J.i^h \geq \bar{w} > i$.*

Proof. At the first invocation of `ei-jump`(i,J) (SubProcedure 8), made at line 7 of Algorithm 15, it holds $i = W^- - 1$ (by line 5 of Algorithm 15). Since $L^{\text{inc}^\iota} = \emptyset$, then `ei-jump()` first assigns $J.i \leftarrow i + 1 = W^-$ at line 3. Since $L_{\mathbf{w}}$ was sorted in increasing order at line 12 of `init_jumper()` (SubProcedure 3), the front entry of $L_{\mathbf{w}}$ has key $w = W^-$, and all of the subsequent entries of $L_{\mathbf{w}}$ are binded to greater keys. Actually, `ei-jump()` consumes the front entry (W^-, L_α) of $L_{\mathbf{w}}$ at line 7; and W^- is assigned to $J.i$ (line 3). These observations imply both Item 1 and Item 2. Now, consider any invocation of `ei-jump`(i, J) (SubProcedure 8) which is not the first, but any subsequent one. Let us check that the front element $(\bar{w}, L_\alpha)$ of $L^\iota_{\mathbf{w}}$ satisfies $\bar{w} = \min\{w_e \mid e \in E, w_e > i^\iota\}$. Consider each line of Algorithm 15 at which the value i^ι could have ever been assigned to i; this may happen only as follows:

– At line 3 of `ei-jump()` (SubProcedure 8), i.e., $J.i \leftarrow i + 1$ $(= i^\iota)$. But then the front element $(\hat{w}, L_\alpha)$ of $L_{\mathbf{w}}$ is also checked at lines 5-6 (because $L_{\mathbf{w}} \neq \emptyset$): and $\hat{w}$ is popped from $L_{\mathbf{w}}$ at line 7, in case $\hat{w} = J.i$ $(= i^\iota)$ holds at line 6.

– The same happens at lines 2-5 of `rejoin_ua-jump()` (SubProcedure 9); just notice that in that case i was incremented just before at line 4 of `ua-jumps()` (SubProcedure 9).

– At lines 9-10 of `ei-jump()` (SubProcedure 8), whenever the front element $(\hat{w}, L_\alpha)$ of $L_{\mathbf{w}}$ is popped, then $J.i \leftarrow \hat{w}$ is assigned.

Therefore, in any case, the following holds:

When the variable i got any of its possible values, say $\hat{i}$ (including i^ι), the front entry $(\hat{w}, L_\alpha)$ of $L_{\mathbf{w}}$ had always been checked, and then popped from $L_{\mathbf{w}}$ if $\hat{w} = \hat{i}$.

Recall, $L_{\mathbf{w}}$ was sorted in increasing order at line 12 of `init_jumper()` (SubProcedure 3).

Therefore, when `ei-jump`(i^ι, J) is invoked at step ι, all of the entries (w, L_α) of $L_{\mathbf{w}}$ such that $w \leq i^\iota$ must already have been popped from $L_{\mathbf{w}}$ before step ι.

Therefore, $\bar{w} = \min\{w_e \mid e \in E, w_e > i^\iota\}$, if $\bar{w}$ is the key (weight) of the front entry of $L^\iota_{\mathbf{w}}$.

Next, since $L^{\text{inc}^\iota} = \emptyset$ and $L^\iota_{\mathbf{w}} \neq \emptyset$ by hypothesis, and by line 9 of `ei-jump()`, at least one further element (w, L_α) of $L_{\mathbf{w}}$ must be popped from $L^\iota_{\mathbf{w}}$, either at line 7 or line 10 of `ei-jump()`, soon after ι. Consider the last element, say w', which is popped after ι and before h. Then, $J.i^h \leftarrow w'$ is assigned either at line 3 or line 11 of `ei-jump()`. Notice, $w' \geq \bar{w} > i^\iota$. Thus, $J.i^h \geq \bar{w} > i^\iota$. □

The following proposition essentially asserts that `ei-jump()` (SubProcedure 8) is correct. To begin, notice that, when `ei-jump`(i, J) is invoked at line 7 of Algorithm 15, then $i \in [W^- - 1, W^+]$. Also recall that any invocation of `ei-jump`(i, J) halts in finite time by Proposition 7.3.

Proposition 7.5. *Consider any invocation of* `ei-jump`(i, J) *(SubProcedure 8) that is made at line 7 of Algorithm 15, say at step ι, and for some $i \in [W^- - 1, W^+]$. Further assume that $L^{inc^\iota} = \emptyset$ and that* `ei-jump`$()$ *halts at step h.*

Suppose the following pre-conditions are all satisfied at invocation time ι, for $s = |\mathcal{F}_{|V|}|$:

(eij-PC-1) $f^{c:\iota}$ is the least-SEPM of $\Gamma_{i,s-1}$; thus, $Inc(f^{c:\iota}, i, s-1) = \emptyset$. Also, $L^\iota_f = \emptyset$.

(eij-PC-2) $\{v \in V \mid 0 < f^{c:\iota}(v) \neq \top\} = \emptyset$;

(eij-PC-3) $L^{inc\,\iota}_{cpy} \subseteq Inc(f^{c:\iota}, i', j')$ *for every* $(i', j') > (i, s-1)$;

(eij-PC-4) $J.\texttt{cnt}^{\iota}$ *and* $J.\texttt{cmp}^{\iota}$ *are both coherent w.r.t.* $f^{c:\iota}$ *in* $\Gamma_{i,s-1}$.

Finally, let $i' \in [W^-, W^+]$, $j' \in [1, s-1]$ *be any indices such that* $(i, s-1) < (i', j') \leq (J.i^h, 1)$. *Then, the following holds.*

1. *Suppose that* $L^{\iota}_{\mathbf{w}} \neq \emptyset$. *Let* $(\hat{w}, L_{\hat{\alpha}})$ *be any entry of* $L^{\iota}_{\mathbf{w}}$ *such that* $\hat{w} = J.i^{\iota'} = i'$ *holds either at line 6 or line 11 of* $\texttt{ei-jump}(i, J)$, *for some step* $\iota' > \iota$. *When the* $\texttt{repair}(L_{\hat{\alpha}}, J)$ *halts soon after, either at line 8 or 12 (respectively), say at some step* $\iota'' > \iota'$, *both* $J.\texttt{cnt}^{\iota''}$ *and* $J.\texttt{cmp}^{\iota''}$ *are coherent w.r.t.* $f^{c:\iota''}$ $(= f^{c:\iota})$ *in* $\Gamma_{i',j'}$.

2. *If* $(i', j') < (J.i^h, 1)$, *then* $Inc(f^{c:\iota}, i', j') = \emptyset$;

3. *It holds that either* $L^{inc^h} \neq \emptyset$ *or both* $L^{inc^h} = \emptyset$ *and* $L^h_{\omega} = \emptyset$.

   Anyway, $L^{inc^h} = Inc(f^{c:h}, i^h, 1)$.

Notice that f^c stands still during $\texttt{ei-jump}()$ (SubProcedure 8), i.e., $f^{c:\iota} = f^{c:\iota'} = f^{c:\iota''} = f^{c:h}$, for steps $\iota, \iota', \iota'', h$ defined as in Proposition 7.5. In the proofs below, we can simply refer to f^c.

Proof of Item (1). Let $u \in V_0$ and $v \in N^{\text{out}}_{\Gamma}(u)$, let i', j' be fixed indices such that $(i, s-1) < (i', j') \leq (J.i^h, 1)$. By (*eij-PC-2*), either $f^c(u) = \top$ or $f^c(u) = 0$, either $f^c(v) = \top$ or $f^c(v) = 0$.

- If $f^c(u) = \top$, then $(u, v) \in E$ is compatible w.r.t. f^c in $\Gamma_{i,s-1}$. So, $J.\texttt{cmp}^{\iota}[(u, v)] = \texttt{T}$ holds by (*eij-PC-4*). Since $f^c(u) = \top$, $\texttt{ei-jump}()$ can't modify $J.\texttt{cmp}[(u, v)]$; see line 2 of $\texttt{repair}()$ (SubProcedure 8). So, $J.\texttt{cmp}^{\iota''}[(u, v)] = \texttt{T}$ is still coherent w.r.t. f^c in $\Gamma_{i',j'}$.

- If $f^c(u) = 0$ and $f^c(v) = \top$, then $(u, v) \in E$ is incompatible w.r.t. f^c in $\Gamma_{i,s-1}$. So, $J.\texttt{cmp}^{\iota}[(u, v)] = \texttt{F}$ holds by (*eij-PC-4*); and it will hold for the whole execution of $\texttt{ei-jump}()$, because $\texttt{ei-jump}()$ never changes $J.\texttt{cmp}$ from $\texttt{F}$ to $\texttt{T}$. Thus, $J.\texttt{cmp}^{\iota''}[(u, v)] = \texttt{F}$ is still coherent w.r.t. f^c in $\Gamma_{i',j'}$.

- Assume $f^c(u) = 0$ and $f^c(v) = 0$.

  Again, $J.\texttt{cmp}^{\iota}[(u, v)]$ is coherent w.r.t. f^c in $\Gamma_{i,s-1}$ by (*eij-PC-4*). We have two cases:

  – If $J.\texttt{cmp}^{\iota}[(u, v)] = \texttt{F}$, then (u, v) is incompatible w.r.t. f^c in $\Gamma_{i,s-1}$, i.e., $f^c(u) < f^c(v) - (w(u, v) - i - F_{s-1})$. Since $f^c(u) = f^c(v) = 0$ and $F_{s-1} = 1$, then:

$$0 = f^c(u) < f^c(v) - w(u, v) + i + F_{s-1} = -w(u, v) + i + 1.$$

Therefore, $w(u,v) \leq i$, because $w(u,v) \in \mathbb{Z}$. Since $(i, s-1) < (i', j')$, then $i < i'$, so $w(u,v) < i'$; this means that (u,v) is still incompatible w.r.t. f^c in $\Gamma_{i',j'}$. Meanwhile, $J.\texttt{cmp}[(u,v)] = \texttt{F}$ stands still (because `ei-jump()` never changes $J.\texttt{cmp}$ from `F` to `T`), therefore, $J.\texttt{cmp}^{\iota''}[(u,v)] = \texttt{F}$.

– If $J.\texttt{cmp}^{\iota}[(u,v)] = \texttt{T}$, then (u,v) is compatible w.r.t. f^c in $\Gamma_{i,s-1}$ by (*eij-PC-4*). So,

  ∗ If (u,v) is compatible w.r.t. f^c in $\Gamma_{i',j'}$, i.e., $f^c(u) \geq f^c(v) - (w(u,v) - i' - F_{j'})$, then, since $f^c(u) = f^c(v) = 0$, we have:

  $$0 = f^c(u) \geq f^c(v) - w(u,v) + i' + F_{j'} = -w(u,v) + i' + F_{j'}.$$

  Then, $w(u,v) > i'$, because $j' \in [1, s-1]$ (so, $F_{j'} > 0$) and $w(u,v) \in \mathbb{Z}$. Consider what happens in `ei-jump()` at ι'. Since $L_{\mathbf{w}}$ was sorted in increasing order, and since $w(u,v) > i'$, then the entry $(w(u,v), L_\alpha)$ is still inside $L_{\mathbf{w}}$ at ι' (indeed, at step ι', the front entry of $L_{\mathbf{w}}$ has key value i' by hypothesis). Therefore, neither the subsequent invocation of `repair()` (line 8 or line 12 of `ei-jump()`), nor any of the previous invocations of `repair()` (before ι'), can alter the state of $J.\texttt{cmp}([u,v])$ from `T` to `F`, just because $(u,v) \in L_\alpha$ is still inside $L_{\mathbf{w}}$ at ι'; so, $J.\texttt{cmp}([u,v]) = \texttt{T}$ stands still, thus, $J.\texttt{cmp}^{\iota''}[(u,v)] = \texttt{T}$.

  ∗ If $(u,v) \in E$ is incompatible w.r.t. f^c in $\Gamma_{i',j'}$, i.e., $f^c(u) < f^c(v) - (w(u,v) - i' - F_{j'})$, then, since $f^c(u) = f^c(v) = 0$, we have:

  $$0 = f^c(u) < f^c(v) - w(u,v) + i' + F_{j'} = -w(u,v) + i' + F_{j'}.$$

  Thus, $w(u,v) \leq i'$, because $f^c(u) = f^c(v) = 0$ and $j' \in [1, s-1]$ (so $F_{j'} > 0$). On the other side, since $(u,v) \in E$ is compatible w.r.t. f^c in $\Gamma_{i,s-1}$, at this point the reader can check that $w(u,v) > i$. Then, by Item 1 of Lemma 7.3, and since $L_{\mathbf{w}}$ was sorted in increasing order, the entry $(w(u,v), L_\alpha)$ is still inside $L_{\mathbf{w}}$ at ι. Therefore, since $w(u,v) \leq i'$, there must be some step $\hat{\iota}$ (such that $\iota < \hat{\iota} \leq \iota'$) at which the entry $(w(u,v), L_\alpha)$ must have been considered, either at line 5 or line 10 of `ei-jump`(i, J), and thus popped from $L_{\mathbf{w}}$. Soon after $\hat{\iota}$, the subsequent invocation of `repair()` (either at line 8 or line 12 of `ei-jump()`) changes the state of $J.\texttt{cmp}^{\hat{\iota}}[(u,v)]$ from `T` to `F` (line 5 of `repair()`), and it decrements $J.\texttt{cnt}^{\hat{\iota}}[(u,v)]$ by one unit (line 4 of `repair()`). Thus $J.\texttt{cmp}$ gets *repaired* so that to be coherent w.r.t. f^c in $\Gamma_{w(u,v),j'}$. Now, by Item 2 of Lemma 7.3, $J.i$ can only increase during the execution of `ei-jump()`. So, from that point on, (u,v) remains incompatible w.r.t. f^c in $\Gamma_{J.i,j'}$ for every $w(u,v) \leq J.i \leq i'$. On the other hand, $J.\texttt{cmp}^{\hat{\iota}}[(u,v)] = \texttt{F}$ stands still, since `ei-jump()` (SubProcedure 8) never changes it from `F` to `T`. So, $J.\texttt{cmp}^{\iota''}[(u,v)] = \texttt{F}$.

This proves that, in any case, $J.\texttt{cmp}^{\iota''}[(u,v)]$ is coherent w.r.t. f^{c} in $\Gamma_{i',j'}$.

This also proves that $J.\texttt{cnt}^{\iota''}$ is coherent w.r.t. f^{c} in $\Gamma_{i',j'}$: indeed, $J.\texttt{cnt}^{\iota}$ was coherent w.r.t. f^{c} in $\Gamma_{i,s-1}$ by (*eij-PC-4*); then $J.\texttt{cnt}$ was decremented by one unit (line 4 of `repair()`) each time that $J.\texttt{cmp}$ was repaired (line 5 of `repair()`), as described above; therefore, at step ι'', the coherency of $J.\texttt{cnt}^{\iota''}$ follows by that of $J.\texttt{cmp}^{\iota''}$.

□

Proof of Item (2). Let $u \in V$. We want to prove that u is consistent w.r.t. f^{c} in $\Gamma_{i',j'}$ for every $i' \in [W^-, W^+]$ and $j' \in [1, s-1]$ such that $(i, s-1) < (i', j') < (J.i^h, 1)$ (if any).

By (*eij-PC-2*), either $f^{\mathrm{c}}(u) = 0$ or $f^{\mathrm{c}}(u) = \top$. If $f^{\mathrm{c}}(u) = \top$, the claim holds trivially. Assume $f^{\mathrm{c}}(u) = 0$. By (*eij-PC-1*), u is consistent w.r.t. f^{c} in $\Gamma_{i,s-1}$. Assume $\hat{w} = J.i^{\iota'} = i'$, either at line 6 or line 11 of `ei-jump`(i, J), for some step $\iota' > \iota$. Assume `repair`$(L_{\hat{\alpha}}, J)$ halts soon after at line 8 or 12 (respectively), for some step ι'' where $\iota'' > \iota'$. We claim u is consistent w.r.t. f^{c} in $\Gamma_{i',j'}$.

If $u \in V_0$, By (*eij-PC-4*), $J.\texttt{cnt}^{\iota}[u]$ is coherent w.r.t. f^{c} in $\Gamma_{i,s-1}$. Thus, since u is consistent w.r.t. f^{c} in $\Gamma_{i,s-1}$, it holds that $J.\texttt{cnt}^{\iota}[u] > 0$. Now, since $(i', j') < (J.i^h, 1)$, then $i' < J.i^h$, thus $L^{\mathrm{inc}^{\iota''}} = \emptyset$. Therefore, $J.\texttt{cnt}^{\iota''}[u] > 0$ (otherwise u would have been inserted into L^{inc} within ι'' at line 7 of `repair()`). By Item 1 of Proposition 7.5, $J.\texttt{cnt}^{\iota''}[u]$ is coherent w.r.t. f^{c} in $\Gamma_{i',j'}$. Since $J.\texttt{cnt}^{\iota''}[u] > 0$ and $J.\texttt{cnt}[u]^{\iota''}$ is coherent, u is consistent w.r.t. f^{c} in $\Gamma_{i',j'}$.

If $u \in V_1$, Since $(i', j') < (J.i^h, i)$, then $i' < J.i^h$, thus $L^{\mathrm{inc}^{\iota''}} = \emptyset$. Let $v \in N_{\Gamma}^{\mathrm{out}}(u)$. By (*eij-PC-2*), either $f^{\mathrm{c}}(v) = 0$ or $f^{\mathrm{c}}(v) = \top$. Since $f^{\mathrm{c}}(u) = 0$ by assumption, $u \in V_1$, and u is consistent w.r.t. f^{c} in $\Gamma_{i,s-1}$ by (*eij-PC-1*), then $f^{\mathrm{c}}(v) = 0$.

Now, we argue that $w(u,v) > i'$.

Assume, for the sake of contradiction, that $w(u,v) \le i'$. On one side, since u is consistent w.r.t. f^{c} in $\Gamma_{i,s-1}$ and since $f^{\mathrm{c}}(u) = f^{\mathrm{c}}(v) = 0$, then:

$$0 = f^{\mathrm{c}}(u) \ge f^{\mathrm{c}}(v) - w(u,v) + i + F_{s-1} = -w(u,v) + i + 1.$$

Thus, $w(u,v) \ge i + 1 > i$. Therefore, by Lemma 7.3, the entry $(w(u,v), L_{\alpha})$ is still inside $L_{\mathbf{w}}$ at step ι. On the other side, since $w(u,v) \le i'$, it is easy to see at this point that within (or soon after) step ι' the entry $(w(u,v), L_{\alpha'})$ must have been popped from $L_{\mathbf{w}}$ either at line 7 or at line 10 of `ei-jump()` (SubProcedure 8). So, $(w(u,v), L_{\alpha'})$ must have been popped from $L_{\mathbf{w}}$ after ι and within ι' (or soon after ι' at line 7). But soon after that, since $u \in V_1$, the subsequent invocation of `repair()` would insert u into L^{inc} at line 8, because $f^{\mathrm{c}}(u) = f^{\mathrm{c}}(v) = 0$ and $L^{\mathrm{inc}}[u] = \emptyset$ at line 2.

Therefore $L^{\mathrm{inc}^{\iota''}} \neq \emptyset$, which is a contradiction. Therefore, $w(u,v) > i'$. Since $v \in N_\Gamma^{\mathrm{out}}(u)$ was chosen arbitrarily, $\forall^{v \in N_\Gamma^{\mathrm{out}}(u)}\ w(u,v) > i'$.

Since $\forall^{v \in N_\Gamma^{\mathrm{out}}(u)}\ w(u,v) > i'$ and $f^{\mathrm{c}}(u) = f^{\mathrm{c}}(v) = 0$, then u is consistent w.r.t. f^{c} in $\Gamma_{i',j'}$.

So, u is consistent w.r.t. f^{c} in $\Gamma_{i',j'}$. Since $u \in V$ was chosen arbitrarily, then $\mathrm{Inc}(f^{\mathrm{c}:\iota}, i', j') = \emptyset$. □

Proof of Item (3). By line 9 of `ei-jump()` (SubProcedure 8), when the `while` loop at lines 9-12 halts, then `ei-jump()` halts soon after at line 13, say at step h, and it must be that either $L^{\mathrm{inc}^h} \neq \emptyset$ or both $L^{\mathrm{inc}^h} = \emptyset$ and $L_{\mathbf{w}}^h = \emptyset$. Now, we want to prove that $L^{\mathrm{inc}^h} = \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$.

- Firstly, $L^{\mathrm{inc}^h} \subseteq \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$:

Assume $u \in L^{\mathrm{inc}^h}$. We have three cases to check:

– If $u \in L_{\mathrm{cpy}}^{\mathrm{inc}\,\iota}$, notice that $J.i^h > i$ by Lemma 7.3, then $u \in \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$ holds by (eij-PC-3);

– If $u \in V_0 \setminus L_{\mathrm{cpy}}^{\mathrm{inc}\,\iota}$, then $J.\mathtt{cnt}^h[u] = 0$ by lines 6-7 of `repair()` (SubProcedure 8). By Item 1 of Proposition 7.5, $J.\mathtt{cnt}^h[u]$ is coherent w.r.t. f^{c} in $\Gamma_{J.i^h,1}$. Therefore, $u \in \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$.

– If $u \in V_1 \setminus L_{\mathrm{cpy}}^{\mathrm{inc}\,\iota}$, then $\exists^{v \in N_\Gamma^{\mathrm{out}}(u)}\ f^{\mathrm{c}}(u) = f^{\mathrm{c}}(v) = 0$ and $w(u,v) = J.i^h$ by lines 2-8 of `repair()`. Therefore, $u \in \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$.

This proves, $L^{\mathrm{inc}^h} \subseteq \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$.

- Secondly, $L^{\mathrm{inc}^h} \supseteq \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$:

Let $u \in \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$. By (*eij-PC-2*), either $f^{\mathrm{c}}(u) = 0$ or $f^{\mathrm{c}}(u) = \top$. Since $u \in \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$, then $f^{\mathrm{c}}(u) = 0$. Now, by (*eij-PC-1*), u is consistent w.r.t. f^{c} in $\Gamma_{i,s-1}$. So, let $(u, \hat{v}) \in E$ be any arc which is compatible w.r.t. f^{c} in $\Gamma_{i,s-1}$.

If $u \in V_0$, then, *at least one* such a compatible $\hat{v} \in N_\Gamma^{\mathrm{out}}(u)$ exists (because $u \in V_0$ is consistent w.r.t. f^{c} in $\Gamma_{i,s-1}$). By (*eij-PC-2*), either $f^{\mathrm{c}}(\hat{v}) = 0$ or $f^{\mathrm{c}}(\hat{v}) = \top$. Since $f^{\mathrm{c}}(u) = 0$ and $(u, \hat{v})$ is compatible w.r.t. f^{c} in $\Gamma_{i,s-1}$, then $f^{\mathrm{c}}(\hat{v}) = 0$. Since $f^{\mathrm{c}}(u) = f^{\mathrm{c}}(\hat{v}) = 0$ and $u \in \mathrm{Inc}(f^{\mathrm{c}}, J.i^h, 1)$, then $w(u, \hat{v}) \leq J.i^h$.

We claim that at some step of execution of line 7 or line 10 in `ei-jump()` (SubProcedure 8), say at step ι' for $\iota < \iota' < h$, the entry $(w(u,\hat{v}), L_\alpha)$ is popped from $L_{\mathbf{w}}$.

Since $f^{\mathrm{c}}(u) = f^{\mathrm{c}}(\hat{v}) = 0$, and $(u,\hat{v})$ is compatible w.r.t. f^{c} in $\Gamma_{i,s-1}$, then $w(u,\hat{v}) > i$. Thus, by Lemma 7.3, when `ei-jump()` is invoked (i.e., at step ι), the front entry $(\bar{w}, L_{\bar{\alpha}})$ of $L_{\mathbf{w}}^{\iota}$ satisfies $\bar{w} = \min\{w_e \mid e \in E, w_e > i\} \leq w(u,\hat{v})$. So, $\bar{w} \leq w(u,\hat{v}) \leq J.i^h$. Thus, at some step of execution ι' for $\iota < \iota' < h$, the entry $(w(u,\hat{v}), L_\alpha)$ must be popped from $L_{\mathbf{w}}$.

Soon after that, `repair`(L_α, J) is invoked: there, since $f^c(u) = f^c(\hat{v}) = 0$ and $u \in V_0$, then $J.$`cnt`$[u]$ is decremented by one unit at line 4 of `repair`$()$.

Indeed, this happens (after ι but before h), *for every* $v \in N^{\text{out}}_\Gamma(u)$ such that $(u,v) \in E$ is compatible w.r.t. f^c in $\Gamma_{i,s-1}$. Thus, eventually and before h, it will hold $J.$`cnt`$[u] = 0$. At that point, u will be inserted into L^{inc} at line 7 of `repair`$()$; and soon after, `ei-jump`$()$ halts (since $L^{\text{inc}} \neq \emptyset$ at line 9). So, $u \in {L^{\text{inc}}}^h$. This holds for every $u \in \text{Inc}(f^c, J.i^h, 1) \cap V_0$. Thus, $\text{Inc}(f^c, J.i^h, 1) \cap V_0 \subseteq {L^{\text{inc}}}^h$.

If $u \in V_1$, then, *all* $\hat{v} \in N^{\text{out}}_\Gamma(u)$ are such that $(u,\hat{v})$ is compatible w.r.t. f^c in $\Gamma_{i,s-1}$, because $u \in V_1$ is consistent w.r.t. f^c in $\Gamma_{i,s-1}$ by (*eij-PC-1*). The argument proceeds almost in the same way as before. By *(eij-PC-2)*, either $f^c(\hat{v}) = 0$ or $f^c(\hat{v}) = \top$. Since $f^c(u) = 0$ and $(u,\hat{v})$ is compatible w.r.t. f^c in $\Gamma_{i,s-1}$, then $f^c(\hat{v}) = 0$. Since $f^c(u) = f^c(\hat{v}) = 0$ and $u \in \text{Inc}(f^c, J.i^h, 1)$, then $w(u,\hat{v}) \leq J.i^h$. By arguing as above, we see that at some step of execution of line 3 in (the considered invocation of) `ei-jump`$()$ (SubProcedure 8), say at step ι' for $\iota < \iota' < h$, the entry $(w(u,\hat{v}), L_\alpha)$ is popped from $L_{\mathbf{w}}$. Soon after that, `repair`(L_α, J) is invoked: there, since $f^c(u) = f^c(\hat{v}) = 0$ and $u \in V_1$, then u is inserted into L^{inc}; soon after, `ei-jump`$()$ halts (since $L^{\text{inc}} \neq \emptyset$); so, $u \in {L^{\text{inc}}}^h$. This holds for every $u \in \text{Inc}(f^c, J.i^h, 1) \cap V_1$; so, $\text{Inc}(f^c, J.i^h, 1) \cap V_1 \subseteq {L^{\text{inc}}}^h$.

Therefore, $\text{Inc}(f^c, J.i^h, 1) = {L^{\text{inc}}}^h$. □

Correctness of **`ua-jumps`**$()$ (SubProcedure 9)

Proposition 7.6. *Consider any invocation of `ei-jump`$()$ (SubProcedure 8) that is made at line 7 of Algorithm 15. Assume that the pre-conditions (eij-PC-1), (eij-PC-2), (eij-PC-3), (eij-PC-4), are all satisfied at invocation time. Further assume that $L^{inc} \neq \emptyset$ at line 8 of Algorithm 15, so that `ua-jumps`$()$ is invoked soon after at line 9. Then, consider any invocation of `J-VI`$(i, s-1, F, J, \Gamma)$ (SubProcedure 6) that is made at line 2 of `ua-jumps`$()$ (SubProcedure 9), for some $i \in [W^-, W^+]$, where $s = |\mathcal{F}_{|V|}|$. Then, the following properties hold.*

1. *The* (PC-1), (w-PC-2), (w-PC-3) *are all satisfied by that invocation of `J-VI`$(i, s-1, F, J, \Gamma)$.*

2. *When the `J-VI`$(i, s-1, F, J, \Gamma)$ halts, say at step h, then the following holds:*

$$L^h_\top = \mathcal{W}_0(\Gamma_{i-1,s-1}) \cap \mathcal{W}_1(\Gamma_{i,s-1}).$$

3. *Assume that `backtrack_ua-jumps`(i, s, F, J, Γ) is invoked at line 7 of `ua-jumps ()`, say at step ι, and assume that it halts at step h.*

(a) *At line 1 of* `backtrack_ua-jumps`(i,s,F,J,Γ)*, it holds:*

$$L^{inc}_{cpy} = \{v \in V \mid 0 < f^{c:\iota}(v) \neq \top\}.$$

(b) *Consider the two induced games* $\Gamma[L^\iota_\top]$ *and* $\Gamma[V \setminus L^\iota_\top]$*. The following holds:*

i. $\forall (v \in L^\iota_\top)\, f^{c:h}(v) = f^*_{w'_{i-1,s-1}}(v)$;

ii. $\forall (v \in V \setminus L^\iota_\top)\, f^{c:h}(v) = f^*_{w'_{i,s-1}}(v)$;

iii. $\forall (u \in L^\iota_\top \cap V_0) \forall (v \in N^{out}_{\Gamma[L^\iota_\top]}(u))$ $J.cmp^h(u,v)$ *is coherent w.r.t.* $f^{c:h}$ *in* $\Gamma[L^\iota_\top]_{i,1}$;

iv. $\forall (v \in L^\iota_\top \cap V_0)$ $J.cnt^h(v)$ *is coherent w.r.t.* $f^{c:h}$ *in* $\Gamma[L^\iota_\top]_{i,1}$;

v. $\forall (j' \in [1,s-1])$ *Inc*$(f^{c:h},i,j') \setminus L^\iota_\top = \emptyset$.

4. *Any invocation of* `ua-jumps()` *(SubProcedure 9) (line 7, Algorithm 15) halts in finite time.*

Proof of Item (1). By induction on the number $k \in \mathbb{N}$ of invocations of `J-VI()` that are made at line 2 of `ua-jumps()`.

Base Case: $k = 1$. Consider the first invocation of `J-VI()` at line 2 of `ua-jumps()`, say it happens at step ι. Just before step ι, Algorithm 15 invoked `ei-jump()` at line 7. By hypothesis, (*eij-PC-1*), (*eij-PC-2*), (*eij-PC-3*), (*eij-PC-4*) are all satisfied at that time. Then:

– (*PC-1*): It is easy to check from the definitions that (*eij-PC-1*) directly implies (*PC-1*).

– (*w-PC-2*): By Item 3 of Proposition 7.5, it holds that $L^{\text{inc}^\iota} = \text{Inc}(f^{c:\iota},i,1)$. Since $\text{Inc}(f^{c:\iota},i,1) \subseteq \text{Inc}(f^{c:\iota},i,s-1)$, then (*w-PC-2*) holds.

– (*w-PC-3*): Let $u \in V \setminus L^{\text{inc}^\iota}$ and $v \in N^{\text{out}}_\Gamma(u)$. We need to check the following two cases.

If $u \in V_0$, by Item 1 of Proposition 7.5, both $J.\texttt{cmp}^\iota$ and $J.\texttt{cnt}^\iota$ are coherent w.r.t. $f^{c:\iota}$ in $\Gamma_{i,1}$. Therefore, (*w-PC-3*) holds when $u \in V_0$.

If $u \in V_1$ and (u,v) is incompatible w.r.t. $f^{c:\iota}$ in $\Gamma_{i,1}$, then, $u \in \text{Inc}(f^{c:\iota},i,1)$. By Item 3 of Proposition 7.5, $\text{Inc}(f^{c:\iota},i,1) = L^{\text{inc}^\iota}$. Thus, $u \in L^{\text{inc}^\iota}$, so (*w-PC-3*) holds when $u \in V_1$.

Therefore, (*w-PC-3*) holds when $k = 1$.

Inductive Step: $k > 1$. Consider the k-th invocation of `J-VI()` for $k > 1$, at line 2 of `ua-jumps()`. Say it happens at step ι. Since $k > 1$, just before step ι, Algorithm 15 performed the $(k-1)$-th invocation of `J-VI()` at line 2 of `ua-jumps()`. Say it happened at step ι_0. By induction hypothesis, at step ι_0 the (*PC-1*), (*w-PC-2*), (*w-PC-3*) were all satisfied. Therefore, the $(k-1)$-th invocation of `J-VI()` at line 2 halted in a correct manner, as prescribed by Proposition 7.4. Soon after that, Algorithm 15 invoked `rejoin_ua-jump()` at line 5 of `ua-jumps()`.

($\star$) The key is that `rejoin_ua-jump()`, apart from copying the energy-levels of L_f back to $J.f$ (with `scl_back_`$f(s-1, F, J)$ at line 1), it takes care of repairing (at line 6) the coherency state of J.`cnt`$[u]$ and J.`cmp`$[(u,v)]$ for all those $(u,v) \in E$ such that: $u \in V_0$, $w(u,v) = i$ and $J.f[u] = J.f[v] = 0$ (if any); moreover, it checks the compatibility state of all those arcs $(u,v) \in E$ such that: $u \in V_1$, $w(u,v) = i$ and $J.f[u] = J.f[v] = 0$ (if any). In doing so, if any $u \in V \setminus L^{\text{inc}}$ is recognized to be inconsistent w.r.t. f^c in $\Gamma_{i,s-1}$, then u is (correctly) inserted into L^{inc}. See the pseudo-code of `repair()` in SubProcedure 8.

With ($\star$) in mind, we can check that (*PC-1*), (*w-PC-2*), (*w-PC-3*) are all satisfied at step ι.

– (*PC-1*). Since `rejoin_ua-jump()` empties L_f (by `scl_back_`$f()$ at line 1), then $L_f^{\iota} = \emptyset$. Next, we argue $f^c \preceq f^*_{w'_{i,s-1}}$. By induction hypothesis, when the $(k-1)$-th invocation of `J-VI()` at line 2 of `ua-jumps()` halts, Proposition 7.4 holds, therefore, $f^c = f^*_{w'_{i-1,s-1}}$. Since $F_{s-1} = 1$, then $w'_{i-1,s-1} = w_{i-1,s-1}$ and $w'_{i,s-1} = w_{i,s-1}$. Therefore, the following holds for every $v \in V$:

$$\begin{aligned}
f^c(v) =& f^*_{w'_{i-1,s-1}}(v) && \text{[by induction hypothesis and Proposition 7.4]} \\
=& f^*_{i-1,s-1}(v) && [\text{by } w'_{i-1,s-1} = w_{i-1,s-1}] \\
\preceq& f^*_{i,s-1}(v) && [\text{by } w_{i-1,s-1} > w_{i,s-1} \text{ and Lemma 7.2}] \\
=& f^*_{w'_{i,s-1}}(v) && [\text{by } w_{i,s-1} = w'_{i,s-1}]
\end{aligned}$$

In summary, $\forall^{v \in V} f^c(v) \preceq f^*_{w'_{i,s-1}}(v)$. This proves (PC-1).

– (w-PC-2). By induction hypothesis and Proposition 7.2, all vertices that were already inside L^{inc} at the end of the $(k-1)$-th invocation of `J-VI()`, at line 2 of `ua-jumps()`, they were all inconsistent w.r.t. $J.f^c$ in $\Gamma_{i-1,s-1}$, so they are still inconsistent w.r.t. $J.f^c$ in $\Gamma_{i,s-1}$, because $w'_{i,s-1} < w'_{i-1,s-1}$. In addition, the repairing process performed by `rejoin_ua-jumps()`, as mentioned in ($\star$), can only add inconsistent vertices to L^{inc}. Therefore, (w-PC-2) holds.

– (w-PC-3). Let $u \in V \setminus L^{\text{inc}^{\iota}}$ and $v \in N^{\text{out}}_{\Gamma}(u)$. We need to check the following two cases.

Case $u \in V_0$. In order to prove Item 1 of (w-PC-3), we need to check three cases.

1. If J.`cmp`$^{\iota}[u,v] =$ `F`, we argue that $(u,v) \in E$ is incompatible w.r.t. f^c in $\Gamma_{i,s-1}$.

   Indeed, one of the following two cases (i) or (ii) holds:

   (i) J.`cmp`$[(u,v)] =$ `F` was already so at the end of the $(k-1)$-th invocation of `J-VI()`. By induction hypothesis and by Item 2 of Proposition 7.4, then (u,v) was incompatible w.r.t. f^c in $\Gamma_{i-1,s-1}$. So, (u,v) is incompatible w.r.t. f^c in $\Gamma_{i,s-1}$ (as $w'_{i,s-1} < w'_{i-1,s-1}$).

   (ii) at the end of the $(k-1)$-th invocation of `J-VI()`, it was J.`cmp`$[(u,v)] =$ `T`. But then, `rejoin_ua-jump`(i,s,F,J) repaired it by setting J.`cmp`$[(u,v)] =$ `F` at line 5 of `repair()`; notice that this (correctly) happens *iff* $w(u,v) =$

i and $J.f[u] = J.f[v] = 0$, so that (u,v) is really incompatible w.r.t. f^c in $\Gamma_{i,s-1}$.

Therefore, in any case, Item 1 of (w-PC-3) holds.

2. If $J.\texttt{cmp}^{\iota}[(u,v)] = \texttt{T}$, we argue that either (u,v) is compatible w.r.t. f^c in $\Gamma_{i,s-1}$ or $v \in L^{\text{inc}^{\iota}}$. Indeed, assume $J.\texttt{cmp}^{\iota}[(u,v)] = \texttt{T}$ and that (u,v) is incompatible w.r.t. f^c in $\Gamma_{i,s-1}$. Since $J.\texttt{cmp}^{\iota}[(u,v)] = \texttt{T}$, then it was as such even when the $(k-1)$-th invocation of `J-VI()` halted at line 2 of `ua-jumps()`. By induction hypothesis and by Item 2 of Proposition 7.4, (u,v) was compatible w.r.t. f^c in $\Gamma_{i-1,s-1}$. But (u,v) is now incompatible w.r.t. f^c in $\Gamma_{i,s-1}$, and still $J.\texttt{cmp}^{\iota}[(u,v)] = \texttt{T}$. Thus, the last invocation of `repair()`, within the last `rejoin_ua-jump()`, has not recognized (u,v) as incompatible (otherwise, it would be $J.\texttt{cmp}^{\iota}[(u,v)] = \texttt{F}$). Therefore, it must be that $f^c(u) > 0$ or $f^c(v) > 0$: otherwise, if $f^c(u) = f^c(v) = 0$, since (u,v) is compatible w.r.t. f^c in $\Gamma_{i-1,s-1}$ but incompatible in $\Gamma_{i,s-1}$, and $w(u,v) \in \mathbb{Z}$, then $w(u,v) = i$ (contradicting the fact that the last invocation of `repair()` has not recognized (u,v) as incompatible). Moreover, since (u,v) is incompatible w.r.t. f^c in $\Gamma_{i,s-1}$, then $f^c(u) \neq \top$; and since (u,v) was compatible w.r.t. f^c in $\Gamma_{i-1,s-1}$ and $f^c(u) \neq \top$, then $f^c(v) \neq \top$. Now, when the $(k-1)$-th invocation of `J-VI()` halts, $L^{\text{inc}} = \{q \in V \mid 0 < f^c(q) \neq \top\}$ holds by induction hypothesis and Item 3 of Proposition 7.4. Since $u \notin L^{\text{inc}^{\iota}}$ and $f^c(u) \neq \top$, then $f^c(u) = 0$. Thus, since either $f^c(u) > 0$ or $f^c(v) > 0$, it holds that $f^c(v) > 0$. So, it is $0 < f^c(v) \neq \top$. Therefore, $v \in L^{\text{inc}^{\iota}}$.

3. By induction hypothesis and by Proposition 7.4, when the $(k-1)$-th invocation of `J-VI()` halts at line 2 of `ua-jumps()`, say at step ι_0, f^c is the least-SEPM of the EG $\Gamma_{i-1,s-1}$ and $J.\texttt{cnt}^{\iota_0}$, $J.\texttt{cmp}^{\iota_0}$ are both coherent w.r.t. f^c in $\Gamma_{i-1,s-1}$. Therefore,

$$\begin{aligned} J.\texttt{cnt}^{\iota_0}[u] &= \left|\left\{v \in N^{\text{out}}_{\Gamma}(u) \mid f^c(u) \succeq f^c(v) \ominus w'_{i-1,s-1}(u,v)\right\}\right| && \text{[by coherency of } J.\texttt{cnt}^{\iota_0}\text{]} \\ &= \left|\left\{v \in N^{\text{out}}_{\Gamma}(u) \mid J.\texttt{cmp}^{\iota_0}[(u,v)] = \texttt{T}\right\}\right|. && \text{[by coherency of } J.\texttt{cmp}^{\iota_0}\text{]} \end{aligned}$$

Moreover, since f^c is least-SEPM of $\Gamma_{i-1,s-1}$, then u is consistent w.r.t. f^c in $\Gamma_{i-1,s-1}$; thus $J.\texttt{cnt}^{\iota_0}[u] > 0$. Then, after ι_0 and before ι, `ua-jumps()` increments i by one unit at line 4 and it invokes `rejoin_ua-jump()` at line 5. There, `repair()` can (possibly) alter the state of both $J.\texttt{cnt}$ and $J.\texttt{cmp}$ at lines 4-5. Whenever the state of $J.\texttt{cmp}$ is modified from `T` to `F`, then $J.\texttt{cnt}$ is decremented by one unit; moreover, whenever $J.\texttt{cnt}[u] = 0$, then `repair()` takes care of inserting u into L^{inc}. Therefore, $J.\texttt{cnt}^{\iota}[u] = \left|\left\{v \in N^{\text{out}}_{\Gamma}(u) \mid J.\texttt{cmp}^{\iota}[(u,v)] = \texttt{T}\right\}\right|$; since $u \notin L^{\text{inc}^{\iota}}$, then $J.\texttt{cnt}^{\iota}[u] > 0$.

Case $u \in V_1$. Let $v \in N^{\text{out}}_{\Gamma}(u)$ be such that (u,v) is incompatible w.r.t. f^c in $\Gamma_{i,j}$. We claim $v \in L^{\text{inc}^{\iota}}$. By induction hypothesis and by Proposition 7.4, when the $(k-1)$-th invocation of `J-VI()` halts at line 2 of `ua-jumps()`, say at step ι_0, f^c is the least-SEPM of the EG $\Gamma_{i-1,s-1}$ and $L^{\text{inc}^{\iota_0}} = \{q \in V \mid 0 < f^c(q) \neq \top\}$. Thus, since $u \in V_1$, the arc (u,v) is compatible w.r.t. f^c in $\Gamma_{i-1,s-1}$. Moreover, since $u \notin L^{\text{inc}^{\iota}}$ by hypothesis, then $u \notin L^{\text{inc}^{\iota_0}}$, thus $f^c(u) = 0$ or $f^c(u) = \top$; but since u is incompatible w.r.t. f^c in $\Gamma_{i,j}$, it is $f^c(u) = 0$. Now, if $0 < f^c(v) \neq \top$, then $v \in L^{\text{inc}^{\iota_0}} \subseteq L^{\text{inc}^{\iota}}$, so we are done. Otherwise, since (u,v) is compatible w.r.t. f^c in $\Gamma_{i-1,s-1}$ and $f^c(u) = 0$, then $f^c(v) \neq \top$. So, $f^c(v) = 0$. Since $f^c(u) = f^c(v) = 0$ and (u,v) is compatible w.r.t. f^c in $\Gamma_{i-1,s-1}$, but incompatible w.r.t. f^c in $\Gamma_{i,s-1}$, and $w(u,v) \in \mathbb{Z}$, then $w(u,v) = i$. Whence, soon after ι_0, when `ua-jumps()` invokes `rejoin_ua-jump()` at line 5; there inside, `repair()` takes care of inserting v into L^{inc}. Therefore, $v \in L^{\text{inc}^{\iota}}$.

This concludes the inductive step, and thus the proof of Item 1 of Proposition 7.6. □

Proof of Item (2). Consider the first invocation of `J-VI()` at line 2 of `ua-jumps()`, say it happens at step ι. Notice that, just before step ι, the `ei-jump()` was invoked by Algorithm 15 at line 7, say at step ι_0. By hypothesis, (*eij-PC-1*), (*eij-PC-2*), (*eij-PC-3*), (*eij-PC-4*) were all satisfied at step ι_0. By (*eij-PC-1*) and Item 2 of Proposition 7.5, $f^{c:\iota_0}$ is the least-SEPM of $\Gamma_{i-1,s-1}$; therefore, $V_{f^{c:\iota_0}} = \mathcal{W}_0(\Gamma_{i-1,s-1})$ by Lemma 6.3. By Item 1 of Proposition 7.6 and Item 1 of Proposition 7.4, when the first invocation of `J-VI()` halts, say at step h, then $f^{c:h}$ is the least-SEPM of $\Gamma_{i,s-1}$; therefore, $V \setminus V_{f^{c:h}} = \mathcal{W}_1(\Gamma_{i,s-1})$ by Lemma 6.3. Moreover, by Item 4 of Proposition 7.4, $L^h_\top = V_{f^{c:\iota}} \cap V \setminus V_{f^{c:h}}$. Therefore, $L^h_\top = \mathcal{W}_0(\Gamma_{i-1,s-1}) \cap \mathcal{W}_1(\Gamma_{i,s-1})$.

Next, consider the k-th invocation of `J-VI()`, for $k > 1$, at line 2 of `ua-jumps()`. By Item 1 of Proposition 7.6 and Item 1 of Proposition 7.4, the following two hold: (i) when the $(k-1)$-th invocation of `J-VI()` halts, at line 2 of `ua-jumps()`, say at step ι, then $f^{c:\iota}$ is the least-SEPM of $\Gamma_{i-1,s-1}$; and $V_{f^{c:\iota}} = \mathcal{W}_0(\Gamma_{i-1,s-1})$ by Lemma 6.3. (ii) when the k-th invocation of `J-VI()` halts, at line 2 of `ua-jumps()`, say at step h, then $f^{c:h}$ is the least-SEPM of $\Gamma_{i,s-1}$; and $V \setminus V_{f^{c:h}} = \mathcal{W}_1(\Gamma_{i,s-1})$ by Lemma 6.3. Notice that, when the k-th invocation of `J-VI()` takes place, soon after ι, the current energy-levels are still $f^{c:\iota}$ (i.e., they are not modified by `rejoin_ua-jumps()` at line 5 of `ua-jumps()`). Moreover, by Item 4 of Proposition 7.4, $L^h_\top = V_{f^{c:\iota}} \cap V \setminus V_{f^{c:h}}$. Therefore, by (i) and (ii), it holds $L^h_\top = \mathcal{W}_0(\Gamma_{i-1,s-1}) \cap \mathcal{W}_1(\Gamma_{i,s-1})$. □

Proof of Item (3). We need to check the following two items (a) and (b).

(a) Consider the state of $L^{\text{inc}}_{\text{cpy}}$ at line 1 of `backtrack_ua-jumps(`i,s,F,J,Γ`)`. By Item 1 of Proposition 7.6, and by Item 3 of Proposition 7.4, when the last invocation of `J-VI()` halts at line 2 of `ua-jumps()`, say at step h_0, it holds $J.L^{\text{inc}^{h_0}} = \{v \in V \mid 0 < f^{\text{c}:h_0}(v) \neq \top\}$. By the copy operation which is performed at line 1 of `backtrack_ua-jumps()`, then $J.L^{\text{inc}}_{\text{cpy}} = J.L^{\text{inc}^{h_0}} = \{v \in V \mid 0 < f^{\text{c}:h_0}(v) \neq \top\}$. This proves (a).

(b) Let us focus on the two induced games $\Gamma[L^{\iota}_{\top}]$ and $\Gamma[V \setminus L^{\iota}_{\top}]$.

i. $\forall(v \in L^{\iota}_{\top})\ f^{\text{c}:h}(v) = f^*_{w'_{i-1,s-1}}(v)$: indeed, by arguing similarly as in the proof of Item 2 of Proposition 7.6, we obtain $\forall^{v\in V}\ J.f^{\iota}[v] = f^*_{w'_{i-1,s-1}}(v)$.
Notice that `backtrack_ua-jump()` modifies the energy-levels only at lines 2-3, where the following assignment is performed:

$$L^h_f[u] \leftarrow \begin{cases} \bot & \text{, if } u \in L^{\iota}_{\top}; \\ L^{\iota}_f[u] & \text{, if } u \in V \setminus L^{\iota}_{\top}. \end{cases}$$

and `scl_back_`f`()` is invoked. Since $\forall(v \in L^{\iota}_{\top})\ L^h_f[u] = \bot$, soon after the invocation of `scl_back_`f`()` at line 3, it must be that $\forall(v \in L^{\iota}_{\top})\ f^{\text{c}:h}(v) = J.f^{\iota}[v] = f^*_{w'_{i-1,s-1}}(v)$. This proves (i).

ii. $\forall(v \in V \setminus L^{\iota}_{\top})\ f^{\text{c}:h}(v) = f^*_{w'_{i,s-1}}(v)$: indeed, by Item 1 of Proposition 7.6 and Item 1 of Proposition 7.4, $\forall(v \in V)\ f^{\text{c}:\iota}(v) = f^*_{w'_{i,s-1}}(v)$. As mentioned, `backtrack_ua-jump()` modifies the energy-levels only at lines 2-3, where $L^h_f[u]$ is assigned (as above in (i)). Since $\forall(v \in V \setminus L^{\iota}_{\top})$ $(L^h_f[u] = L^{\iota}_f[u]$ and $f^{\text{c}:\iota}(v) = f^*_{w'_{i,s-1}}(v))$, then (ii) holds.

iii. $\forall(u \in L^{\iota}_{\top} \cap V_0)\forall(v \in N^{\text{out}}_{\Gamma[L^{\iota}_{\top}]}(u))\ J.\texttt{cmp}^h(u,v)$ is coherent w.r.t. $f^{\text{c}:h}$ in $\Gamma[L^{\iota}_{\top}]_{i,1}$: indeed, at lines 4-7 of `backtrack_ua-jump()`, for each $u \in L^{\iota}_{\top} \cap V_0$, it is invoked the `init_cnt_cmp`$(u,i,1,F,J,\Gamma[L^{\iota}_{\top}])$ (line 7). Therefore, (iii) holds.

iv. $\forall(v \in L^{\iota}_{\top} \cap V_0)\ J.\texttt{cnt}^h(v)$ is coherent w.r.t. $f^{\text{c}:h}$ in $\Gamma[L^{\iota}_{\top}]_{i,1}$: same argument as in (iii).

v. $\forall(j' \in [1,s-1])\ \text{Inc}(f^{\text{c}:h},i,j') \setminus L^{\iota}_{\top} = \emptyset$: indeed, let $u \in V \setminus L^{\iota}_{\top}$ and let $j' \in [1,s-1]$ be fixed arbitrarily. We want to show that $u \notin \text{Inc}(f^{\text{c}:h},i,j')$. Since $f^*_{w'_{i,s-1}}$ is the least-SEPM of $\Gamma_{i,s-1}$, then $u \notin \text{Inc}(f^*_{w'_{i,s-1}},i,s-1)$. We have two cases.

Case $u \in V_0 \setminus L^{\iota}_{\top}$ Since $u \in V_0 \setminus \text{Inc}(f^*_{w'_{i,s-1}},i,s-1)$, for *some* $v \in N^{\text{out}}_{\Gamma}(u)$ it holds that:

$$f^*_{w'_{i,s-1}}(u) \succeq f^*_{w'_{i,s-1}}(v) \ominus w'_{i,s-1}(u,v). \qquad (*_0)$$

By Item (i) of Proposition 7.6, it holds that $\forall(v \in L^{\iota}_{\top})\ f^{\text{c}:h}(v) = f^*_{w'_{i-1,s-1}}(v)$. By Item (ii) of Proposition 7.6, it holds that $\forall(v \in V \setminus L^{\iota}_{\top})\ f^{\text{c}:h}(v) = f^*_{w'_{i,s-1}}(v)$; so $f^{\text{c}:h}(u) = f^*_{w'_{i,s-1}}(u)$. By Lemma 7.2, $f^*_{w'_{i-1,s-1}} \preceq f^*_{w'_{i,s-1}}$.

Then, since $f^{c:h}(u) = f^*_{w'_{i,s-1}}(u)$, since $f^{c:h}(v) \in \{f^*_{w'_{i-1,s-1}}(v), f^*_{w'_{i,s-1}}(v)\}$ and $f^*_{w'_{i-1,s-1}} \preceq f^*_{w'_{i,s-1}}$, from $(*_0)$ we obtain the following inequality:

$$f^{c:h}(u) \succeq f^{c:h}(v) \ominus w'_{i,s-1}(u,v).$$

Now, since $w'_{i,j'}(u,v) \geq w'_{i,s-1}(u,v)$, it also holds $f^{c:h}(u) \succeq f^{c:h}(v) \ominus w'_{i,j'}(u,v)$. This proves that $u \notin \text{Inc}(f^{c:h}, i, j')$.

Case $u \in V_1 \setminus L^\iota_\top$ Since $u \in V_1 \setminus \text{Inc}(f^*_{w'_{i,s-1}}, i, s-1)$, for *all* $v \in N^{\text{out}}_\Gamma(u)$ it holds that:

$$f^*_{w'_{i,s-1}}(u) \succeq f^*_{w'_{i,s-1}}(v) \ominus w'_{i,s-1}(u,v).$$

By arguing as in the previous case, we obtain that for every $v \in N^{\text{out}}_\Gamma(u)$ the following holds: $f^{c:h}(u) \succeq f^{c:h}(v) \ominus w'_{i,s-1}(u,v)$. This proves that $u \notin \text{Inc}(f^{c:h}, i, j')$.

□

Proof of Item (4). The fact that `ua-jumps()` halts in finite time follows directly from Item 1 of Proposition 7.6 and the definition of the subprocedure `rejoin_ua-jump()` and that of `backtrack_ua-jump()`. □

Correctness of `solve_MPG()` (Algorithm 15)

As shown next, it turns out that (PC-1), (w-PC-2), (w-PC-3) are all satisfied by Algorithm 15.

Proposition 7.7. *Let $i \in [W^- - 1, W^+]$ and $j \in [1, s-1]$. The following two propositions hold.*

1. *Consider any invocation of `ei-jump`(i, J) (SubProcedure 8) at line 7 of Algorithm 15 such that $L^{inc} = \emptyset$. Then, (eij-PC-1), (eij-PC-2), (eij-PC-3), (eij-PC-4) are all satisfied w.r.t. Γ.*

2. *Consider any invocation of `J-VI`$(i, j, F, J, \Gamma[S])$ at line 11 of Algorithm 15. Then, (PC-1), (w-PC-2), (w-PC-3) are all satisfied w.r.t. the sub-arena $\Gamma[S]$.*

Proof. We prove Item 1 and 2 jointly, arguing by induction on the number k_1 of invocations of `ei-jump()` at line 7 of Algorithm 15 *and* the number k_2 of invocations of `J-VI()` at line 11.

Base Case: $k_1 = 1$ and $k_2 = 0$. So, the first subprocedure to be invoked is `ei-jump`(i, J) at line 7 of Algorithm 15, say at step ι. Notice that: $i^\iota = W^- - 1$; $\forall (v \in V)\, f^{c:\iota}(v) = 0$; $\forall (v \in V_0)\, J.\texttt{cnt}^\iota[v] = |N^{\text{out}}_\Gamma(v)|$ and $\forall (u \in V_0) \forall (v \in N^{\text{out}}_\Gamma(u))\, J.\text{cmp}^\iota[u,v] = \texttt{T}$; $L^\iota_f = L^{\text{inc}^\iota} = L^{\text{inc}\,\iota}_{\text{cpy}} = \emptyset$. Also notice that for every $u \in V$ and $v \in N^{\text{out}}_\Gamma(u)$ the following holds:

$$w'_{i^\iota, s-1}(u,v) = w'_{W^- - 1, s-1}(u,v) = w(u,v) - W^- \geq 0.$$

With this, it is straightforward to check that *(eij-PC-1)*, *(eij-PC-2)*, *(eij-PC-3)*, *(eij-PC-4)* hold.

Inductive Step: $k_1 = 1$ and $k_2 \geq 1$, or $k_1 > 1$. We need to check three cases.

1. Assume that `J-VI`$(i,j,F,J,\Gamma[S])$ is invoked at line 11 of Algorithm 15, say at step ι_1, soon after that `ua-jumps()` halted at line 9 of Algorithm 15. So, we aim at showing Item 2.

   Notice that: $j^{\iota_1} = 1$ holds (by line 10 of Algorithm 15). Let us check the (PC-1), (w-PC-2), (w-PC-3) w.r.t. $\Gamma[S]$. By line 4 of `backtrack_ua-jump()`, $S = L^{\iota}_{\top}$ for some step $\iota < \iota_1$.

   – *PC-1*: By line 2 of `backtrack_ua-jump()`, it holds $\forall(u \in L^{\iota}_{\top})\ L^{\iota}_{f}[u] = \bot$. By Item [3, (b), (i)] of Proposition 7.6, $\forall(v \in L^{\iota}_{\top})\ f^{\mathrm{c}:\iota_1}(v) = f^{*}_{w'_{i-1,s-1}}(v)$. By Lemma 7.2, $f^{*}_{w'_{i-1,s-1}} \preceq f^{*}_{w'_{i,j^{\iota_1}}}$. Therefore, $\forall(v \in L^{\iota}_{\top})\ f^{\mathrm{c}:\iota_1}(v) \preceq f^{*}_{w'_{i,j^{\iota_1}}}(v)$. This proves that (PC-1) holds w.r.t. $\Gamma[L^{\iota}_{\top}] = \Gamma[S]$.

   – *w-PC-2:* By lines 5-17 of `backtrack_ua-jumps()`, and since `init_cnt_cmp()` is correct, $L^{\mathrm{inc}^{\iota_1}} \subseteq \mathrm{Inc}(f^{\mathrm{c}:\iota_1},i,1)$. Since $j^{\iota_1} = 1$, then (w-PC-2) holds w.r.t. $\Gamma[L^{\iota}_{\top}] = \Gamma[S]$.

   – *w-PC-3:* By induction hypothesis and by Item [3, (b), (iii) and (iv)] of Proposition 7.6, J.`cnt`$^{\iota_1}$ and J.`cmp`$^{\iota_1}$ are coherent w.r.t. $f^{\mathrm{c}:\iota_1}$ in $\Gamma[L^{\iota}_{\top}]_{i,1}$; also, if $u \in V_1 \cap L^{\iota}_{\top}$ and $u \in \mathrm{Inc}(f^{\mathrm{c}:\iota_1},i,1)$, then $u \in L^{\mathrm{inc}^{\iota_1}}$ by lines 11-17 of `backtrack_ua-jump()`. Since $j^{\iota_1} = 1$, this proves that (PC-3) holds w.r.t. $\Gamma[L^{\iota}_{\top}] = \Gamma[S]$, so (w-PC-3) holds as well.

2. Assume that `J-VI`$(i,j,F,J,\Gamma[S])$ is invoked at line 11 of Algorithm 15, say at step ι_2, soon after that a previous invocation of `J-VI`$(i,j-1,F,J,\Gamma[S])$ halted at line 11 say at step ι_1. Notice that $j \in [2,s-1]$ in that case. Let us check (PC-1), (w-PC-2), (w-PC-3) w.r.t. $\Gamma[S]$. By line 4 of `backtrack_ua-jump()`, $S = L^{\iota}_{\top}$ for some step $\iota < \iota_1$.

   – *(PC-1)*: By lines 2-3 of `scl_back()` (which was executed at line 13 of Algorithm 15, just before ι_2), it holds $\forall(u \in L^{\iota}_{\top})\ L^{\iota_1}_{f}[u] = \bot$. By induction hypothesis and by Item 1 of Proposition 7.4, the following holds:

   $$\forall(v \in L^{\iota}_{\top})\ f^{\mathrm{c}:\iota_2}(v) = f^{*}_{w'_{i,j^{\iota_1}}}(v) = f^{*}_{w'_{i,j^{\iota_2}-1}}(v).$$

   By Lemma 7.2, $f^{*}_{w'_{i,j^{\iota_2}-1}} \preceq f^{*}_{w'_{i,j^{\iota_2}}}$. Therefore, $\forall(v \in L^{\iota}_{\top})\ f^{\mathrm{c}:\iota_2}(v) \preceq f^{*}_{w'_{i,j^{\iota_2}}}(v)$. Whence, (PC-1) holds w.r.t. $\Gamma[L^{\iota}_{\top}] = \Gamma[S]$.

   – *(w-PC-2):* By induction hypothesis and by Item 3 of Proposition 7.4, then:

   $$L^{\mathrm{inc}^{\iota_2}} = \{v \in V \mid 0 < f^{\mathrm{c}:\iota_2}(v) \neq \top\}.$$

   Thus, by Lemma 7.1, $L^{\mathrm{inc}^{\iota_2}} \subseteq \mathrm{Inc}(f^{\mathrm{c}:\iota_2},i,j^{\iota_2})$. So, (w-PC-2) holds w.r.t. $\Gamma[L^{\iota}_{\top}] = \Gamma[S]$.

   – *(w-PC-3):* Let $u \in V \setminus L^{\mathrm{inc}^{\iota_2}}$, and let $v \in N^{\mathrm{out}}_{\Gamma[L^{\iota}_{\top}]}(v)$.

If $u \in V_0$, we need to check the state of $J.\texttt{cmp}^{\iota_2}[(u,v)]$ and $J.\texttt{cnt}^{\iota_2}[u]$.

1. If $J.\texttt{cmp}^{\iota_2}[u,v] = \texttt{F}$, we argue that $(u,v) \in E$ is incompatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j}$. Indeed, it was already $J.\texttt{cmp}^{\iota_2}[(u,v)] = \texttt{F}$ when the previous $\texttt{J-VI}()$ (that invoked at step ι_1) halted. Then, by induction hypothesis and by Item 3 of Proposition 7.4, (u,v) is incompatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j-1}$. Thus, (u,v) is incompatible w.r.t. $f^{c:\iota_2}$ also in $\Gamma[S]_{i,j}$ (because $w'_{i,j} < w'_{i,j-1}$). So, $J.\texttt{cmp}^{\iota_2}[(u,v)]$ is coherent w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j}$.

2. If $J.\texttt{cmp}^{\iota_2}[(u,v)] = \texttt{T}$, we argue that either (u,v) is compatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j}$ or it holds that $v \in L^{\text{inc}^{\iota_2}}$. Indeed, assume that $J.\texttt{cmp}^{\iota_2}[(u,v)] = \texttt{T}$ and that (u,v) is incompatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j}$. Since $J.\texttt{cmp}^{\iota_2}[(u,v)] = \texttt{T}$, then it was as such even when the previous $\texttt{J-VI}()$ (that invoked at step ι_1) halted. So, (u,v) was compatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j-1}$. Since $u \notin L^{\text{inc}^{\iota_2}}$, then $f^{c:\iota_2}(u) = 0$ (indeed, if $f^{c:\iota_2}(u) = \top$, then (u,v) would have been compatible). Therefore, it is not possible that $f^{c:\iota_2}(v) = 0$; since, otherwise, from the fact that $f^{c:\iota_2}(u) = f^{c:\iota_2}(v) = 0$ and (u,v) is compatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j-1}$, it would be $w'(u,v)_{i,j-1} \geq 0$; and since $w(u,v) \in \mathbb{Z}$ and $0 < F_{j-1} < F_j \leq 1$ where $j \in [2, s-1]$, it would be $w'_{i,j}(u,v) \geq 0$ as well, so (u,v) would be compatible w.r.t. f^c in $\Gamma[S]_{i,j}$. Also, it is not possible that $f^{c:\iota_2}(v) = \top$, since otherwise (u,v) would have been incompatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j-1}$ (because $f^{c:\iota_2}(u) = 0$). Therefore, $0 < f^{c:\iota_2}(v) < \top$. Then, induction hypothesis and by Item 3 of Proposition 7.4, $v \in L^{\text{inc}^{\iota_2}}$.

3. By induction hypothesis and by Proposition 7.4, $f^{c:\iota_2}$ is the least-SEPM of $\Gamma[S]_{i,j-1}$ and $J.\texttt{cnt}^{\iota_2}$, $J.\texttt{cmp}^{\iota_2}$ are both coherent w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j-1}$. Therefore,

$$\begin{aligned} J.\texttt{cnt}^{\iota_2}[u] &= \left|\left\{v \in N^{\text{out}}_{\Gamma[S]}(u) \mid f^{c:\iota_2}(u) \succeq f^{c:\iota_2}(v) \ominus w'_{i,j-1}(u,v)\right\}\right| && \text{[by coherency of } J.\texttt{cnt}^{\iota_2}] \\ &= \left|\left\{v \in N^{\text{out}}_{\Gamma[S]}(u) \mid J.\texttt{cmp}^{\iota_2}[(u,v)] = \texttt{T}\right\}\right|. && \text{[by coherency of } J.\texttt{cmp}^{\iota_2}] \end{aligned}$$

Moreover, since $f^{c:\iota_2}$ is least-SEPM of $\Gamma[S]_{i,j-1}$, then u is consistent w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j-1}$, thus $J.\texttt{cnt}^{\iota_2}[u] > 0$.

If $u \in V_1$, assume (u,v) is incompatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j}$, for some $v \in N^{\text{out}}_{\Gamma[S]}(u)$.

We want to prove $v \in L^{\text{inc}^{\iota_2}}$. By induction hypothesis and by Proposition 7.4, $f^{c:\iota_2}$ is the least-SEPM of $\Gamma[S]_{i,j-1}$ and $L^{\text{inc}^{\iota_2}} = \{q \in V \mid 0 < f^{c:\iota_2}(q) \neq \top\}$. Thus, since $v \in V_1$, (u,v) is compatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j-1}$. Moreover, since $u \notin L^{\text{inc}^{\iota_2}}$ by hypothesis, then $f^{c:\iota_2}(u) = 0$ or $f^c(u) = \top$; but since (u,v) is incompatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j}$, it is $f^{c:\iota_2}(u) = 0$. Therefore, it is not possible that $f^{c:\iota_2}(v) = 0$; since, otherwise, from the fact that $f^{c:\iota_2}(u) = f^{c:\iota_2}(v) = 0$ and (u,v) is compatible w.r.t. $f^{c:\iota_2}$ in $\Gamma[S]_{i,j-1}$, it would be:

$$w'(u,v)_{i,j-1} = w(u,v) - i - F_{j-1} \geq 0,$$

since $w(u,v) \in \mathbb{Z}$ and $0 < F_{j-1} < F_j \leq 1$ where $j \in [2,s-1]$, it would be $w'_{i,j}(u,v) \geq 0$, so (u,v) would have been compatible w.r.t. f^{c} in $\Gamma[S]_{i,j}$. Also, it is not possible that $f^{\mathrm{c}:\iota_2}(v) = \top$, since otherwise (u,v) would have been incompatible w.r.t. $f^{\mathrm{c}:\iota_2}$ in $\Gamma[S]_{i,j}$ (because $f^{\mathrm{c}:\iota_2}(u) = 0$). Therefore, $0 < f^{\mathrm{c}:\iota_2}(v) < \top$. Then, induction hypothesis and by Item 3 of Proposition 7.4, $v \in L^{\mathrm{inc}^{\iota_2}}$.

3. Assume that `ei-jump`(i,J) is invoked at line 7 of Algorithm 15, say at step ι_1, and that $L^{\mathrm{inc}^{\iota_1}} = \emptyset$. Then, the following properties hold.

   – (eij-PC-1) $f^{\mathrm{c}:\iota_1}$ is the least-SEPM of $\Gamma_{i,s-1}$: indeed, consider the previous invocation `J-VI`$(i,j^{\iota_0},F,J,\Gamma[S^{\iota_0}])$ at line 11 of Algorithm 15, say it was invoked at step ι_0 before ι_1. By induction hypothesis and by Item 1 of Proposition 7.4, $f^{\mathrm{c}:\iota_1}$ is the least-SEPM of $\Gamma[S^{\iota_0}]_{i,j^{\iota_0}}$. Since $L^{\mathrm{inc}^{\iota_1}} = \emptyset$ by assumption, by induction hypothesis and Item 3 of Proposition 7.4, then $\{v \in S^{\iota_0} \mid 0 < f^{\mathrm{c}:\iota_1}(v) \neq \top\} = L^{\mathrm{inc}^{\iota_1}} = \emptyset$. We claim that $\forall(u \in S^{\iota_0})\, f^{\mathrm{c}:\iota_1}(u) = \top$. Indeed, the following holds.

If $u \in V_0$, since $f^{\mathrm{c}:\iota_1}$ is the least-SEPM of $\Gamma[S^{\iota_0}]_{i,j^{\iota_0}}$, there exists $v \in N^{\mathrm{out}}_{\Gamma[S^{\iota_0}]}(u)$ such that (u,v) is compatible w.r.t. $f^{\mathrm{c}:\iota_1}$ in $\Gamma[S^{\iota_0}]_{i,j^{\iota_0}}$. So, it is not possible that $f^{\mathrm{c}:\iota_1}(u) = 0$: otherwise, it would be $f^{\mathrm{c}:\iota_1}(v) = 0$ as well (because either $f^{\mathrm{c}:\iota_1}(v) = 0$ or $f^{\mathrm{c}:\iota_1}(v) = \top$), and since $w(u,v) \in \mathbb{Z}$ and $0 < F_{j^{\iota_0}} \leq 1$ where $j^{\iota_0} \in [1,s-1]$, then (u,v) would be compatible w.r.t. $f^{\mathrm{c}:\iota_1}$ even in $\Gamma[S^{\iota_0}]_{i,s-1}$, thus $f^*_{w'_{i,s-1}}(u) = 0$. But this contradicts the fact that, by induction hypothesis, Item 1 of Proposition 7.6 and Item 1 of Proposition 7.4, $f^*_{w'_{i,s-1}}(u) = \top$. Therefore, $f^{\mathrm{c}:\iota_1}(u) = \top$.

If $u \in V_1$, since $f^{\mathrm{c}:\iota_1}$ is the least-SEPM of $\Gamma[S^{\iota_0}]_{i,j^{\iota_0}}$, for every $v \in N^{\mathrm{out}}_{\Gamma[S^{\iota_0}]}(u)$, the arc (u,v) is compatible w.r.t. $f^{\mathrm{c}:\iota_1}$ in $\Gamma[S^{\iota_0}]_{i,j^{\iota_0}}$. Now, by arguing as in the previous case (i.e., $u \in V_0$), it holds that $\forall(u \in S^{\iota_0})\, f^{\mathrm{c}:\iota_1}(u) = \top$.

   Thus, $\forall(u \in S^{\iota_0})\, f^{\mathrm{c}:\iota_1}(u) = \top = f^*_{i,s-1}(u)$. By induction hypothesis and Item [3, (b), (ii)] of Proposition 7.6, $\forall(u \in V \setminus S^{\iota_0})\, f^{\mathrm{c}:\iota_1}(u) = f^*_{w'_{i,s-1}}(u)$. So, $f^{\mathrm{c}:\iota_1} = f^*_{w'_{i,s-1}}$.

   – (eij-PC-2) $L^{\mathrm{inc}^{\iota_1}} = \{v \in S^{\iota_0} \mid 0 < f^{\mathrm{c}:\iota_1}(v) \neq \top\}$: this holds by induction hypothesis and by Item 3 of Proposition 7.4.

   – (eij-PC-3) $L^{\mathrm{inc}^{\iota}}_{\mathrm{cpy}} \subseteq \mathrm{Inc}(f^{\mathrm{c}:\iota},i',j')$ for every $(i',j') > (i,s-1)$: this holds by induction hypothesis plus Item [3, (a)] of Proposition 7.6 and Lemma 7.1.

   – (eij-PC-4): consider the previous invocation of `J-VI`$(i,j^{\iota_0},F,J,\Gamma[S^{\iota_0}])$ at line 11 of Algorithm 15, say at step ι_0, just before ι_1. By induction hypothesis and by Item 2 of Proposition 7.4, for every $u \in V_0 \cap S^{\iota_0}$, $J.\mathtt{cnt}^{\iota_1}[u]$ and $J.\mathtt{cmp}^{\iota_1}[(u,\cdot)]$ are both coherent w.r.t. $f^{\mathrm{c}:\iota_1}$ in $\Gamma[S^{\iota_0}]_{i,j^{\iota_0}}$; also, for every $u \in V_0 \setminus S^{\iota_0}$, $J.\mathtt{cnt}^{\iota_1}[u]$ and $J.\mathtt{cmp}^{\iota_1}[(u,\cdot)]$ are both coherent w.r.t. $f^*_{w'_{i,s-1}}$ in $\Gamma_{i,s-1}$. Since (eij-PC-1) holds, then $f^{\mathrm{c}:\iota_1} = f^*_{w'_{i,s-1}}$. So, (eij-PC-4) holds.

□

Lemma 7.4. *Let $\hat{v} \in V$, assume $\texttt{val}^{\Gamma}(\hat{v}) = \hat{\imath} - F_{\hat{\jmath}-1}$, for some $\hat{\imath} \in [W^-, W^+]$ and $\hat{\jmath} \in [1, s-1]$.*

Then, eventually, Algorithm 15 invokes $\texttt{J-VI}(\hat{\imath}, \hat{\jmath}, F, J, \Gamma[S])$ at line 11, for some $S \subseteq V$.

Proof. For the sake of contradiction, for any $S \subseteq V$, assume that $\texttt{J-VI}(\hat{\imath}, \hat{\jmath}, F, J, \Gamma[S])$ is never invoked at line 11 of Algorithm 15. At each iteration of the main `while` loop of Algorithm 15 (lines 6-14), j is incremented (line 14); meanwhile, the value of i stands still until (eventually) `ei-jump`() and (possibly) `ua-jumps`() increase it (also resetting $j \leftarrow 1$). Therefore, since $\texttt{J-VI}(\hat{\imath}, \hat{\jmath}, F, J, \Gamma[S])$ is never invoked at line 11, there are $i_0 \in [W^-, W^+]$ and $j_0 \in [1, s-1]$, where $(i_0, j_0) < (\hat{\imath}, \hat{\jmath})$, such that one of the following two hold:

- Either $\texttt{ei-jump}(i_0, J)$ (line 7) is invoked and, when it halts say at step h, it holds $J.i^h > \hat{\imath}$.

  In that case, by Item 1 of Proposition 7.7 and Item 2 of Proposition 7.5, $f^*_{\hat{\imath},\hat{\jmath}-1}(\hat{v}) = f^*_{\hat{\imath},\hat{\jmath}}(\hat{v})$. On the other hand, since $\texttt{val}^{\Gamma}(\hat{v}) = \hat{\imath} - F_{\hat{\jmath}-1}$, then $\hat{v} \in \mathcal{W}_0(\Gamma_{\hat{\imath},\hat{\jmath}-1}) \cap \mathcal{W}_1(\Gamma_{\hat{\imath},\hat{\jmath}})$ by Theorem 6.3; so, by Lemma 6.3, $f^*_{\hat{\imath},\hat{\jmath}-1}(\hat{v}) \neq \top$ and $f^*_{\hat{\imath},\hat{\jmath}}(\hat{v}) = \top$. So, $\top \neq f^*_{\hat{\imath},\hat{\jmath}-1}(\hat{v}) = \top$; this is absurd.

- Or $\texttt{ua-jumps}(i_0, s, F, J, \Gamma)$ (line 9) is invoked and, when it halts say at step h, $J.i^h > \hat{\imath}$.

  In that case, during the execution of $\texttt{ua-jumps}(i_0, s, F, J, \Gamma)$, at some step $\hat{\imath}$, it is invoked $\texttt{J-VI}(\hat{\imath}, s-1, F, J, \Gamma)$ (line 2 of `ua-jumps`()); and when it halts, say at step $\hat{\imath}_h$, by Item 2 of Proposition 7.6 and by line 6 of `ua-jumps`(), then $L^{\hat{\imath}_h}_{\top} = \mathcal{W}_0(\Gamma_{\hat{\imath}-1,s-1}) \cap \mathcal{W}_1(\Gamma_{\hat{\imath},s-1}) = \emptyset$. Still, $\texttt{val}^{\Gamma}(\hat{v}) = \hat{\imath} - F_{\hat{\jmath}-1}$, then $\hat{v} \in \mathcal{W}_0(\Gamma_{\hat{\imath},\hat{\jmath}-1}) \cap \mathcal{W}_1(\Gamma_{\hat{\imath},\hat{\jmath}})$ by Theorem 6.3. Since $\rho = \{w_{i,j}\}_{i,j}$ is monotone decreasing, then $\mathcal{W}_0(\Gamma_{\hat{\imath},\hat{\jmath}-1}) \subseteq \mathcal{W}_0(\Gamma_{\hat{\imath}-1,s-1})$ and $\mathcal{W}_1(\Gamma_{\hat{\imath},\hat{\jmath}}) \subseteq \mathcal{W}_1(\Gamma_{\hat{\imath},s-1})$. Then, $\hat{v} \in \mathcal{W}_0(\Gamma_{\hat{\imath},\hat{\jmath}-1}) \cap \mathcal{W}_1(\Gamma_{\hat{\imath},\hat{\jmath}}) \subseteq \mathcal{W}_0(\Gamma_{\hat{\imath}-1,s-1}) \cap \mathcal{W}_1(\Gamma_{\hat{\imath},s-1}) = \emptyset$, but this is absurd.

In either case, we arrive at some contradiction.

Therefore, eventually, Algorithm 15 invokes `J-VI`() at line 11 on input $(\hat{\imath}, \hat{\jmath})$. □

Theorem 7.1. *Given any input MPG $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$, Algorithm 15 halts in finite time.*

If $(\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma^*_0)$ is returned, then: $\mathcal{W}_0$ is the winning set of Player 0 in Γ, $\mathcal{W}_1$ is that of Player 1, $\forall^{v \in V} \nu(v) = \texttt{val}^{\Gamma}(v)$, σ^*_0 is an optimal positional strategy for Player 0 in Γ.

Proof. Firstly, we argue that Algorithm 15 halts in finite time. Recall, by Propositions [7.4, 7.5, 7.6, 7.7], it holds that any invocation of `J-VI()`, `ei-jump()`, `ua-jumps()` (respectively) halts in finite time. It is easy to check at this point that (by lines 14 of Algorithm 15, line 4 and 8 of `ei-jump()`, line 5 of `ua-jumps()`, and since L_ω was sorted in increasing order), whenever `J-VI()` is invoked at line 11 of Algorithm 15 – at any two consequential steps ι_0, ι_1 (i.e., such that $\iota_0 < \iota_1$) – then $(i^{\iota_0}, j^{\iota_0}) < (i^{\iota_1}, j^{\iota_1})$. Also, by Propositions 7.4-7.7, whenever `J-VI()` is invoked at line 11 of Algorithm 15, if it halts say at step ι_h, then $f^{c:\iota_h}$ is the least-SEPM of $\Gamma_{i^{\iota_h}, j^{\iota_h}}$; so, eventually, say when $(i^{\hat{\iota}_h}, j^{\hat{\iota}_h})$ are sufficiently large, then $\forall^{u \in V} f^{c:\hat{\iota}_h}(u) = \top$; and, by Item 3 of Proposition 7.4, $L^{\text{inc}^{\hat{\iota}_h}} = \{v \in V \mid 0 < f^{c:\hat{\iota}_h} \neq \top\}$, so, $L^{\text{inc}^{\hat{\iota}_h}} = \emptyset$. Consider the first invocation of `ei-jump()` (line 7 of Algorithm 15) that is made soon after this $\hat{\iota}_h$, and say it halts at step h. By Item 3 of Proposition 7.5, $L^{\text{inc}^h} = \text{Inc}(f^{c:h}, J.i^h, 1)$. Since $\forall^{u \in V} f^{c:\hat{\iota}_h}(u) = f^{c:h}(u) = \top$, then $\text{Inc}(f^{c:h}, J.i^h, 1) = \emptyset$. Therefore, $L^{\text{inc}^h} = \emptyset$. Therefore, Algorithm 15 halts at line 8 soon after h.

Secondly, we argue that Algorithm 15 returns $(\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma_0^*)$ correctly.

On one side, $\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma_0^*$ are accessed only when `set_vars()` is invoked (line 12 of Algorithm 15). Just before that, at line 11, some `J-VI()` must have been invoked; say it halts at step h. By Items 1 and 2 of Proposition 7.6, $L^h_\top = \mathcal{W}_0(\Gamma_{i^h, j^h - 1}) \cap \mathcal{W}_1(\Gamma_{i^h, j^h})$. Therefore, by Theorem 6.3, ν is assigned correctly; so, $\mathcal{W}_0, \mathcal{W}_1$ are also assigned correctly. At this point, by Theorem 6.4, also σ_0^* is assigned correctly.

Conversely, let $\hat{v} \in V$ and assume $\texttt{val}^\Gamma(v) = \hat{\imath} - F_{\hat{\jmath}-1}$ for some $\hat{\imath} \in [W^-, W^+]$ and $\hat{\jmath} \in [1, s-1]$. By Lemma 7.4, eventually, Algorithm 15 invokes `J-VI`$(\hat{\imath}, \hat{\jmath}, F, J, \Gamma)$ at line 11. By Items 1 and 2 of Proposition 7.6, when `J-VI`$(\hat{\imath}, \hat{\jmath}, F, J, \Gamma)$ halts, say at step h, it holds that $L^h_\top = \mathcal{W}_0(\Gamma_{\hat{\imath}, \hat{\jmath}-1}) \cap \mathcal{W}_1(\Gamma_{\hat{\imath}, \hat{\jmath}})$. Therefore, soon after at line 12, `set_vars()` assigns to $\mathcal{W}_0, \mathcal{W}_1, \nu, \sigma_0^*$ a correct state. □

7.3.3 Complexity of Algorithm 15

The complexity of Algorithm 15 follows, essentially, from the fact that [*Inv-EI*] is satisfed.

Proposition 7.8. *Algorithm 15 satisfies [*Inv-EI*]: whenever a Scan-Phase is executed (each time that a Value-Iteration is invoked), an energy-level $f(v)$ strictly increases for at least one $v \in V$. So, the energy-lifting operator δ is applied (successfully) at least once per each `J-VI()`.*

Proof. By lines 1 and 9 of `ei-jump()` (SubProcedure 8), lines 1-6 of `ua-jumps()`, and line 8 of Algorithm 15, whenever `J-VI()` is invoked either at line 11 of Algorithm 15 or at line 2 of `ua-jumps()` (SubProcedure 9), say at step ι, then $L^{\text{inc}^\iota} \neq \emptyset$. Moreover, by Proposition 7.7, by Item 3 of Propositions 7.4 and Lemma 7.1, $L^{\text{inc}^\iota} \subseteq \text{Inc}(f^{c:\iota}, i^\iota, j^\iota)$. Therefore, during each `J-VI()` that is possibly invoked by Algorithm 15, at least one application of δ is performed (line 2-3 of `J-VI()`) (because $L^{\text{inc}^\iota} \neq \emptyset$) and every single application of $\delta(f^c, v)$ that is made during `J-VI()`, say at step $\hat{\imath}$, for any $v \in V$, really increases $f^{c:\hat{\imath}}(v)$ (because $L^{\text{inc}^{\hat{\imath}}} \subseteq \text{Inc}(f^{c:\hat{\imath}}, i^{\hat{\imath}}, j^{\hat{\imath}})$). $\square$

Theorem 7.2. *Given an input MPG $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$, Algorithm 15 halts within the following time bound:*

$$O(|E|\log|V|) + \Theta\Big(\sum_{v \in V} deg_\Gamma(v) \cdot \ell^1_\Gamma(v)\Big) = O(|V|^2|E|W),$$

The working space is $\Theta(|V| + |E|)$.

Proof. The initialization of L_ω takes $O(|E|\log|V|)$ time, i.e., the cost for sorting $\{w_e \mid e \in W\}$. Each single application of δ, which is possibly done during any execution of `J-VI()` throughout Algorithm 15, it takes time $\Theta(\mathtt{deg}_\Gamma(v))$. Indeed, by Proposition 7.1, the total aggregate time spent for all applications of δ in Algorithm 15 is $\Theta\big(\sum_{v\in V} \mathtt{deg}_\Gamma(v) \cdot \ell^1_\Gamma(v)\big)$. It is not difficult to check from the description of Algorithm 15, at this point, that the time spent between any two subsequent applications of δ can increase the total time amount $\sum_{v\in V} \mathtt{deg}_\Gamma(v) \cdot \ell^1_\Gamma(v)$ of Algorithm 15 only by a constant factor. Notice that the aggregate total cost of all the invocations of `repair()` is $O(|E|)$. In Section 6.3 it was shown how to generate $\mathcal{F}_{|V|}$ iteratively, one term after another, in $O(1)$ time-delay and $O(1)$ total space, as in [97]. It is also easy to check, at this point, that Algorithm 15 works with $\Theta(|V| + |E|)$ space. $\square$

7.3.4 An Experimental Evaluation of Algorithm 15

This section describes an empirical evaluation of Algorithm 15. All algorithms and procedures employed in this practical evaluation have been implemented in C/C++ and executed on a Linux machine having the following characteristics:

– Intel Core i5-4278U CPU @ 2.60GHz x2;
– 3.8GB RAM;
– Ubuntu 15.10 Operating System.

The main goal of this experiment was: (i) to determine the average computation time of Algorithm 15, with respect to randomly-generated MPGs, in order to give an idea of the practical behavior it; (ii) to offer an experimental comparison between Algorithm 15 and the algorithm which is offered in [38], i.e., Algorithm 16, in order to give evidence and experimental confirmation of the algorithmic improvements made over [38]. Here we propose a summary of the obtained results presenting a brief report about, Test 1, Test 2.

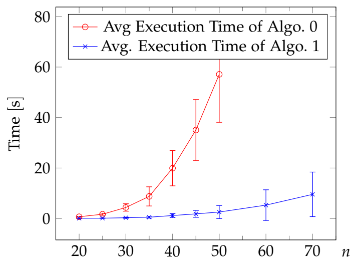
In all of our tests, in order to generate a suitable dataset of MPGs, our choice has been to use the `randomgame` procedure of `pgsolver` suite [98], that can produce random arenas instances for any given number of nodes. We exploited `randomgame` as follows:

$\|V\|$	μ (sec)	σ
20	0.69	0.21
25	1.69	0.43
30	4.37	1.47
35	8.77	3.79
40	19.95	6.99
45	35.06	12.0
50	57.10	18.9

(a) Test 1, μ, Algo-0.

$\|V\|$	μ (sec)	σ
20	0.09	0.05
25	0.13	0.07
30	0.30	0.24
35	0.52	0.39
40	1.18	0.77
45	1.83	1.37
50	2.56	2.59
60	5.30	6.08
70	9.57	8.83

(b) Test 1, μ, Algo-1.



(c) Interpolation of average execution times in Test 1 for Algo-0 (red, mark=o) and Algo-1 (blue, mark=x).

Figure 7.3: Results of Test 1 on Average Execution Time

1. First, `randomgame` was used to generate random directed graphs, with out-degree taken uniformly at random in $[1, |V|]$;

2. Then, the resulting graphs were translated into MPGs by weighting each arc with an integer randomly chosen in the interval $[-W, W]$, where W was chosen accordingly to the test type;

With such settings, the resulting MPGs are characterized by $|V|$ and W.

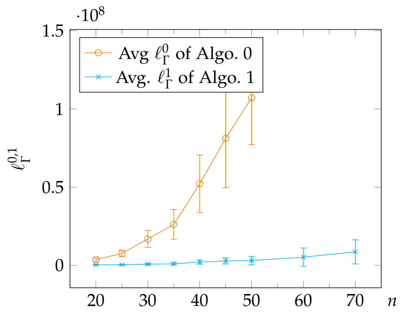
In Test 1 the average computation time was determined for different orders of $|V|$. For each $n \in \{20, 25, 30, 35, 40, 45, 50\}$, 25 MPGs instances with maximum weight $W = 100$ were generated by `randomgame`. Each instance had been solved both with Algorithm 16 and Algorithm 1. In addition, to experiment a little further on Algorithm 15, for each $n \in \{60, 70\}$, 25 MPGs instances with maximum weight (fixed to) $W = 100$ were also generated by `randomgame` and solved only with Algorithm 1. The results of the test are summarized in Fig. 7.3, where each execution mean time is depicted as a point with a vertical bar representing its confidence interval determined according to its std-dev. As shown by Fig. 7.3, Test 1 gives experimental evidence of the supremacy of Algorithm 15 over Algorithm 16. In order to provide a better insight on

$\|V\|$	$\mu\ (\ell^0_\Gamma)$	σ
20	3.71E+06	1.05E+06
25	7.67E+06	2.09E+06
30	1.69E+07	5.35E+06
35	2.62E+07	9.52E+06
40	5.22E+07	1.84E+07
45	8.12E+07	3.16E+07
50	1.07E+08	2.99E+07

(a) Test 1, ℓ^0_Γ, Algo-0.

$\|V\|$	$\mu\ (\ell^1_\Gamma)$	σ
20	3.53E+05	2.04E+05
25	4.45E+05	2.17E+05
30	7.92E+05	5.83E+05
35	9.91E+05	7.53E+05
40	2.13E+06	1.34E+06
45	2.88E+06	1.98E+06
50	3.08E+06	2.72E+06
60	5.28E+06	5.82E+06
70	8.66E+06	7.71E+06

(b) Test 1, ℓ^1_Γ, Algo-1.



(c) Interpolation of avgerage values of $\ell^0_\Gamma, \ell^1_\Gamma$ in Test 1 for Algo-0 (orange, mark=o) and Algo-1 (cyan, mark=x).

Figure 7.4: Results of Test 1 on $\ell^0_\Gamma, \ell^1_\Gamma$

the behavior of the algorithms, a comparison between the values of ℓ^0_Γ and ℓ^1_Γ is offered in Fig. 7.4. Test 1 confirms that $\ell^1_\Gamma \ll \ell^0_\Gamma$ (by a factor $\geq 10^2$ when $|V| \geq 50$) on randomly generated MPGs. The numerical results of Table 7.3a-7.3b suggest that the std-dev of both the avgerage running time of Algorithm 1 and of ℓ^1_Γ is greater (in proportion) than that of Algorithm 16 and ℓ^0_Γ; but thinking about it this actually turns out to be a benefit: as a certain proportion of MPGs instances can now exhibit quite a smaller value of ℓ^1_Γ, then the running time improves, but the std-dev fluctuates more meanwhile.

In Test 2 the average computation time was determined for different orders of W. For each $W \in \{50,100,150,200,250,300,350\}$, 25 MPGs instances with maximum weight W, and $|V| = 25$ (fixed), were generated by `randomgame`. Each instance had been solved both with Algorithm 16 and Algorithm 1. The results of the test are summarized in Fig. 7.5 and Fig. 7.6, where each execution mean time and $\ell^{0,1}_\Gamma$ is depicted as a point with a vertical bar representing its confidence interval determined according to its std-dev.

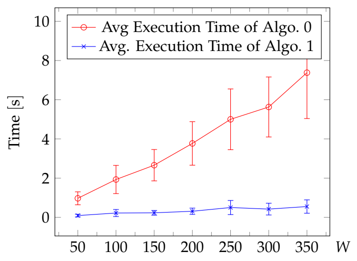
In summary our experiments suggest that, even in practice, Algorithm 15 is significantly faster than the Algorithm 16 devised in [35,38].

W	μ (sec)	σ
50	0.97	0.33
100	1.93	0.72
150	2.66	0.80
200	3.77	1.11
250	5.00	1.55
300	5.63	1.53
350	7.38	2.34

(a) Test 2, μ, Algo-0.

W	μ (sec)	σ
50	0.09	0.07
100	0.22	0.18
150	0.23	0.12
200	0.31	0.16
250	0.50	0.36
300	0.42	0.30
350	0.55	0.34

(b) Test 2, μ, Algo-1.



(c) Interpolation of average execution times in Test 2 for Algo-0 (red, mark=o) and Algo-1 (blue, mark=x).

Figure 7.5: Results of Test 2 on Average Execution Time

7.4 Conclusion

We offered a faster $O(|E|\log|V|)+\Theta(\sum_{v\in V}\deg_\Gamma(v)\cdot\ell(v))=O(|V|^2|E|W)$ time energy algorithm for the Value Problem and Optimal Strategy Synthesis in MPGs. The result was achieved by introducing a novel algorithmic scheme based on so-called *EI* and *UA* Jumps.

We ask whether the least-SEPM of reweighted EGs of the kind Γ^{w-q}, for $q\in S_\Gamma$, can be computed in $o(|V|^2|E|W)$ time: this could lead to an improved time complexity upper bound for MPGs (hopefully matching the time spent for solving EGs). To conclude, it would be interesting to adapt Algorithm 16 so that to work within the strategy-improvement framework, instead of value-iteration, because the former seems to exhibit a faster converge in practice.

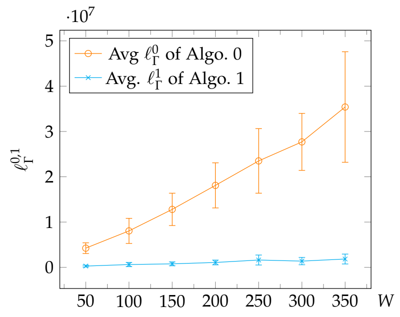
Many questions remain open on this way.

W	μ (ℓ^0_Γ)	σ
50	4.24E+06	1.18E+06
100	8.05E+06	2.77E+06
150	1.28E+07	3.57E+06
200	1.81E+07	4.98E+06
250	2.35E+07	7.13E+06
300	2.77E+07	6.29E+06
350	3.54E+07	1.22E+07

(a) Test 2, ℓ^0_Γ, Algo-0.

W	μ (ℓ^1_Γ)	σ
50	2.84E+05	2.02E+05
100	6.21E+05	4.62E+05
150	7.75E+05	4.24E+05
200	1.08E+06	5.22E+05
250	1.62E+06	1.10E+06
300	1.38E+06	8.07E+05
350	1.84E+06	1.09E+06

(b) Test 2, ℓ^1_Γ, Algo-1.



(c) Interpolation of avgerage values of $\ell^0_\Gamma, \ell^1_\Gamma$ in Test 2 for Algo-0 (orange, mark=o) and Algo-1 (cyan, mark=x).

Figure 7.6: Results of Test 2 on $\ell^0_\Gamma, \ell^1_\Gamma$

8 The Energy Structure of Optimal Positional Strategies in MPGs

Chapter Abstract

This chapter studies structural aspects concerning Optimal Positional Strategies (OPSs) in Mean Payoff Games (MPGs), it's a contribution to understanding the relationship between OPSs in MPGs and Small Energy-Progress Measures (SEPMs) in reweighted Energy Games (EGs). Firstly, it is observed that the space of all OPSs, $\texttt{opt}_\Gamma\Sigma^M_0$, admits a *unique complete decomposition* in terms

of so-called *extremal*-SEPMs in reweighted EGs; this points out what we called the "Energy-Lattice $\mathcal{X}^*_\Gamma$ of $\texttt{opt}_\Gamma\Sigma^M_0$". Secondly, it is offered a *pseudo-polynomial total-time* recursive procedure for *enumerating* (w/o repetitions) all the elements of $\mathcal{X}^*_\Gamma$, and for computing the corresponding partitioning of $\texttt{opt}_\Gamma\Sigma^M_0$. It is observed that the corresponding recursion tree defines an additional lattice $\mathcal{B}^*_\Gamma$, whose elements are certain subgames $\Gamma' \subseteq \Gamma$ that we call *basic* subgames. The extremal-SEPMs of a given MPG Γ coincide with the least-SEPMs of the basic subgames of Γ; so, $\mathcal{X}^*_\Gamma$ is the energy-lattice comprising all and only the *least*-SEPMs of the *basic* subgames of Γ. The complexity of the proposed enumeration for both $\mathcal{B}^*_\Gamma$ and $\mathcal{X}^*_\Gamma$ is $O(|V|^3|E|W|\mathcal{B}^*_\Gamma|)$ total time and $O(|V||E|) + \Theta(|E||\mathcal{B}^*_\Gamma|)$ working space. Finally, it is constructed an MPG Γ for which $|\mathcal{B}^*_\Gamma| > |\mathcal{X}^*_\Gamma|$, this shows that $\mathcal{B}^*_\Gamma$ and $\mathcal{X}^*_\Gamma$ are not isomorphic.

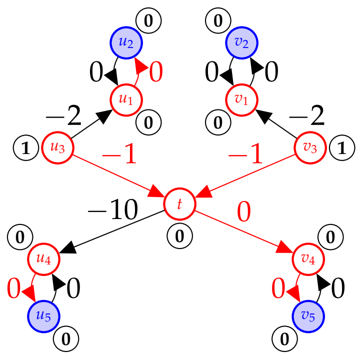


Figure 8.1: An MPG Γ_d for which $|\mathcal{B}^*_\Gamma| > |\mathcal{X}^*_\Gamma|$.

This chapter is a revised version of [35].

8.1 Introduction

This chapter studies the relationship between Optimal Positional Strategies (OPSs) in MPGs and Small Energy-Progress Measures (SEPMs) in reweighted EGs. Actually this is an extended and revised version of Section 5 in [35].

8.1.1 Contribution

This chapter contributes in the following way.

1. *An Energy-Lattice Decomposition of the Space of Optimal Positional Strategies in MPGs.*

Let's denote by $\texttt{opt}_{\Gamma}\Sigma^M_0$ the space of all the optimal positional strategies in a given MPG Γ. What allows the algorithms given in [35, 37, 38] to compute at least one $\sigma^*_0 \in \texttt{opt}_{\Gamma}\Sigma^M_0$ is a *compatibility* relation that links optimal arcs in MPGs to arcs that are *compatible* w.r.t. least-SEPMs in reweighted EGs. The family $\mathcal{E}_\Gamma$ of all SEPMs of a given EG Γ forms a complete finite lattice, the Energy-Lattice of the EG Γ. Firstly, we observe that even though compatibility w.r.t. *least*-SEPMs in reweighted EGs implies optimality of positional strategies in MPGs (see Theorem 6.4), the converse doesn't hold generally (see Proposition 8.1). Thus a natural question was whether compatibility w.r.t. SEPMs was really appropriate to capture (e.g., to provide a recursive enumeration of) the whole $\texttt{opt}_{\Gamma}\Sigma^M_0$ and not just a proper subset of it. Partially motivated by this question we explored on the relationship between $\texttt{opt}_{\Gamma}\Sigma^M_0$ and $\mathcal{E}_\Gamma$. In Theorem 8.2, it is observed a unique complete decomposition of $\texttt{opt}_{\Gamma}\Sigma^M_0$ which is expressed in terms of so called *extremal*-SEPMs in reweighted EGs. This points out what we called the "Energy-Lattice $\mathcal{X}^*_\Gamma$ associated to $\texttt{opt}_{\Gamma}\Sigma^M_0$", the family of all the extremal-SEPMs of a given MPG Γ. So, compatibility w.r.t. SEPMs actually turns out to be appropriate for constructing the whole $\texttt{opt}_{\Gamma}\Sigma^M_0$; but an entire lattice $\mathcal{X}^*_\Gamma$ of extremal-SEPMs then arises (and not just the least-SEPM, which turns out to account only for the join/top component of $\texttt{opt}_{\Gamma}\Sigma^M_0$).

2. *A Recursive Enumeration of Extremal-SEPMs and Optimal Positional Strategies in MPGs.*

It is offered a pseudo-polynomial total time recursive procedure for enumerating (w/o repetitions) all the elements of $\mathcal{X}^*_\Gamma$, and for computing the associated partitioning of $\texttt{opt}_{\Gamma}\Sigma^M_0$. This shows that the above mentioned compatibility relation is appropriate so to extend the algorithm given in [37], recursively, in order to compute the whole $\texttt{opt}_{\Gamma}\Sigma^M_0$ and $\mathcal{X}^*_\Gamma$. It is observed that the corresponding recursion tree actually defines an additional lattice $\mathcal{B}^*_\Gamma$, whose elements are certain subgames $\Gamma' \subseteq \Gamma$ that we call *basic* subgames. The extremal-SEPMs of a given Γ coincide with the least-SEPMs of the basic subgames of Γ; so, $\mathcal{X}^*_\Gamma$ is the energy-lattice comprising all and only the *least*-SEPMs of the *basic* subgames of Γ. The total time complexity of the proposed enumeration for both $\mathcal{B}^*_\Gamma$ and $\mathcal{X}^*_\Gamma$ is $O(|V|^3|E|W|\mathcal{B}^*_\Gamma|)$, it works in space $O(|V||E|) + \Theta\big(|E||\mathcal{B}^*_\Gamma|\big)$. An example of MPG Γ for which $|\mathcal{B}^*_\Gamma| > |\mathcal{X}^*_\Gamma|$ ends this chapter.

8.1.2 Organization

The following Section 6.2 introduces some notation and provides the required background on infinite 2-player pebble games and related algorithmic results. In Section 6.3, a suitable relation between values, optimal strategies, and certain reweighting operations is recalled from [35,38]. Section 8.2 offers a unique and complete energy-lattice decomposition of $\texttt{opt}_{\Gamma}\Sigma_0^M$. Finally, Section 8.3 provides a recursive enumeration of $\mathcal{X}_{\Gamma}^*$ and the corresponding partitioning of $\texttt{opt}_{\Gamma}\Sigma_0^M$.

8.2 An Energy-Lattice Decomposition of $\texttt{opt}_{\Gamma}\Sigma_0^M$

Consider the example arena Γ_{ex} shown in Fig. 8.2.

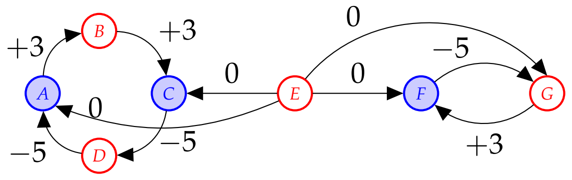


Figure 8.2: An arena $\Gamma_{\text{ex}} = \langle V, \mathsf{E}, w, (V_0, V_1)\rangle$. Here, $V = \{A,B,C,D,E,F,G\}$ and $\mathsf{E} = \{(A,B,+3),(B,C,+3),(C,D,-5),(D,A,-5),(E,A,0),(E,C,0),(E,F,0),(E,G,0),(F,G,-5),(G,F,+3)\}$. Also, $V_0 = \{B,D,E,G\}$ is colored in red, while $V_1 = \{A,C,F\}$ is filled in blue.

It is easy to see that $\forall^{v\in V}\texttt{val}^{\Gamma_{\text{ex}}}(v) = -1$. Indeed, Γ_{ex} contains only two cycles, i.e., $C_L = [A,B,C,D]$ and $C_R = [F,G]$, also notice that $w(C_L)/C_L = w(C_R)/C_R = -1$. The least-SEPM f^* of the reweighted EG Γ_{ex}^{w+1} can be computed by running a Value Iteration [14]. Taking into account the reweighting $w \rightsquigarrow w+1$, as in Fig. 8.3: $f^*(A) = f^*(E) = f^*(G) = 0$, $f^*(B) = f^*(D) = f^*(F) = 4$, and $f^*(C) = 8$.

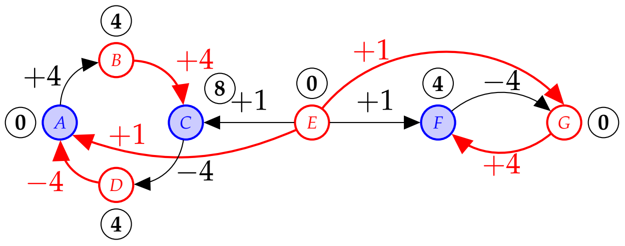


Figure 8.3: The least-SEPM f^* of Γ_{ex}^{w+1} (energy-levels are depicted in circled boldface). All and only those arcs of Player 0 that are compatible with f^* are $(B,C),(D,A),(E,A),(E,G),(G,F)$ (thick red arcs).

So, Γ_{ex} (Fig. 8.3) implies the following.

Proposition 8.1. *The converse statement of Theorem 6.4 doesn't hold; there exist infinitely many MPGs Γ having at least one $\sigma_0 \in \texttt{opt}_{\Gamma}\Sigma_0^M$ which is not compatible with the least-SEPM of Γ.*

Proof. Consider the Γ_{ex} of Fig. 8.3, and the least-SEPM f^* of the EG Γ_{ex}^{w+1}. The only vertex at which Player 0 really has a choice is E. Every arc going out of E is optimal in the MPG Γ_{ex}: whatever arc $(E,X) \in \mathsf{E}$ (for any $X \in \{A,C,F,G\}$) Player 0 chooses at E, the resulting payoff equals $\mathtt{val}^{\Gamma_{\text{ex}}}(E) = -1$. Let f^* be the least-SEPM of f^* in Γ_{ex}^{w+1}. Observe, (E,C) and (E,F) are not compatible with f^* in Γ_{ex}^{w+1}, only (E,A) and (E,G) are. For instance, the positional strategy $\sigma_0 \in \Sigma_0^M$ defined as $\sigma_0(E) \triangleq F$, $\sigma_0(B) \triangleq C$, $\sigma_0(D) \triangleq A$, $\sigma_0(G) \triangleq F$ ensures a payoff $\forall^{v\in V}\mathtt{val}^{\Gamma_{\text{ex}}}(v) = -1$, but it is not compatible with the least-SEPM f^* of Γ_{ex}^{w+1} (because $f^*(E) = 0 < 3 = f^*(F) \ominus w(E,F)$). It is easy to turn the Γ_{ex} of Fig. 8.3 into a family on infinitely many similar examples. □

We now aim at strengthening the relationship between $\text{opt}_\Gamma\Sigma_0^M$ and the Energy-Lattice $\mathcal{E}_\Gamma$. For this, we assume *wlog* $\exists^{\nu\in\mathbf{Q}}\forall^{v\in V}\mathtt{val}^\Gamma(v) = \nu$; this follows from Theorem 8.1, i.e., a refined formulation of the determinacy theorem offered in [8].

Theorem 8.1 ([8]). *Let Γ be an MPG and let $\{C_i\}_{i=1}^m$ be a partition (called* ergodic*) of its vertices into $m \geq 1$ classes each one having the same optimal value $\nu_i \in \mathbf{Q}$. Formally, $V = \bigsqcup_{i=1}^m C_i$ and $\forall^{i\in[m]}\forall^{v\in C_i}\mathtt{val}^{\Gamma_i}(v) = \nu_i$, where $\Gamma_i \triangleq \Gamma_{|C_i}$.*

Then, Player 0 has no vertices with outgoing arcs leading from C_i to C_j whenever $\nu_i < \nu_j$, and Player 1 has no vertices with outgoing arcs leading from C_i to C_j whenever $\nu_i > \nu_j$; moreover, there exist $\sigma_0 \in \Sigma_0^M$ and $\sigma_1 \in \Sigma_1^M$ such that:

– If the game starts from any vertex in C_i, then σ_0 secures a gain at least ν_i to Player 0 and σ_1 secures a loss at most ν_i to Player 1;

– Any play that starts from C_i always stays in C_i, if it is consistent with both strategies σ_0,σ_1, i.e., if Player 0 plays according to σ_0, and Player 1 according to σ_1.

By Theorem 8.1 we can study $\mathtt{opt}_{\Gamma_i}\Sigma_0^M$, independently w.r.t. $\mathtt{opt}_{\Gamma_j}\Sigma_0^M$ for $j \neq i$.

We say that an MPG Γ is *ν-valued* if and only if $\exists^{\nu\in\mathbf{Q}}\forall^{v\in V}\mathtt{val}^\Gamma(v) = \nu$.

Given an MPG Γ and $\sigma_0 \in \Sigma_0^M(\Gamma)$, recall, $G(\Gamma,\sigma_0) \triangleq (V,E',w')$ is obtained from G^Γ by deleting all and only those arcs that are not part of σ_0, i.e.,

$$E' \triangleq \big\{(u,v) \in E \mid u \in V_0 \text{ and } v = \sigma_0(u)\big\} \cup \big\{(u,v) \in E \mid u \in V_1\big\},$$

where each $e \in E'$ is weighted as in Γ, i.e., $w' : E' \to \mathbb{Z} : e \mapsto w_e$.

When $G = (V,E,w)$ is a weighted directed graph, a *feasible-potential (FP)* for G is any map $\pi : V \to \mathcal{C}_G$ s.t. $\forall^{u\in V}\forall^{v\in N^{\text{out}}(u)}\pi(u) \succeq \pi(v) \ominus w(u,v)$. The *least*-FP $\pi^* = \pi^*_G$ is the (unique) FP s.t., for any other FP π, it holds $\forall^{v\in V}\pi^*(v) \preceq \pi(v)$. Given G, the Bellman-Ford algorithm can be used to produce π^*_G in $O(|V||E|)$ time. Let $\pi^*_{G(\Gamma,\sigma_0)}$ be the *least-FP* of $G(\Gamma,\sigma_0)$. Notice, for every $\sigma_0 \in \Sigma_0^M$, the least-FP $\pi^*_{G(\Gamma,\sigma_0)}$ is actually a SEPM for the EG Γ; still it can differ from the least-SEPM of Γ, due to σ_0. We consider the following family of strategies.

Definition 8.1 ($\Delta_0^M(f,\Gamma)$-Strategies). *Let $\Gamma = \langle V,E,w,(V_0,V_1)\rangle$ and let $f : V \to \mathcal{C}_\Gamma$ be a SEPM for the EG Γ. Let $\Delta_0^M(f,\Gamma) \subseteq \Sigma_0^M(\Gamma)$ be the family of all and only those* *positional strategies of Player 0 in Γ s.t. $\pi^*_{G(\Gamma,\sigma_0)}$ coincides with f pointwisely, i.e.,*

$$\Delta_0^M(f,\Gamma) \triangleq \Big\{\sigma_0 \in \Sigma_0^M(\Gamma) \mid \forall^{v\in V}\, \pi^*_{G(\Gamma,\sigma_0)}(v) = f(v)\Big\}.$$

We now aim at exploring further on the relationship between $\mathcal{E}_\Gamma$ and $\texttt{opt}_\Gamma\Sigma_0^M$.

Definition 8.2 (The Energy-Lattice of $\texttt{opt}_\Gamma\Sigma_0^M$)**.** *Let Γ be a ν-valued MPG. Let $\mathcal{X} \subseteq \mathcal{E}_{\Gamma^{w-\nu}}$ be a sublattice of SEPMs of the reweighted EG $\Gamma^{w-\nu}$.*

*We say that $\mathcal{X}$ is an "*Energy-Lattice *of $\texttt{opt}_\Gamma\Sigma_0^M$" iff $\forall^{f\in\mathcal{X}}\Delta_0^M(f,\Gamma^{w-\nu}) \neq \emptyset$ and the following disjoint-set decomposition holds:*

$$\texttt{opt}_\Gamma\Sigma_0^M = \bigsqcup_{f\in\mathcal{X}} \Delta_0^M(f,\Gamma^{w-\nu}).$$

Lemma 8.1. *Let Γ be a ν-valued MPG, and let $\sigma_0^* \in \texttt{opt}_\Gamma\Sigma_0^M$. Then, $G(\Gamma^{w-\nu},\sigma_0^*)$ is* conservative *(i.e., it contains* no *negative cycle).*

Proof. Let $C \triangleq (v_1\ldots,v_k,v_1)$ by any cycle in $G(\Gamma^{w-\nu},\sigma_0^*)$. Since we have $\sigma_0^* \in \texttt{opt}_\Gamma\Sigma_0^M$ and $\forall^{v\in V}\texttt{val}^\Gamma(v) = \nu$, thus by Lemma 6.1:

$$w(C)/k = \frac{1}{k}\sum_{i=1}^{k} w(v_i,v_{i+1}) \geq \nu(\text{for } v_{k+1} \triangleq v_1),$$

so that, assuming $w' \triangleq w - \nu$, then:

$$w'(C)/k = \frac{1}{k}\sum_{i=1}^{k}\big(w(v_i,v_{i+1}) - \nu\big) = w(C)/k - \nu \geq \nu - \nu = 0.$$

□

Some aspects of the following Proposition 8.2 rely heavily on Theorem 6.4: the compatibility relation comes again into play. Moreover, we observe that Proposition 8.2 is equivalent to the following fact, which provides a sufficient condition for a positional strategy to be optimal. Consider a ν-valued MPG Γ, for some $\nu \in \mathbf{Q}$, and let $\sigma_0^* \in \texttt{opt}_\Gamma\Sigma_0^M$. Let $\hat{\sigma}_0 \in \Sigma_0^M(\Gamma)$ be any (not necessarily optimal) positional strategy for Player 0 in the MPG Γ. Suppose the following holds:

$$\forall^{v\in V}\pi^*_{G(\Gamma^{w-\nu},\hat{\sigma}_0)}(v) = \pi^*_{G(\Gamma^{w-\nu},\sigma_0^*)}(v).$$

Then, by Proposition 8.2, $\hat{\sigma}_0$ is an optimal positional strategy for Player 0 in the MPG Γ.

We are thus relying on the same *compatibility* relation between Σ_0^M and SEPMs in reweighted EGs which was at the *base* of Theorem 6.4, aiming at extending Theorem 6.4 so to describe the whole $\texttt{opt}_\Gamma\Sigma_0^M$ (and not just the join/top component of it).

Proposition 8.2. *Let the MPG Γ be ν-valued, for some $\nu \in \mathbf{Q}$.*
There is at least one Energy-Lattice of $\texttt{opt}_\Gamma\Sigma_0^M$:

$$\mathcal{X}^*_\Gamma \triangleq \{\pi^*_{G(\Gamma^{w-\nu},\sigma_0)} \mid \sigma_0 \in \texttt{opt}_\Gamma\Sigma_0^M\}.$$

Proof. The only non-trivial point to check being: $\bigsqcup_{f\in\mathcal{X}^*_\Gamma} \Delta_0^M(f,\Gamma^{w-\nu}) \subseteq \texttt{opt}_\Gamma\Sigma_0^M$.

For this, we shall rely on Theorem 6.4. Let $\hat{f} \in \mathcal{X}^*_\Gamma$ and $\hat{\sigma}_0 \in \Delta_0^M(\hat{f},\Gamma^{w-\nu})$ be fixed (arbitrarily). Since $\hat{f} \in \mathcal{X}^*_\Gamma$, then $\hat{f} = \pi^*_{G(\Gamma^{w-\nu},\sigma_0^*)}$ for some $\sigma_0^* \in \texttt{opt}_\Gamma\Sigma_0^M$. Therefore, the following holds:

$$\pi^*_{G(\Gamma^{w-\nu},\hat{\sigma}_0)} = \hat{f} = \pi^*_{G(\Gamma^{w-\nu},\sigma_0^*)}.$$

Clearly, $\hat{\sigma}_0$ is compatible with $\hat{f}$ in the EG $\Gamma^{w-\nu}$, because $\hat{f} = \pi^*_{G(\Gamma^{w-\nu},\hat{\sigma}_0)}$. By Lemma 8.1, since σ_0^* is optimal, then $G(\Gamma^{w-\nu},\sigma_0^*)$ is conservative. Therefore:

$$V_{\hat{f}} = V_{\pi^*_{G(\Gamma^{w-\nu},\sigma_0^*)}} = V.$$

Notice, $\hat{\sigma}_0$ satisfies exactly the hypotheses required by Theorem 6.4. Therefore, $\hat{\sigma}_0 \in \texttt{opt}_\Gamma\Sigma_0^M$. This proves (*).This also shows $\texttt{opt}_\Gamma\Sigma_0^M = \bigsqcup_{f\in\mathcal{X}^*_\Gamma} \Delta_0^M(f,\Gamma^{w-\nu})$, and concludes the proof. □

Proposition 8.3. *Let the MPG Γ be ν-valued, for some $\nu \in \mathbf{Q}$. Let $\mathcal{X}^*_{\Gamma 1}$ and $\mathcal{X}^*_{\Gamma 2}$ be two Energy-Lattices for $\texttt{opt}_\Gamma\Sigma_0^M$. Then, $\mathcal{X}^*_{\Gamma 1} = \mathcal{X}^*_{\Gamma 2}$.*

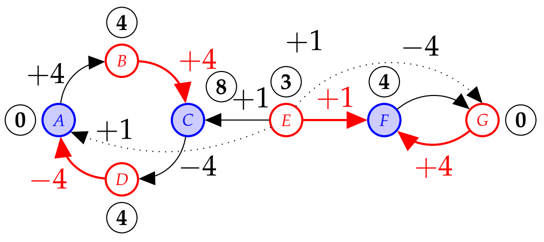
Proof. By symmetry, it is sufficient to prove that $\mathcal{X}^*_{\Gamma 1} \subseteq \mathcal{X}^*_{\Gamma 2}$. Let $f_1 \in \mathcal{X}^*_{\Gamma 1}$ be fixed (arbitrarily). Then, $f_1 = \pi^*_{G(\Gamma^{w-\nu},\hat{\sigma}_0)}$ for some $\hat{\sigma}_0 \in \texttt{opt}_\Gamma\Sigma_0^M$. Since $\hat{\sigma}_0 \in \texttt{opt}_\Gamma\Sigma_0^M$ and since $\mathcal{X}^*_{\Gamma 2}$ is an Energy-Lattices, there exists $f_2 \in \mathcal{X}^*_{\Gamma 2}$ s.t. $\hat{\sigma}_0 \in \Delta_0^M(f_2,\Gamma^{w-\nu})$, which implies $\pi^*_{G(\Gamma^{w-\nu},\hat{\sigma}_0)} = f_2$. Thus, $f_1 = \pi^*_{G(\Gamma^{w-\nu},\hat{\sigma}_0)} = f_2$. This implies $f_1 \in \mathcal{X}^*_{\Gamma 2}$. □

The next theorem summarizes the main point of this section.

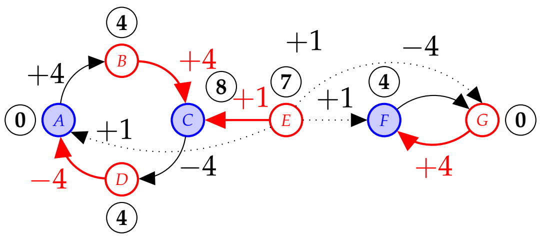
Theorem 8.2. *Let Γ be a ν-valued MPG, for some $\nu \in \mathbf{Q}$. Then, $\mathcal{X}^*_\Gamma \triangleq \{\pi^*_{G(\Gamma^{w-\nu},\sigma_0)} \mid \sigma_0 \in \texttt{opt}_\Gamma\Sigma_0^M\}$ is the unique Energy-Lattice of $\texttt{opt}_\Gamma\Sigma_0^M$.*

Proof. By Proposition 8.2 and Proposition 8.3. □

Example 8.1. *Consider the MPG Γ_{ex}, as defined in Fig. 8.2. Then, $\mathcal{X}^*_{\Gamma_{ex}} = \{f^*, f_1, f_2\}$, where f^* is the least-SEPM of the reweighted EG Γ^{w+1}_{ex}, and where the following holds: $f_1(A) = f_2(A) = f^*(A) = 0$; $f_1(B) = f_2(B) = f^*(B) = 4$; $f_1(C) = f_2(C) = f^*(C) = 8$; $f_1(D) = f_2(D) = f^*(D) = 4$; $f_1(F) = f_2(F) = f^*(F) = 4$; $f_1(G) = f_2(G) = f^*(G) = 0$; finally, $f^*(E) = 0$, $f_1(E) = 3$, $f_2(E) = 7$. An illustration of f_1 is offered in Fig. 8.4a (energy-levels are depicted in circled boldface). whereas f_2 is* *depicted in Fig. 8.4b. Notice that $f^*(v) \leq f_1(v) \leq f_2(v)$ for every $v \in V$, as shown in Fig. 8.4.*



(a) The extremal-SEPM f_1 of Γ^{w+1}_{ex}



(b) The extremal-SEPM f_2 of Γ^{w+1}_{ex}.

Definition 8.3. *Each element $f \in \mathcal{X}^*_\Gamma$ is called* extremal-*SEPM.*

The next lemma is the converse of Lemma 8.1.

Lemma 8.2. *Let the MPG Γ be ν-valued, for some $\nu \in \mathbf{Q}$. Consider any $\sigma_0 \in \Sigma^M_0(\Gamma)$, and assume that $G(\Gamma^{w-\nu}, \sigma_0)$ is conservative. Then, $\sigma_0 \in \texttt{opt}_\Gamma \Sigma^M_0$.*

Proof. Let $C = (v_1, \ldots, v_\ell v_1)$ any cycle in $G(\Gamma, \sigma_0)$. Then, the following holds (if $v_{\ell+1} = v_1$): $\frac{w(C)}{\ell} = \frac{1}{\ell}\sum_{i=1}^{\ell} w(v_i, v_{i+1}) = \nu + \frac{1}{\ell}\sum_{i=1}^{\ell}\Big(w(v_i, v_{i+1}) - \nu\Big) \geq \nu$, where $\frac{1}{\ell}\sum_{i=1}^{\ell}\big(w(v_i, v_{i+1}) - \nu\big) \geq 0$ holds because $G(\Gamma^{w-\nu}, \sigma_0)$ is conservative. By Lemma 6.1, since $w(C)/\ell \geq \nu$ for every cycle C in $G^\Gamma_{\sigma_0}$, then $\sigma_0 \in \texttt{opt}_\Gamma \Sigma^M_0$. □

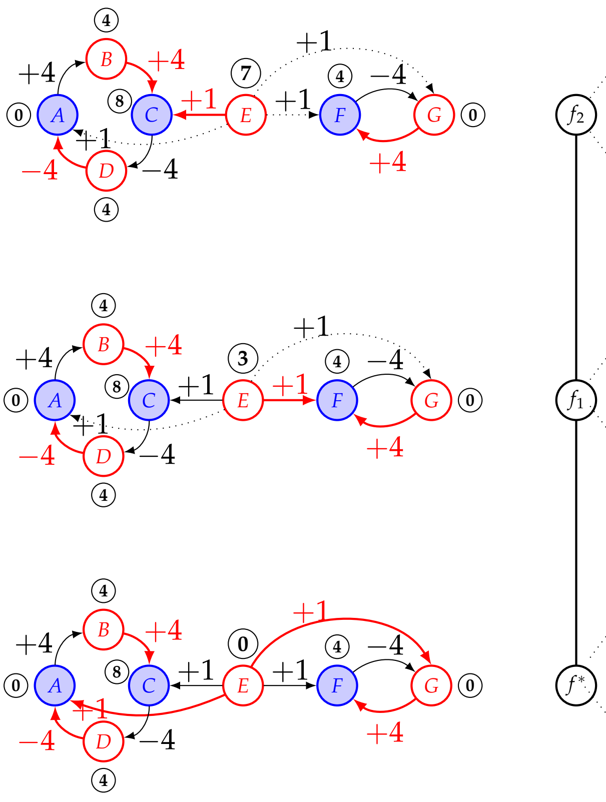
The following proposition asserts some properties of the extremal-SEPMs.

Proposition 8.4. *Let the MPG Γ be ν-valued, for some $\nu \in \mathbf{Q}$. Let $\mathcal{X}^*_\Gamma$ be the Energy-Lattice of $\texttt{opt}_\Gamma \Sigma^M_0$. Moreover, let $f : V \to \mathcal{C}_\Gamma$ be a SEPM for the reweighted EG $\Gamma^{w-\nu}$. Then, the following three properties are equivalent:*

1. *$f \in \mathcal{X}^*_\Gamma$;*
2. *There exists $\sigma_0 \in \texttt{opt}_\Gamma \Sigma^M_0$ s.t. $\pi^*_{G(\Gamma^{w-\nu}, \sigma_0)}(v) = f(v)$ for every $v \in V$.*
3. *$V_f = \mathcal{W}_0(\Gamma^{w-\nu}) = V$ and $\Delta^M_0(f, \Gamma^{w-\nu}) \neq \emptyset$;*

Proof of (1 $\iff$ 2). Indeed, $\mathcal{X}^*_\Gamma = \{\pi^*_{G(\Gamma^{w-\nu}, \sigma_0)} \mid \sigma_0 \in \texttt{opt}_\Gamma \Sigma^M_0\}$. □

Proof of (1 $\Rightarrow$ 3). Assume $f \in \mathcal{X}^*_\Gamma$. Since (1 $\iff$ 2), there exist $\sigma_0 \in \texttt{opt}_\Gamma \Sigma^M_0$ s.t. $\pi^*_{G(\Gamma^{w-\nu}, \sigma_0)} = f$. Thus, $\sigma_0 \in \Delta^M_0(f, \Gamma^{w-\nu})$, so that $\Delta^M_0(f, \Gamma^{w-\nu}) \neq \emptyset$. We claim $V_f = \mathcal{W}_0(\Gamma^{w-\nu}) = V$. Since $\forall (v \in V)\, \texttt{val}^\Gamma(v) = \nu$, then $\mathcal{W}_0(\Gamma^{w-\nu}) = V$ by Lemma 6.2. Next, $G(\Gamma^{w-\nu}, \sigma_0)$ is conservative by Lemma 8.1. Since $G(\Gamma^{w-\nu}, \sigma_0)$ is conservative and $f = \pi^*_{G(\Gamma^{w-\nu}, \sigma_0)}$, then $V_f = V$. Therefore, $V_f = \mathcal{W}_0(\Gamma^{w-\nu}) = V$. □



$\Delta_0^M(f_1,\Gamma^{w-\nu}) = \{\sigma_0^{(4)}\}$
$\sigma_0^{(4)}(B) = C$
$\sigma_0^{(4)}(D) = A$
$\sigma_0^{(4)}(E) = C$
$\sigma_0^{(4)}(G) = F$

$\Delta_0^M(f_1,\Gamma^{w-\nu}) = \{\sigma_0^{(3)}\}$
$\sigma_0^{(3)}(B) = C$
$\sigma_0^{(3)}(D) = A$
$\sigma_0^{(3)}(E) = F$
$\sigma_0^{(3)}(G) = F$

$\Delta_0^M(f^*,\Gamma^{w-\nu}) = \{\sigma_0^{(1)},\sigma_0^{(2)}\}$	
$\sigma_0^{(1)}(B) = C$	$\sigma_0^{(2)}(B) = C$
$\sigma_0^{(1)}(D) = A$	$\sigma_0^{(2)}(D) = A$
$\sigma_0^{(1)}(E) = A$	$\sigma_0^{(2)}(E) = G$
$\sigma_0^{(1)}(G) = F$	$\sigma_0^{(2)}(G) = F$

Figure 8.4: The decomposition of $\texttt{opt}_\Gamma\Sigma_0^M$ (right), for the MPG Γ_{ex}, which corresponds to the Energy-Lattice $\mathcal{X}^*_{\Gamma_{\text{ex}}} = \{f^*, f_1, f_2\}$ (center) (as in Example 8.1). Here, $f^* \leq f_1 \leq f_2$. This brings a lattice $\mathcal{D}^*_{\Gamma_{\text{ex}}}$ of 3 basic subgames of Γ_{ex} (left).

Proof of (1 $\Leftarrow$ 3). Since $\Delta_0^M(f,\Gamma^{w-\nu}) \neq \emptyset$, pick some $\sigma_0 \in \Delta_0^M(f,\Gamma^{w-\nu})$; so, $f = \pi^*_{G(\Gamma^{w-\nu},\sigma_0)}$. Since $V_f = V$ and $f = \pi^*_{G(\Gamma^{w-\nu},\sigma_0)}$, then $G(\Gamma^{w-\nu},\sigma_0)$ is conservative. Since $G(\Gamma^{w-\nu},\sigma_0)$ is conservative, then $\sigma_0 \in \texttt{opt}_\Gamma\Sigma_0^M$ by Lemma 8.2. Since $f = \pi^*_{G^*}$ and $\sigma_0 \in \texttt{opt}_\Gamma\Sigma_0^M$, then $f \in \mathcal{X}^*_\Gamma$ because 2 $\Rightarrow$ 1. □

8.3 A Recursive Enumeration of $\mathcal{X}^*_\Gamma$ and $\texttt{opt}_\Gamma\Sigma_0^M$

An enumeration algorithm for a set S provides an exhaustive listing of all the elements of S (without repetitions). As mentioned in Section 8.2, by Theorem 8.1, no loss of generality occurs if we assume Γ to be ν-valued for some $\nu \in \mathbf{Q}$. One run of the algorithm given in [37] allows one to partition an MPG Γ, into several domains Γ_i each one being ν_i-valued for $\nu_i \in S_\Gamma$; in $O(|V|^2|E|W)$ time and linear space. Still, by Proposition 8.1, Theorem 6.4 is not sufficient for enumerating the whole $\texttt{opt}_\Gamma\Sigma_0^M$; it is enough only for $\Delta_0^M(f^*_\nu,\Gamma^{w-\nu})$ where f^*_ν is the least-SEPM of $\Gamma^{w-\nu}$, which is just the join/top component of $\texttt{opt}_\Gamma\Sigma_0^M$. However, thanks to Theorem 8.2, we now have a refined description of $\texttt{opt}_\Gamma\Sigma_0^M$

in terms $\mathcal{X}^*_\Gamma$.

We offer a recursive enumeration of all the extremal-SEPMs, i.e., $\mathcal{X}^*_\Gamma$, and for computing the corresponding partitioning of $\texttt{opt}_\Gamma(\Sigma^M_0)$. In order to avoid duplicate elements in the enumeration, the algorithm needs to store a lattice $\mathcal{B}^*_\Gamma$ of subgames of Γ, which is related to $\mathcal{X}^*_\Gamma$. We assume to have a data-structure T_Γ supporting the following operations, given a subarena Γ' of Γ: $\texttt{insert}(\Gamma', T_\Gamma)$ stores Γ' into T_Γ; $\texttt{contains}(\Gamma', T_\Gamma)$ returns `T` if and only if Γ' is in T_Γ, and `F` otherwise. A simple implementation of T_Γ goes by indexing $N^{\text{out}}_{\Gamma'}(v)$ for each $v \in V$ (e.g., with a trie data-structure). This runs in $O(|E|\log|V|)$ time, consuming $O(|E|)$ space per stored item. Similarly, one can index SEPMs in $O(|V|\log(|V|W))$ time and $O(|V|)$ space per stored item. The listing procedure is named `enum()`, it takes a ν-valued MPG Γ as input and goes as follows.

1. Compute the least-SEPM f^* of Γ, and `print` Γ to output. Theorem 6.4 can be employed at this stage for enumerating $\Delta^M_0(f^*, \Gamma^{w-\nu})$: indeed, these are all and only those positional strategies lying in the *Cartesian* product of all arcs $(u,v) \in E$ *compatible* with f^* in $\Gamma^{w-\nu}$ (because f^* is the least-SEPM of Γ).

2. Let `St` $\leftarrow \emptyset$ be an empty stack of vertices.

3. For each $\hat{u} \in V_0$, do the following:

    - Compute $E_{\hat{u}} \leftarrow \{(\hat{u},v) \in E \mid f^*(\hat{u}) \prec f^*(v) \ominus (w(\hat{u},v) - \nu)\}$;
    - If $E_{\hat{u}} \neq \emptyset$, then:
        - Let $E' \leftarrow E_{\hat{u}} \cup \{(u,v) \in E \mid u \neq \hat{u}\}$ and $\Gamma' \leftarrow (V, E', w, \langle V_0, V_1\rangle)$.
        - If $\texttt{contains}(\Gamma', T_\Gamma) = \texttt{F}$, do the following:
            * Compute the least-SEPM f'^* of $\Gamma'^{w-\nu}$;
            * If $V_{f'^*} = V$:
              – Push $\hat{u}$ on top of `St` and $\texttt{insert}(\Gamma', T_\Gamma)$.
              – If $\texttt{contains}(f'^*, T_\Gamma) = \texttt{F}$, then $\texttt{insert}(f'^*, T_\Gamma)$ and `print` f'^*.

4. While `St` $\neq \emptyset$:

    - `pop` $\hat{u}$ from `St`; Let $E_{\hat{u}} \leftarrow \{(\hat{u},v) \in E \mid f^*(\hat{u}) \prec f^*(v) \ominus (w(\hat{u},v) - \nu)\}$, and $E' \leftarrow E_{\hat{u}} \cup \{(u,v) \in E \mid u \neq \hat{u}\}$, and $\Gamma' \leftarrow (V, E', w, \langle V_0, V_1\rangle)$;
    - Make a recursive call to `enum()` on input Γ'.

Down the recursion tree, when computing least-SEPMs, the children Value-Iterations can amortize by starting from the energy-levels of the parent. The lattice of subgames $\mathcal{B}^*_\Gamma$ comprises all and only those subgames $\Gamma' \subseteq \Gamma$ that are eventually inserted into T_Γ at Step (3) of enum(); these are called the *basic subgames* of Γ. The correctness of `enum()` follows by Theorem 8.2 and Theorem 6.4. In summary, we obtain the following result.

Theorem 8.3. *There exists a recursive algorithm for enumerating (w/o repetitions) all elements of $\mathcal{B}^*_\Gamma$ with time-delay*[1] *$O(|V|^3|E|W)$, on any input MPG Γ; moreover, the algorithm works with $O(|V||E|)+\Theta(|E||\mathcal{B}^*_\Gamma|)$ space. So, it enumerates $\mathcal{X}^*_\Gamma$ (w/o repetitions) in $O(|V|^3|E|W|\mathcal{B}^*_\Gamma|)$ total time, and $O(|V||E|)+\Theta(|E||\mathcal{B}^*_\Gamma|)$ space.*

To conclude we observe that $\mathcal{B}^*_\Gamma$ and $\mathcal{X}^*_\Gamma$ are not isomorphic as lattices, not even as sets (the cardinality of $\mathcal{B}^*_\Gamma$ can be greater that that of $\mathcal{X}^*_\Gamma$). Indeed, there is a surjective antitone mapping φ_Γ from $\mathcal{B}^*_\Gamma$ onto $\mathcal{X}^*_\Gamma$, (i.e., φ_Γ sends $\Gamma' \in \mathcal{B}^*_\Gamma$ to its least-SEPM $f^*_{\Gamma'} \in \mathcal{X}^*_\Gamma$); still, we can construct instances of MPGs such that $|\mathcal{B}^*_\Gamma| > |\mathcal{X}^*_\Gamma|$, i.e., φ_Γ is not into and $\mathcal{B}^*_\Gamma$, $\mathcal{X}^*_\Gamma$ are not isomorphic. That would be a case of *degeneracy*, and an example MPG Γ_{d} is given in Fig. 8.5.

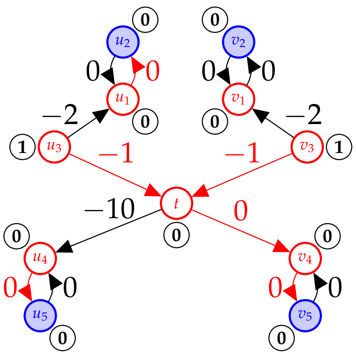


Figure 8.5: An MPG Γ_{d} for which $|\mathcal{B}^*_\Gamma| > |\mathcal{X}^*_\Gamma|$.

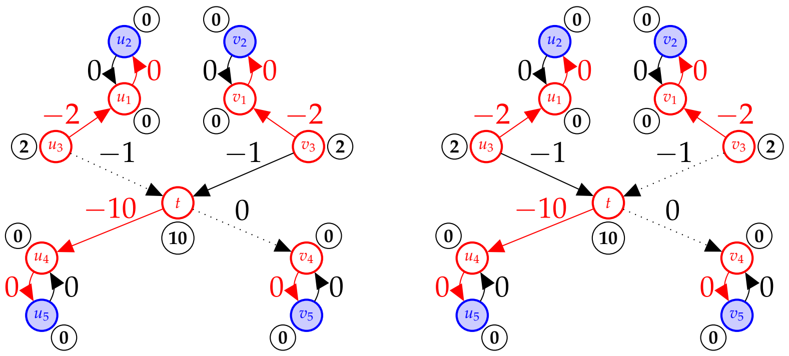


Figure 8.6: Two basic subgames $\Gamma^1_d \neq \Gamma^2_d$ of Γ_{d}, having the same least-SEPM $f^*_1 = f^*_2$.

In the MPG Γ_{d}, Player 0 has to decide how to move only at u_3,v_3 and t; the remaining moves are forced. The least-SEPM f^* of Γ_{d} is: $f^*(u_3)=1$, $f^*(v_3)=1$, $f^*(t)=0$, and $\forall_{x\in V_{\Gamma_{\mathrm{d}}}\setminus\{u_3,v_3,t\}} f^*(x)=0$; leading to the following memoryless strategy: $\sigma^*_0(u_3)=t$, $\sigma^*_0(v_3)=t$, $\sigma^*_0(t)=v_4$. Then, consider the lattice of

[1]A listing algorithm has $O(f(n))$ *time-delay* when the time spent between any two consecutives is $O(f(n))$.

subgames $\mathcal{B}^*_{\Gamma_{\mathrm{d}}}$; particularly, consider the following two basic subgames Γ^1_d, Γ^2_d: let Γ'_d be the arena obtained by removing the arc (t, v_4) from Γ_d; let Γ^1_d be the arena obtained by removing the arc (u_3, t) from Γ'_d; let Γ^2_d be the arena obtained by removing the arc (v_3, t) from Γ'_d. See Fig. 8.6 for an illustration. Next, let f^*_1, f^*_2 be the least-SEPMs of Γ^1_d and Γ^2_d, respectively; then, $f^*_1(u_3) = f^*_2(u_3) = 2$, $f^*_1(v_3) = f^*_2(v_3) = 2$, $f^*_1(t) = f^*_2(t) = 10$, and $\forall_{x \in V_{\Gamma_{\mathrm{d}}} \setminus \{u_3, v_3, t\}} f^*_1(x) = f^*_2(x) = 0$. Thus, $\Gamma^1_d \neq \Gamma^2_d$, but $f^*_1 = f^*_2$; this proves that Γ_{d} is *degenerate* and that $\mathcal{B}^*_\Gamma$, $\mathcal{X}^*_\Gamma$ are not isomorphic.

Bibliography

[1] Daniel Andersson and Sergei Vorobyov. Fast algorithms for monotonic discounted linear programs with two variables per inequality. Technical report, Preprint NI06019-LAA, Isaac Netwon Institute for Mathematical Sciences, Cambridge, UK, 2006.

[2] Roman Barták and Ondřej Čepek. Temporal networks with alternatives: Complexity and model. In David Wilson and Geoff Sutcliffe, editors, *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference, May 7-9, 2007, Key West, Florida, USA.*, pages 641–646. AAAI Press, 2007.

[3] Jordan Bell and Brett Stevens. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309(1):1 – 31, 2009.

[4] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[5] Claude Berge. Topological games with perfect information, in: Contributions to the theory of games. *Annals of Math. Studies, Princeton University Press*, 39:165–178, 1957.

[6] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*, volume 1, 2. Academic Press, 1982.

[7] Claudio Bettini, Xiaoyang Sean Wang, and Sushil Jajodia. Temporal reasoning in workflow systems. *Dist. & Paral. Data.*, 11(3):269–306, 2002.

[8] Henrik Björklund, Sven Sandberg, and Sergei Vorobyov. Memoryless determinacy of parity and mean payoff games: a simple proof. *Theoretical Computer Science*, 310(1–3):365 – 378, 2004.

[9] Henrik Björklund and Sergei Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210 – 229, 2007.

[10] Hans L. Bodlaender, Michael J. Dinneen, and Bakhadyr Khoussainov. *Algorithms and Computation: 12th International Symposium, ISAAC 2001 Christchurch, New Zealand, December 19–21, 2001 Proceedings*, chapter On

Game-Theoretic Models of Networks, pages 550–561. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[11] Charles L. Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3(1/4):35–39, 1901.

[12] P. Bouyer, U. Fahrenberg, K. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer Berlin Heidelberg, 2008.

[13] Julian Bradfield and Igor Walukiewicz. The mu-calculus and model-checking.

[14] L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J. F. Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011.

[15] Lubos Brim and Jakub Chaloupka. Using strategy improvement to stay alive. *Int. J. Found. Comput. Sci.*, 23(3):585–608, 2012.

[16] Richard J. Büchi. Using determinancy of games to eliminate quantifiers. In *FCT*, pages 367–378, 1977.

[17] Massimo Cairo, Carlo Comin, and Romeo Rizzi. Instantaneous reaction-time in dynamic-consistency checking of conditional simple temporal networks. In *23rd International Symposium on Temporal Representation and Reasoning, TIME 2016, Kongens Lyngby, Denmark, October 17-19, 2016*, pages 80–89, 2016.

[18] A. Chakrabarti, L. de Alfaro, T. Henzinger, and M. Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *Embedded Software*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer Berlin Heidelberg, 2003.

[19] K. Chatterjee, M. Henzinger, S. Krinninger, and D. Nanongkai. Polynomial-time algorithms for energy games with special weight structures. *Algorithmica*, 70(3):457–492, 2014.

[20] S J Chinn and G R Madey. Temporal representation and reasoning for workflow in engineering design change review. *IEEE Transactions on Engineering Management*, 47(4):485–492, 2000.

[21] A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, and M. Roveri. Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In *21st Intern. Symp. on Temp. Repres. and Reasoning, TIME 2014, Verona, Italy*, pages 27–36, 2014.

[22] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[23] C. Combi, L. Hunsberger, and R. Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. In *ICAART 2013 - Proc. of the 5th Intern. Conf. on Agents and Artif. Intell., Vol. 2, Spain, 2013*, pages 144–156, 2013.

[24] Carlo Combi, Mauro Gambini, Sara Migliorini, and Roberto Posenato. Modelling temporal, data-centric medical processes. In *Proc. of the 2nd ACM SIGHIT Int. Health Informatics Symp.*, IHI '12, pages 141–150, New York, NY, USA, 2012. ACM.

[25] Carlo Combi, Mauro Gambini, Sara Migliorini, and Roberto Posenato. Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, 44(9):1182–1203, Sept. 2014.

[26] Carlo Combi, Matteo Gozzi, Roberto Posenato, and Giuseppe Pozzi. Conceptual modeling of flexible temporal workflows. *ACM Trans. Auton. Adapt. Syst.*, 7(2):19:1–19:29, July 2012.

[27] Carlo Combi and Roberto Posenato. Controllability in temporal conceptual workflow schemata. In *BPM 2009 - Proc. of the 7th Business Process Management Conference*, pages 64–79, 2009.

[28] Carlo Combi and Giuseppe Pozzi. Architectures for a temporal workflow management system. In *Proc. of the 2004 ACM Symp. on Applied Computing*, SAC '04, pages 659–666, New York, NY, USA, 2004. ACM.

[29] Carlo Comin. Algebraic characterization of the class of languages recognized by measure only quantum automata. *Fundamenta Informaticae*, 134(3-4):335–353, 2014.

[30] Carlo Comin. A HyTN Consistency Check Algorithm Implementation in C++. `http://profs.scienze.univr.it/~posenato/software/hytn/2015_v1_Code.tgz`, January 2015.

[31] Carlo Comin, Anthony Labarre, Romeo Rizzi, and Stéphane Vialette. *Sorting with Forbidden Intermediates. Algorithms for Computational Biology: Third International Conference, AlCoB 2016, Trujillo, Spain, June 21-22, 2016, Proceedings*, pages 133–144. Springer International Publishing, Cham, 2016.

[32] Carlo Comin, Roberto Posenato, and Romeo Rizzi. A tractable generalization of simple temporal networks and its relation to mean payoff games. In *21st International Symposium on Temporal Representation and Reasoning (TIME 2014)*, pages 7–16. IEEE CPS, sep 2014.

[33] Carlo Comin, Roberto Posenato, and Romeo Rizzi. Hyper temporal networks. *Constraints*, pages 1–39, March, 2016.

[34] Carlo Comin and Romeo Rizzi. Dynamic consistency of conditional simple temporal networks via mean payoff games: a singly-exponential time DC-Checking. In *22nd International Symposium on Temporal Representation and Reasoning (TIME 2015)*, pages 19–28. IEEE CPS, sep 2015.

[35] Carlo Comin and Romeo Rizzi. Energy structure and improved complexity upper bound for optimal positional strategies in mean payoff games. In *3rd International Workshop on Strategic Reasoning*, volume 20, September 2015.

[36] Carlo Comin and Romeo Rizzi. An improved upper bound on maximal clique listing via rectangular fast matrix multiplication. *CoRR*, abs/1506.01082, 2015.

[37] Carlo Comin and Romeo Rizzi. Faster $O(|V|^2|E|W)$-time energy algorithms for optimal strategy synthesis in mean payoff games. *CoRR*, abs/1609.01517, 2016.

[38] Carlo Comin and Romeo Rizzi. Improved pseudo-polynomial bound for the value problem and optimal strategy synthesis in mean payoff games. *Algorithmica*, pages 1–27, 2016.

[39] Carlo Comin and Romeo Rizzi. Linear time algorithm for update games via strongly-trap-connected components. *CoRR*, abs/1610.09679, 2016.

[40] Carlo Comin and Romeo Rizzi. Checking dynamic consistency of conditional hyper temporal networks via mean payoff games (hardness and (pseudo) singly-exponential time algorithm). *Information and Computation*, to appear, 2017.

[41] John H. Conway. *On Numbers and Games*. Academic Press, 1976.

[42] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[43] Jaco de Bakker. *Mathematical Theory of Program Correctness*. Prentice Hall, 1980.

[44] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1–3):61–95, 1991.

[45] M. J. Dinneen and B. Khoussainov. Update games and update networks. In *Proceedings of the 10th Australasian Workshop on Combinatorial Algorithms*, AWOCA '99, pages 7–18, 1999.

[46] Michael J. Dinneen and Bakhadyr Khoussainov. *Graph-Theoretic Concepts in Computer Science: 26th International Workshop, WG 2000 Konstanz, Germany, June 15–17, 2000 Proceedings*, chapter Update Networks and Their Routing Strategies, pages 127–136. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[47] Johann Eder, Wolfgang Gruber, and Euthimios Panagos. Temporal modeling of workflows with conditional execution paths. In M Ibrahim, J Küng, and N Revell, editors, *Database and Expert Systems Applications (DEXA 2000)*, volume 1873 of *LNCS*, pages 243–253. Springer Berlin Heidelberg, 2000.

[48] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In Matthias Jarke and Andreas Oberweis, editors, *Advanced Information Systems Engineering*, volume 1626 of *LNCS*, pages 286–300. Springer Berlin Heidelberg, 1999.

[49] Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979.

[50] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377, 1991.

[51] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM J. Comput.*, 29(1):132–158, 1999.

[52] E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, pages 385–396, 1993.

[53] E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model checking for the μ-calculus and its fragments. *Theor. Comput. Sci.*, 258(1-2):491–522, 2001.

[54] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 267–278, 1986.

[55] L. R. Ford and D. R. Fulkerson. *Flows in networks*. Princeton University Press Princeton, N.J, 1962.

[56] Lance Fortnow and Virginia Gold. Three papers cited for laying foundation of growth in algorithmic game theory. Gödel Prize 2012. Association for Computing Machinery (ACM SIGACT).

[57] David Gale and Frank Stewart. Infinite games with perfect information. *Contributions to the theory of games, vol. 2. Annals of Mathematical Studies*, 28:245–266, 1953.

[58] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[59] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[60] Pedro M. Gonzalez del Foyo and José Reinaldo Silva. Using time Petri Nets for modeling and verification of timed constrained workflow systems. In *ABCM Symposium Series in Mechatronics*, pages 471–478. Dept. Of Mechatronics, University of São Paulo, Brazil, 2008.

[61] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[62] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

[63] Patrick M. Grundy. Mathematics and games. *Eureka*, 2:6–8, 1939.

[64] V.A. Gurvich, A.V. Karzanov, and L.G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28(5):85 – 91, 1988.

[65] David Hollingsworth. The workflow reference model. http://www.wfmc.org/standards/model.htm, 1995.

[66] Florian Horn. Explicit muller games are PTIME. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008, December 9-11, 2008, Bangalore, India*, pages 235–243, 2008.

[67] L. Hunsberger, R. Posenato, and C. Combi. The dynamic controllability of conditional stns with uncertainty. In *Proc. of the Plan. and Plan Exec. for Real-World Syst.: Princip. and Pract. (PlanEx)*, ICAPS-2012, page 121–128, Atibaia, Sao Paulo, Brazil, 2012.

[68] Luke Hunsberger and Roberto Posenato. Checking the dynamic consistency of conditional temporal networks with bounded reaction times. In *Proc. of the Twenty-Sixth Internat. Conf. on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, pages 175–183, 2016.

[69] Luke Hunsberger, Roberto Posenato, and Carlo Combi. A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In *22nd International*

Symposium on Temporal Representation and Reasoning, TIME 2015, Kassel, Germany, September 23-25, 2015, pages 4–18, 2015.

[70] Marcin Jurdziński. Deciding the winner in parity games is in UP $\cap$ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[71] Lina Khatib, Paoul Morris, Robert Morris, Francesca Rossi, Alessandro Sperduti, and Kristen Brent Venable. Solving and learning a tractable class of soft temporal constraints: Theoretical and experimental results. *AI Communications*, 20(3):181–209, August 2007.

[72] Lina Khatib, Paul Morris, Robert Morris, and Francesca Rossi. Temporal constraint reasoning with preferences. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, pages 322–327, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[73] Manolis Koubarakis. From local to global consistency in temporal constraint networks. *Theoretical Computer Science*, 173(1):89 – 112, 1997.

[74] Elias Koutsoupias and Christos H. Papadimitriou. Worst-case equilibria. In *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, pages 404–413, 1999.

[75] Dexter Kozen. Results on the Propositional μ-Calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[76] Andreas Lanz and Manfred Reichert. Enabling time-aware process support with the atapis toolset. In Lior Limonad and Barbara Weber, editors, *Proceedings of the BPM Demo Sessions 2014*, volume 1295 of *CEUR Workshop Proceedings*, pages 41–45. CEUR, 2014.

[77] Andreas Lanz, B Weber, and Manfred Reichert. Time patterns for process-aware information systems. *Requirements Engineering*, 19(2):113–141, September 2012.

[78] Christophe Lecoutre. *Constraint Networks: Targeting Simplicity for Techniques and Algorithms*. Wiley-ISTE, 2009.

[79] Y.M. Lifshits and D.S. Pavlov. Potential theory for mean payoff games. *Journal of Mathematical Sciences*, 145(3):4967–4974, 2007.

[80] Martin Loebl and Jaroslav Nešetřil. Linearity and unprovability of set union problem strategies. *Journal of Algorithms*, 23(2):207 – 220, 1997.

[81] Donald A. Martin. Borel determinacy. *Annals of Mathematics. Second Series.*, 102 (2):363–371, 1975.

[82] Donald A. Martin. A purely inductive proof of borel determinacy. *Recursion theory. Proc. Sympos. Pure Math.*, 1:303–308, 1982.

[83] Donald A. Martin and John R. Steel. A proof of projective determinacy. *Journal of the American Mathematical Society*, 2:71–125, 1989.

[84] Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966.

[85] Jan Mendling, Hajo A. Reijers, and Wil M. P. van der Aalst. Seven process modeling guidelines (7PMG). *Information and Software Technology*, 52(2):127–136, 2010.

[86] P.M. Merlin and David J. Farber. Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036–1043, Sep 1976.

[87] M. D. Moffitt and M. E. Pollack. Applying local search to disjunctive temporal problems. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK*, pages 242–247, 2005.

[88] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[89] Paul Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, pages 494–499, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[90] Jan Mycielski and Hugo Steinhaus. A mathematical axiom contradicting the axiom of choice. *Bulletin de l'Académie Polonaise des Sciences*, 3:1–3, 1962.

[91] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1-2):166–196, 2001.

[92] Damian Niwinski. On Fixed-Point Clones (Extended Abstract). In *Automata, Languages and Programming, 13th International Colloquium, ICALP86, Rennes, France, July 15-19, 1986, Proceedings*, pages 464–473, 1986.

[93] A. Oddi. Constraint-based strategies for the disjunctive temporal problem: Some new results. In *Proceedings of the Sixth European Conference on Planning*, 2014.

[94] John C. Oxtoby. The Banach-Mazur game and Banach Category Theorem. *Annals of Math. Studies, Princeton University Press*, III:159–163, 1957.

[95] A.K. Pani and G.P. Bhattacharjee. Temporal representation and reasoning in artificial intelligence: A review. *Mathematical and Computer Modelling*, 34(1–2):55–80, 2001.

[96] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[97] Jakub Pawlewicz and Mihai Pătraşcu. Order statistics in the farey sequences in sublinear time and counting primitive lattice points in polygons. *Algorithmica*, 55(2):271–282, 2009.

[98] pgsolver. The pgsolver collection of parity game solvers. https://github.com/tcsprojects/pgsolver, 2013.

[99] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[100] Tim Roughgarden and Éva Tardos. How bad is selfish routing? In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 93–102, 2000.

[101] Stuart Russell and Peter Norvig. *Artificial intelligence : a modern approach*. Prentice Hall, 3 edition, December 2010.

[102] T. K. Satish Kumar. On the tractability of restricted disjunctive temporal problems. In *ICAPS 2005 - Proceedings of the 15th International Conference on Automated Planning and Scheduling*, pages 110–119, June 2005.

[103] Sven Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic*, volume 5213 of *LNCS*, pages 369–384. Springer, 2008.

[104] Sven Schewe, Ashutosh Trivedi, and Thomas Varghese. Symmetric strategy improvement. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 388–400. Springer, 2015.

[105] Dana Scott and Jaco de Bakker. A theory of programs. *IBM Vienna*, 1979.

[106] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. John Wiley & Sons, 2008.

[107] D.E. Smith, J. Frank, and A.K. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.

[108] Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.

[109] Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Inf. Comput.*, 81(3):249–264, 1989.

[110] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[111] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[112] I. Tsamardinos and M. E. Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artif. Intell.*, 151(1-2):43–89, 2003.

[113] Ioannis Tsamardinos, Thierry Vidal, and Martha E Pollack. Ctp: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003.

[114] Stanisław Ulam. *Adventures of a mathematician*. Charles Scribner and Sons, New York, 1976.

[115] Stanisław Ulam. *The Scottish Book*. Los Alamos Scient. Lab. Monograph, 1977.

[116] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[117] Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

[118] Thomas Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Soc. Math. Belg.*, 8(2), 2001.

[119] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.

[120] Michael Wooldridge and Julian Gutierrez, editors. *Moshe Y. Vardi, A Revisionist History of Algorithmic Game Theory (Invited Talk)*, 3rd International Workshop on Strategic Reasoning, SR 2015, Oxford, U.K. 21st-22nd September, 2015.

[121] Stéphane Gaubert Xavier Allamigeon, Pascal Benchimol. The tropical shadow-vertex algorithm solves mean payoff games in polynomial time on average. In *ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 89–100, 2014.

[122] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

[123] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.