University of Trento

University of Verona

Doctoral Thesis

From optimization to listing: theoretical advances in some enumeration problems

Author:
Alice Raffaele

Supervisor:
Prof. Romeo Rizzi

Academic Year 2021/2022

From optimization to listing: theoretical advances in some enumeration problems

by Alice Raffaele

A doctoral thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Joint Doctoral Programme in Mathematics – Cycle XXXIII

University of Trento, Department of Mathematics
University of Verona, Department of Computer Science

Trento, Italy

March 2022

Abstract

The main aim of this thesis is to investigate some problems relevant in enumeration and optimization, for which I present new theoretical results.

First, I focus on a classical enumeration problem in graph theory with several applications, such as network reliability. Given an undirected graph, the objective is to list all its bonds, i.e., its minimal cuts. I provide two new algorithms, the former having the same time complexity as the state of the art by [Tsukiyama et al., 1980], whereas the latter offers an improvement. Indeed, by refining the branching strategy of [Tsukiyama et al., 1980] and relying on some dynamic data structures by [Holm et al., 2001], it is possible to define an $\widetilde{O}(n)$-delay algorithm to output each bond of the graph as a bipartition of the n vertices. Disregarding the polylogarithmic factors hidden in the $\widetilde{O}$ notation, this is the first algorithm to list bonds in a time linear in the number of vertices.

Then, I move to studying two well-known problems in theoretical computer science, that are checking the duality of two monotone Boolean functions, and computing the dual of a monotone Boolean function. Also these are relevant in many fields, such as linear programming. [Fredman and Khachiyan, 1996] developed the first quasi-polynomial time algorithm to solve the decision problem, thus proving that it is not coNP-complete. However, no polynomial-time algorithm has been discovered yet. Here, by focusing on the symmetry of the two input objects and exploiting full covers introduced by [Boros and Makino, 2009], I define an alternative decomposition approach. This offers a strong bound which, however, in the worst case, is still the same as [Fredman and Khachiyan, 1996]. Anyway, I also show how to adapt it to obtain a polynomial-space algorithm to solve the dualization problem.

Finally, as extra content, this thesis contains an appendix about the topic of communicating operations research. By starting from two side projects not related to enumeration, and by comparing some relevant considerations and opinions by researchers and practitioners, I discuss the problem of properly promoting, fostering, and communicating findings in this research area to laypeople.

Declaration of Authorship

I, Alice Raffaele, hereby declare that this thesis, titled *"From optimization to listing: theoretical advances in some enumeration problems"*, was carried out by me for the degree of Doctor of Philosophy in Mathematics under the guidance and supervision of Prof. Romeo Rizzi, Department of Computer Science, University of Verona.

I certify that this thesis has not been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains original material not previously published or written by other people, except where the due reference was clearly pointed out in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Trento, the University of Verona, and, where applicable, any partner institution responsible for the joint-award of this degree. Also, I certify that, where I have consulted the published work of others, this is always clearly attributed. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work. I have acknowledged all main sources of help.

Signed: Alice Raffaele

Date: March 18, 2022

"My vocation is to write, and I have known this for a long time.
I hope I won't be misunderstood; I know nothing about the value of the things
that I am able to write. I know that writing is my vocation."
Natalia Ginzburg, *"My vocation"*, in *"The Little Virtues"*

"When life gives you lemons,
make your best lemonade,
and plant lemon seeds."

Contents

List of Figures

List of Tables

List of Algorithms

To William

Preface

"Until the moment preceding that in which we begin to write, we have the world at our disposal – the one that for each of us constitutes the world, a sum of information, experiences, values – the world given as a whole; [...], and we want to extract a discourse, a story, a feeling from this world: or perhaps more exactly we want to carry out an operation that allows us to place ourselves in this world."

Italo Calvino, *On beginning and ending*, in *Six Memos for the Next Millennium*

Operations research is a branch of applied mathematics. Given its interdisciplinary nature, it can be related to other several disciplines, such as algorithms, graph theory, and combinatorics. Also, it can be exploited to tackle many situations arising from reality, such as logistics, production and facilities planning, marketing and finance, and so on. Thus, the problems studied in operations research are disparate, spanning from more theoretical questions to very practical applications. Usually, what all these problems share is the goal to minimize (or maximize) a given *objective*, by deciding the values of some objects called *variables*, and taking into account some *constraints* to be satisfied. Here we talk about *optimization* problems. Since the years of World War II, operations research has been developing different paradigms and approaches to study such problems, formulate them as models, and solve them. Generally, exact methods are preferred because they allow for obtaining an optimal solution with certainty. However, sometimes these methods can struggle in their computation, e.g., when the size of the input instance is too big or when the problem itself is hard to be solved (in the computational complexity sense). To compute a solution at least "good", one can then rely on other methods, such as approximations algorithms or heuristics. By using an algorithm of the former kind, the solution obtained would also come with a proved bound on the distance from the optimum, whereas the latter would not provide any information on the quality of the solution. Another approach considers to compute a set of feasible solutions, choose the best one found, and maybe starting from this to launch an algorithm of the previous kinds. Sometimes it may also be useful to compute and output all feasible solutions, for instance when the objective function of the problem is not clear or when it is dynamic and subject to changes. In this case, we move from the context of optimization to the one of *listing*, by introducing enumeration problems and algorithms.

Outline of the thesis

The main aim of this thesis is to investigate some relevant problems in enumeration and optimization, for which I present new theoretical results.

In **Chapter 1**, I provide a short introduction on the research area of enumeration, by highlighting some differences with optimization problems and algorithms, in terms of computational complexity and techniques.

The two core chapters of the thesis have a similar structure. First, I briefly introduce the problem studied and the main results obtained. I also summarize the state of the art both in terms of the attempts done to solve the problem and in terms of the applications it is relevant to. Then, I provide basic notions and definitions in order to formally define the problem. Successively, I fully describe and explain my contribution. Finally, I conclude by discussing current and further work, as well as other possible research directions.

Chapter 2 is based on the following manuscript, submitted for publication:

- [Raffaele et al., 2021]. Raffaele, A., Rizzi, R., and Uno, T., *Listing the bonds of a graph in $\widetilde{O}(n)$-delay*. June 2021. *Submitted*.

Here I work on some fundamental concepts of graph theory, that are, connectivity and cuts. Given an undirected and connected graph, a cut is a set of edges that, if removed, would disconnect the graph. A cut can also be represented as a bipartition of the vertices of the graph. Usually, in linear programming, we look for a *minimum* cut, i.e., a cut having the minimum number of edges or the smallest capacity. To compute one, all possible bipartitions of vertices can be examined, by checking the corresponding set of edges connecting the two partitions. However, this approach is not efficient, given the exponential number of bipartitions to examine. Rather than doing this, we apply the famous max flow-min cut theorem by [Ford and Fulkerson, 2015] and we rely on polynomial-time algorithms for the maximum-flow problem. What if we want to obtain, instead, all the cuts? To represent them, we can compute only those that are *minimal* under inclusion. When a cut is minimal, none of its subsets is also a cut, and the corresponding bipartition of vertices leads to two connected induced subgraphs.

The problem of listing all minimal cuts, or *bonds*, of a graph, has been quite studied in enumeration. The state of the art corresponds to the polynomial-delay algorithm by [Tsukiyama et al., 1980], that allows outputting a new bond in a time linear in the number of edges of the given graph. In this part of the thesis, I propose two algorithms, the former having the same time complexity as [Tsukiyama et al., 1980], whereas the latter improving this by relying on dynamic-graph algorithms [Holm et al., 2001]. In this way, it is possible to obtain the first polylogarithmic-delay listing algorithm for bonds, linear in the number of vertices of the given graph.

Chapter 3 is based on the following manuscript, submitted for publication:

- [Raffaele and Rizzi, 2021]. Raffaele, A. and Rizzi, R., *A new decomposition for the Monotone Boolean Duality problem*. July 2021. *Submitted*.

Here I study the *Monotone Boolean Duality* problem and the *Monotone Boolean Dualization* problem. The former is the following well-known decision problem: given two Monotone Boolean functions, one described by a disjunctive normal form and the other by a conjunctive normal form, the goal is to determine whether they are *dual* of each other, i.e., whether they represent the exact same function. The latter is similar but it is instead a listing problem: given a Monotone Boolean function, expressed through a disjunctive normal form, the goal is to compute its dual, expressed through a conjunctive normal form.

What brought me to these problems is a fundamental question in linear programming. A polyhedron can be described either as the intersection of a set of some closed halfspaces or as the Minkowski sum of the convex hull of a finite set of points plus a conical combination of vectors ([Ziegler, 2012]). How can we move from one representation to the other, possibly in an efficient way? This question, known as the *Vertex (Facet) Enumeration* problem, is still open, though some positive and negative results have been obtained in the last twenty years (e.g., [Bussieck and Lübbecke, 1998], [Khachiyan et al., 2006]). There is a connection between the Vertex (Facet) Enumeration problem and another problem denoted by *Joint Generation*, which also can be related to the Monotone Boolean duality problem and to the Monotone Boolean dualization problem (e.g., [Boros et al., 2002]). Thus, I decided to put my focus on these.

Despite the remarkable result of [Fredman and Khachiyan, 1996], who proved that the Monotone Boolean Duality problem is not coNP-complete by providing a quasi-polynomial time algorithm, the exact complexity of the problem has not been discovered yet. Several attempts and approaches have been proposed in the last twenty-five years, the most relevant being [Elbassioni, 2008] and [Boros and Makino, 2009]. By starting from these, first I propose a new decomposition scheme for the Monotone Boolean Duality problem. This approach better highlights the inherent symmetry of the problem and offers a strong bound, even if, in the worst case, it still has the same time complexity as [Fredman and Khachiyan, 1996]. Then, I show how to reduce the Monotone Boolean Dualization problem to the Monotone Boolean Duality problem. Being this a decision problem, there is no need to worry about the memory needed to solve it. Instead, when dealing with dualization, more attention should be paid in trying not to occupy an exponential space. By adapting this approach introducing some changes, I am able to present a polynomial-space algorithm to compute the dual of a given monotone Boolean function.

In **Chapter 4**, I draw some general conclusions on the two core topics discussed, before moving on to some extra contents. Working in operations research can offer several and disparate problems to investigate and explore. Indeed, my doctorate had quite a manifold nature, by including many other projects not limited to enumeration, some more connected to the so-called *third mission* of universities, that is, exploiting knowledge and research for the benefit of the social, cultural, and economic development ([Compagnucci and Spigarelli, 2020]). This is the reason why this thesis contains an appendix which, I hope, will be as attractive as the main contributions.

Appendix A is an extra essay about the communication of operations research to laypeople. It is mainly based on the following manuscript, already published.

- [Raffaele, 2021]. Raffaele, A., *Becoming Visible: Why We Should be Better Communicators Now*. Springer Nature Operations Research Forum 2, 7 (2021). `https://doi.org/10.1007/s43069-020-00051-y`. *Adapted by permission from Copyright Clearance Center.*

In order to introduce the topic and the relevance of communicating operations research, I start by providing an overview on two side projects I worked on during my doctorate. The former side project is an industrial case study, about the analysis of the current implementation of the interlibrary loan service provided by the public company Province of Brescia. In particular, I focus on the evaluation of the routing aspects, which I tackle by using mixed-integer linear programming. This part is based on the following manuscript, still a working paper:

- [Raffaele et al., 2022] Raffaele, A., Gussago, M., Zavatteri, M., Bazzoli, F., and Rizzi, R., *Analysis, evaluation, and improvement of the routing of an interlibrary loan service: the case study of Province of Brescia* (2022). *Working paper*.

The latter side project is instead related to mathematics education and the impact that operations research can have to increase motivation and interest towards mathematics, especially in secondary-school students. This led me to develop, together with three other young researchers and a high-school teacher, an educational initiative based on operations research addressed to Grades 10–12, as described in [Raffaele and Gobbi, 2021], [Colajanni et al., 2022], and [Taranto et al., 2022].

What these two side projects share is the higher purpose to make operations research more known, exploited, and appreciated. They represent just two examples of the various fields, outside the academic environment, where operations research can be promoted and applied by using terminology appropriate to the context. By starting from these, I reflect on the problem of properly communicating research findings to laypeople, as well as the issue of visibility of operations research. First, I collect and compare some relevant considerations and opinions by researchers and practitioners on the topic. Then, I bring some food for thought, in order to refuel the discussion and to encourage researchers to take action, to make operations research more visible.

Finally, in the **References**, I report all scientific papers, books, and other kinds of resources consulted and cited throughout the whole document.

Summary of contributions

In the main body of the thesis:

- I provide two new enumeration algorithms to solve the problem of listing all bonds in an undirected graph. The former has the same time complexity as the state of the art, whereas the latter improves it, by offering the first polylogarithmic-delay algorithm for bonds, linear in the number of vertices.
- I propose another decomposition for the Monotone Boolean Duality problem, where I highlight the symmetric nature of the problem. In the worst case, this has the same complexity as the state of the art.
- I propose a polynomial-space enumeration algorithm to solve the Monotone Boolean Dualization problem.

In the appendix:

- I discuss the problem of promoting and communicating operations research to laypeople in different contexts, from mathematics education to industry.

Chapter 1

Introduction to enumeration

"Whenever humanity seems condemned to heaviness, I think I should fly like Perseus into a different space. I don't mean escaping into dreams or into the irrational. I mean that I have to change my approach, look at the world from a different perspective, with a different logic and with fresh methods of cognition and verification."

Italo Calvino, *Lightness*, in *Six Memos for the Next Millennium*

In this first chapter, I give a brief overview on the research area of enumeration. In particular, I provide some basic definitions and notions, needed in the following chapters of the thesis.

1.1 Enumeration problems and algorithms

In theoretical computer science, a *decision problem* asks to answer the following question: *"Given a problem, does it admit a feasible solution?"*. In operations research, we typically study *optimization problems*, i.e., problems where we would like to minimize (or maximize) an objective function by deciding the values of some variables and, at the same time, satisfying a set of constraints. The question to answer becomes: *"Given a problem, what is an optimal solution, among all the feasible ones?"*. To manage this, first we can use mathematical modelling to formulate the problem according to a certain paradigm (e.g., linear programming). Then, we can exploit existing solvers to get the solution we are interested in. We can rely on exact methods, approximation algorithms, or heuristics. However, there may be several issues making the task more difficult to achieve. For instance, sometimes solvers can struggle in finding optimal solutions when dealing with big instances having a huge number of variables and/or constraints. In this case, heuristics and meta-heuristics are usually preferred to get good sub-optimal solutions, even if they cannot offer a guarantee in terms of distance from the optimum, unlike approximation algorithms.

Other times, the available data describing a problem may not be sufficient to find a proper criterion to identify the characteristics of an optimal solution. Consequently, also the objective function may be unclear to be defined and stated.

A possible way to deal with this is not to limit the search to an optimal solution only, but to list other (or all) possible solutions, thus relying to *enumeration*. This term comes from the latin word "*enumeratio, -tionis*"[1], that indicates the action of listing all objects in a collection, in a specific or random order. This concept has been studied in several disciplines, for instance in philosophy and logic. In mathematics and computer science, an *enumeration* (or *listing*) *problem* consists in listing all elements in a given set, according to some specific conditions. It thus asks the following question: "*Given a problem, what are all its feasible solutions?*". A procedure that solves an enumeration problem is called an *enumeration* (or *listing*) *algorithm*. Consider, for example, the well-known *Travelling Salesman Problem* (TSP). Given a list of cities and the distances between each pair of them, the optimization version of the TSP asks to compute the shortest tour to visit each city exactly once and return to the origin city ([Applegate et al., 2006]). The decision version of the TSP asks only to determine whether a tour of a given length exists. In the listing version of the TSP, we want to return all possible tours satisfying the constraints, no matter their length. A nice way to link optimization and enumeration is listing all *near-optimal* solutions, i.e., all feasible solutions such that their objective function values are within a certain range from the optimal value (see, e.g., [Shmoys, 1995]). In our example about the TSP, we might be interested in computing all tours of a length within $k\%$ of the optimum.

What are the main features we require to an enumeration algorithm? First, we want it to be *correct*, i.e., all its outputs must be feasible solutions of the input instance of the given problem. Then, we ask an enumeration algorithm to output all the solutions of the input instance, i.e., it has to be *complete*. Also, an enumeration algorithm has to *avoid duplication*, i.e., there are no repetitions and each solution of the given input instance is output exactly once. There are several ways to guarantee this. For instance, we could store in memory all solutions already computed

[1] `http://www.perseus.tufts.edu/hopper/text?doc=Perseus:text:1999.04.0059:entry=enumero`

and checking, whenever a new solution is found, whether it has been already output or not, by including a sub-procedure to decide it, or by defining a canonical form of encoding solutions in order to avoid isomorphic solutions ([Marino, 2015]). Finally, we would like to enumerate quickly. Indeed, a desirable attribute of an enumeration algorithm is *efficiency*. We may be able to compute a feasible solution in polynomial time, but we can also consider how to efficiently derive other solutions from that ([Uno, 2016]). When a problem is small and the variables are few, or when the number of solutions can be controlled by some parameters (e.g., solution size), then enumerating all the solutions may not be a huge task. When an enumeration problem admits an exponential number of solutions, one can try to unify similar solutions into others. For instance, some problems allow defining *maximal* (or *minimal*) under-inclusion solutions, that subsume the information contained in others. These may be exponentially fewer, and thus an enumeration algorithm could focus on just outputting these ([Uno, 2016], [Conte and Uno, 2019]).

This chapter is organized as follows. In Section 1.2, I present some applications of enumeration in different disciplines. In Section 1.3, I briefly describe the main complexity classes used to classify enumeration problems. Successively, in Sections 1.4 and 1.5, I summarize the most common techniques used to design and to analyse, respectively, enumeration algorithms.

1.2 Applications

Enumeration algorithms are often used to list objects in *graph theory*, a discipline that studies graphs, i.e., mathematical structures representing pairwise relations between sets of entities. For instance, one can be interested in finding all paths (e.g., [Danielson, 1968], [Eppstein, 1998], and [Birmelé et al., 2012]), cliques (e.g., [Bron and Kerbosch, 1973]), cycles (e.g., [Johnson, 1975] and [Ferreira et al., 2014]), trees (e.g., [Gabow and Myers, 1978], [Shioura et al., 1997], and [Ferreira et al., 2011]), cuts (e.g., [Tsukiyama et al., 1980] and [Abel and Bicker, 1982]), or independent sets (e.g., [Johnson et al., 1988]).

The strong interest in graphs and their related structures also comes from the fact that these are often exploited in practice, to tackle other problems arising from contexts that are only apparently far from mathematics. Indeed, real networks are often modelled as graphs, in order to be studied. One of the most intuitive examples is *road transportation*, where streets are represented as edges (or arcs) of an undirected (directed) graph, whereas their intersection points are the vertices.

Among the measures of interest, there is network reliability, that is, the capacity of a network to offer certain services even during a failure. Reliability has an impact on the design, implementation, and evaluation of a network. For instance, it is useful to detect possible points of failure or to identify minimal cuts that would disconnect the network into several components (e.g., [Tsukiyama et al., 1980] and [Rausand, 2014]). This last application is the motivation of the work presented in Chapter 2 (see, in particular, Section 2.2). Also *electrical networks* have been studied in the literature, for instance to compute electrical-circuit parameters by using spanning trees, or to examine program flow by using cycles in a program-flow graph ([Read and Tarjan, 1975]). Nowadays, other kinds of networks, that are being more and more studied, are *social networks*. Here, some entities (individuals and/or organizations) are connected and interact with each other, according to some kind of relationships established. Particular communities can be composed that relate to specific structures in graphs. For instance, when every member is connected to all the others in a community, a clique can be used to model and study the phenomenon. Indeed, maximal-clique enumeration has often been applied (e.g., [Pan and Santos, 2008], [Modani and Dey, 2009], and [Conte et al., 2016]).

Other possible applications in *combinatorial optimization* involve listing interesting objects in matroids, such as bases, hyperplanes, flats of given rank, circuits through a given element, generalized Steiner trees, and multiway cuts in graphs ([Khachiyan et al., 2005]). In *operations research*, there have been some relevant results in linear programming about the generation of all the vertices of a polyhedron ([Khachiyan et al., 2009]), whereas, in case of polytopes, the problem is still open ([Khachiyan et al., 2005]). These applications are part of the motivation of the study of the problems investigated in Chapter 3 (in particular, see Section 3.2 for more details).

Enumeration is also very efficient in *data mining*, e.g., to find all densely connected structures, to perform similarity analysis, or to identify patterns ([Uno, 2016]). Patterns and motifs are quite important in *computational biology* too. Maximal-clique enumeration algorithms have also been used to find cis-regulatory motivs ([Baldwin et al., 2004]), in particular to observe the actions of a large number of genes in response to any experimental stimulus ([Abu-khzam et al., 2005]). In this case, enumeration was preferred to clustering because it allowed a better representation of the genes behaviour, letting transcript membership in multiple cliques, not forcing genes to belong to one cluster only, and avoiding to determine the number of clusters in advance, losing critical information. Another interesting application in biology concerns metabolic networks and the enumeration of all pathways to reach a set of desired targets by converting a set of source molecules ([Liu et al., 2015] and [Ravikrishnan et al., 2018]).

In *chemistry*, enumeration algorithms have been utilized to represent structures in a given family, such as the alkane molecular family. Indeed, alkanes can be represented as trees whose nodes, corresponding to carbonium atoms, have degree less than or equal to 4; edges are instead the primary links of the molecules. The problem of enumerating all isomeric acyclic structures in alkanes was already tackled in 1875 by Cayley, who manually listed the alkane isomers and alkyl radicals with up to 13 carbonium atoms ([Rains and Sloane, 1999]). More recently, it was addressed again by [Aringhieri et al., 2003].

Enumeration algorithms have been widely exploited in *databases*, especially when the goal is to efficiently list the results of a query, considering the input and output size of the data. [Bagan et al., 2007] were interested in finding the class of queries that can be computed with linear pre-processing time and constant delay in enumerating the results. An analogous goal was pursued by [Carmeli and Kröll, 2019], in order to identify the structures in conjunctive queries that can be answered with near-optimal time guarantees. Database management is not the only area of informatics where one can find enumeration algorithms. In *coding theory*, [Tomita et al., 2019] listed single-deletion correcting codes by means of maximum cliques. Speaking about *cryptography*, [Veyrat-Charvillon et al., 2013] investigated the impact of key enumeration in cryptanalytic contexts to analyse the complexity of attacks when partial information of key bytes is available. In *system and process modelling*, [Cordone et al., 2005] proposed a partitioning algorithm to enumerate in a Petri net all minimal siphons, i.e., special substructures useful to identify deadlock-prevention policies.

To consult an evergoing list of enumeration algorithms, as well as their several possible applications, that go beyond the few examples proposed here, I refer the interested reader to [Wasa, 2019], who has been collecting enumeration algorithms since 2016. Up to now (December 2021), the list contains 517 enumeration problems, discussed and studied in 349 different scientific papers.

In general, for more details about enumeration algorithms, I suggest the interested reader to look at [Marino, 2015], [Grossi, 2016], [Kiyomi, 2016], and [Uno, 2016] (the last three all contained in the book entitled "*Encyclopedia of Algorithms*", edited by [Kao, 2016]).

1.3 Complexity classes of enumeration problems

Since the number of solutions of an enumeration problem could be exponential in the size of the input instance, performances of enumeration algorithms are measured differently than procedures for decision and optimization problems. Ad hoc complexity classes have been introduced to take into account both the input and the output sizes ([Johnson et al., 1988]). In particular, a parameter used in the evaluation is the total number of solutions, considered as an invariant ([Uno, 2016]). When this number is small, we can expect an efficient algorithm to terminate in a short time. Otherwise, we can allow the algorithm to spend more time ([Marino, 2015]). In what follows, I summarize the main classes described by [Capelli and Strozecki, 2017]. I restrict to problems with a finite total number of solutions.

EnumP is the class of all enumeration problems for which there exists an algorithm that, given an input instance and a possible output, decides the correctness of the output in polynomial time in the input and the output sizes. The class EnumP-complete contains those problems in EnumP to which any problem in EnumP reduces.

When the time required to compute and output all the possible solutions is polynomial in the size of the input and in the total number of solutions, we talk about *output-polynomial* algorithms ([Johnson et al., 1988]). The class of enumeration problems that admit such procedures is denoted by OutputP. EnumP and OutputP are analogous to NP and P, respectively. Also, OutputP = EnumP if and only if P = NP ([Capelli and Strozecki, 2017]).

One could be also interested in evaluating the time needed to output the first k solutions. We talk about *incremental-polynomial time* if the k-th output can be obtained in polynomial time in k itself and in the size of the input instance. The corresponding class is denoted by IncP, which is also defined as the class of problems solvable by an algorithm with a delay polynomial in the number of the already computed solutions and in the size of the input.

A subclass of IncP is DelayP, whose problems admit *polynomial-delay* algorithms, i.e., the delay between two consecutive outputs is polynomial in the size of the input instance and it is independent from the output. In other words, the delay when outputting two solutions is regular, at most polynomial. This implies an output-polynomial algorithm. Indeed, in a polynomial total-time algorithm, the delay between two consecutive solutions has to be polynomial on average ([Marino, 2015]). This case would also allow efficiently outputting all solutions in a specified order, such as lexicographic. When the delay is invariable and independent both from the input and the output, then it is *constant*. Finally, an enumeration algorithm is said to be *output-linear* if it terminates in linear time in the size of the input instance and the total number of solutions.

In terms of space complexity, the goal is to design enumeration algorithms that need polynomial space.

For a more comprehensive catalogue of known complexity classes for enumeration problems, their classification and hierarchy, I refer the interested reader to [Capelli and Strozecki, 2017].

1.4 Designing enumeration algorithms

To list all feasible solutions of an enumeration problem, we need to decide how to move in its search space. When efficiency is not compromised, for instance when dealing with small-size instances, a brute-force method can be appropriate to use. Through such an algorithm, we can guess every possible choice, list all candidate solutions, and output the feasible ones only. Another approach could be to enlarge already computed solutions and remove the isomorphic ones ([Marino, 2015], [Uno, 2016]). However, some conditions may be imposed by the problems themselves, such as having an exponential number of choices to be considered, without any guarantees to get a feasible solution. Thus, there may be the risk of performing a huge number of useless operations. That is why enumeration algorithms are usually designed to aim at efficiency, in order not to explore the whole search space ([Conte, 2018]). How can we smartly visit the search space? A possible way, given a computed solution, is to define a solution neighbourhood, in order to reach all the close solutions by moving iteratively through the neighbourhoods. Other methods are developed to avoid duplication or to define the canonical form of a solution ([Uno, 2016]).

In the following subsections, I recall some high-quality techniques for enumeration used (and also combined) to design more complex algorithms.

1.4.1 Backtracking

The *backtracking* technique derives straightly from backtracking programming. Initially, it has been proposed to solve problems such as finding all solutions to the eight queens problem on a chessboard or computing all simple cycles in a network ([Floyd, 1967]). It is a systematic procedure that relies on a depth-first search approach. Backtracking is usually applied to problems where the goal is to list minimal (or maximal) elements of a given set, according to some criteria. First, we define an ordering (e.g., lexicographic) of the elements of the set. Then, starting from an empty solution, we examine each element to decide whether to include it or not in the partial solution we are building. When we have made a decision for every element, we evaluate the solution obtained and, if feasible, we output it. Then, we "backtrack", by changing the decision about the last element in the ordering and checking the new obtained solution. *"Whenever we have tried both including and excluding an element, we back up to the previous element, change our decision, and move forward again"* ([Read and Tarjan, 1975], page 237). The procedure can also backtrack when a partial solution does not satisfy the conditions of the problem. In this way, useless operations that would not lead to any output are skipped. More generally, the backtracking move consists in first restoring the partial feasible solution computed in a given point, and then trying other alternatives not already considered ([Floyd, 1967]). To avoid duplication, elements are usually indexed. Whenever it is possible to apply the backtracking schema, we obtain a polynomial-delay algorithm, whose space complexity is also polynomial ([Marino, 2015]).

Examples of recent applications of backtracking are [Wang et al., 2021] and [Gianinazzi et al., 2021], to enumerate distributed subgraphs in static and dynamic data graphs, and to list k-cliques in sparse graphs using parallelism, respectively.

1.4.2 Binary partition

Binary partition is a recursive partition algorithm, similar to branch-and-bound methods ([Uno, 2016]). Consider a subset of the set of solutions, composed of all solutions satisfying a given property. This subset is output only if it is a singleton, otherwise it is split in two non-empty sets, whose solutions are characterized by two disjoint properties ([Marino, 2015]). The partition process requires a polynomial time and the number of iterations is bounded by twice the number of possible solutions. Thus, an algorithm based on binary partition is output-polynomial time. If the height of the recursion tree is polynomial in the size of the input, then the partition process is polynomial-delay. This technique is usually used to design algorithms that are fast in practice, or where the number of solutions can be bound in the worst-case scenario ([Conte and Uno, 2019]).

Binary partition has been recently applied by [Conte et al., 2020] to enumerate s-d separators in directed acyclic graphs (also with applications to temporal graphs), and by [Kurita et al., 2021b] to list independent sets in graphs with bounded clique number.

1.4.3 Reverse search

When there exists a natural neighbourhood relation between the solutions to be output, or a natural partial order, one way to design an efficient enumeration algorithm is by applying *reverse search* by [Avis and Fukuda, 1996]. Given an enumeration problem, we can think about its possible solutions as the vertices of a graph. Two solutions are adjacent if one belongs to the neighbourhood of the other. To iteratively move from one solution to another, we apply a local-search method. We define the *trace* of this local search as the directed graph derived from the graph of solutions by applying the local search itself. In other words, the trace contains all the edges of the original graph used in the local search. The length of the longest directed path in the trace is called the *height* of the trace. When the local search is finite, the trace is a directed spanning forest of the graph of solutions. More specifically, we need to define a parent operation, in order to reduce a node in the tree to its unique parent node. A child operation, defined by inverting the parent operation, determines if an object is a valid child of a given parent object. It is this parent-children relationship that induces a forest. Each component (i.e., a tree) contains exactly one solution as root. Starting from this solution, we can traverse the component by applying depth-first search and the child operation, in order to generate the children nodes and output the feasible solutions. The traversal depends on how the local search is defined. Thus, the most crucial part to design a reverse-search-based algorithm is specifying how to obtain a suitable parent-child relationship between the solutions. We do not need to know the whole graph of solutions, but we can rely on an adjacency oracle that, given a vertex, returns an adjacent vertex only once. In this way, the number of iterations is equal to the number of solutions, that is, the cost per iteration also corresponds to the cost to output a solution. If finding the next adjacent vertex, by applying the local search method and calling the adjacency oracle, is polynomial in the input size, then also the resulting computation time per iteration is polynomial. This would imply an output-polynomial-time algorithm. When the local search and the adjacency oracle are independent from the input size, then the time complexity is linear in the output size (i.e., the number of possible solutions). Since the backtracking operation is performed by computing the parent node not by using a stack, the space complexity needed is polynomial in the input size.

Some examples of recent applications of reverse search are an efficient enumeration of dominating sets for sparse graphs by [Kurita et al., 2021a], and the listing of all maximal strongly-connected cliques in a directed graph by [Conte et al., 2021].

1.4.4 A note on parallel computing

Given their goal of listing all feasible solutions, possibly in an exponential number, enumeration problems can become computationally intractable very quickly. Thus, *parallel computing* becomes necessary when scaling to large input instances. In this computing architecture, several processors are exploited to simultaneously perform operations or processes. An input large problem is usually divided into smaller parts or subproblems, assigned each to a different processor. The main reason to rely on parallel computing is thus to improve efficiency and speed.

When designing parallel algorithms, one has to consider including a *dynamic load-balancing* mechanism, i.e., how to dynamically assign subproblems to processors when they are free and run out of work. Also, a *checkpointing* procedure may be needed, to save the state of a computation in an external memory, in order to be able to interrupt and resume computation when required. Dynamic load balancing and checkpointing support a dynamically changing number of processors, e.g., when machines need to be turned off or if new processors are available ([Marzetta, 1998], [Brüngger et al., 1999]). Another function common to parallel programs is a *termination detection*. One of the simplest paradigms to implement parallel computing is called *master-slave*. A master processor is usually in charge of managing sequential tasks, such as reading the input problem, initializing the other processors, and distributing the initial work. It also receives results of computation from slaves and passes them needed information. Processors can be independent from each other (in this case, we say they are *embarrassingly parallel*), or they may need to communicate, in order to exchange data or results.

In the context of enumeration, parallelization can be directly integrated in the structure of an enumeration algorithm by taking care of communication between processors, load balancing, data sharing, and synchronization (see, e.g., [Weibel, 2010]), or it can be achieved by adding a separate wrapper layer (see, e.g., [Marzetta, 1998]). The latter method is usually preferred, since it enhances the reuse of existing sequential algorithms with minimal changes ([Avis and Jordan, 2018]), and it allows for a simpler maintenance of both the parallel techniques adopted and the underlying enumeration algorithms implemented ([Avis and Roumanis, 2013]).

According to the heaviness of the wrapper and the amount of programming effort, some enumeration techniques can be more suitable for efficient parallelization. Search algorithms are easily parallelized when, for instance, any node is expanded by generating some neighbors. If each neighbor defines a new search problem of the same type, then it can be processed almost independently from the others ([Brüngger et al., 1999]). A branching operation, as in binary partition, can offer a very natural division of work between processes ([Weibel, 2010]). However, we must consider whether the generated subproblems must exchange data among them (as in branch-and-bound, when bounds have to be updated and shared among different processors).

Among the techniques we have just described in this subsection, the one really suitable for parallelization is reverse search, for the following reasons. First, it offers a lack-of-memory property. Indeed, it is not required to store more than one node of the tree at any given time, and no database is required for visited nodes ([Avis and Roumanis, 2013], [Avis and Jordan, 2018]). This allows the method to be restarted from any node in the reverse search tree, by relying only on a description of this node. After a restart, all remaining nodes of the tree are generated. This is typically not achievable with backtracking techniques, where restart is possible only with a complete database of visited nodes and the backtrack stack to the root of the search tree ([Avis and Jordan, 2018]). With reverse search, instead, checkpointing is easily obtained through small and simple files, with potentially different parameters or number of processes. Then, processors do not need to interact with each other, thus the communication overhead is minimal, to collect the computed output. It is thus enough to design a load-balancing method to parallelize reverse search.

Several implementations of parallel reverse search have been proposed in the last twenty-five years (see, e.g., [Marzetta, 1998], [Brüngger et al., 1999], [Weibel, 2010], [Avis and Roumanis, 2013], [Jordan et al., 2017], and [Avis and Jordan, 2018]). They all share the fact that their efficiency depends on the structure of the underlying enumeration problem and the balancing of the reverse search tree. [Avis and Jordan, 2021] showed how to abstract and generalize these ideas for other tree-search programs, such as backtracking and branch-and-bound, supporting sharing information between processes.

1.5 Analysing enumeration algorithms

To evaluate the efficiency of an algorithm, two techniques we can rely on are *asymptotic analysis* and *amortized analysis* ([Cormen et al., 2009]). In the former, first we provide an upper bound to the time required by each operation; then, to obtain the overall complexity, we sum up the individual contributions. In other words, asymptotic analysis concerns how the performance of a given operation scales when the input size increases. In the latter, instead, we evaluate the time required by a sequence of operations ([Tarjan, 1985]). We consider the average performance of each operation in the worst case, without saying anything about the cost of a specific operation in that sequence. In this case, we are interested in understanding how the average of the performance of all the operations considered would change when the input size increases. Since the former technique can be too pessimistic in the worst-case scenario, the latter is usually preferred to get a more precise analysis at micro level.

There are three main approaches to perform amortized analysis. In the *aggregate method*, to obtain the amortized cost, we divide the overall time needed to perform a sequence of operations by the number of operations itself. In the *accounting method*, we maintain an account with the underlying data structure used by the algorithm. Initially, the account contains no credits (or charges). Each operation is charged with an amount of credits that may not correspond to the actual cost of the operation. If we overcharge it, the excess charge will be deposited to the account as credit. For some operations, we may charge nothing: in such a case, we will use some charges available as credit. Analysis ensures that the account is never at debit. The amortized cost for each type of operation corresponds to the amount charged for that type, and it is an upper bound on the actual cost for any sequence of operations. Finally, in the *potential function method*, a potential measure is assigned to the data structure, according to its current configuration in terms of structural parameters (e.g., the number of elements). The amortized cost of an operation is equal to its actual cost, plus the difference between the potential before and after the operation.

Since enumeration algorithms are often quite efficient in practice, with respect to their theoretical bounds ([Uno, 2016]), amortized analysis is commonly used to provide a more realistic evaluation. As seen in Section 1.4, enumeration algorithms are usually based on recursive approaches. Starting from a problem, an iteration of an enumeration algorithm can generate several subproblems of smaller size. To evaluate the efficiency of an enumeration algorithm, we can consider the time spent by the procedure to solve all the subproblems. We assume a tree-shaped recursive algorithm, where the root node corresponds to the original problem and the internal nodes and the leaves are smaller subproblems. We know that the number of iterations exponentially increases as we go deep into the recursion. It is difficult to know or even estimate this total number of iterations and the overall computational time. Usually, we observe that subproblems generated by a node are smaller than their input problems, and that at the bottom of the recursion there are many iterations taking very short time. Instead, near the root of the recursion, the number of iterations is small but the time spent is greater. This implies that the amortized computation time per iteration, or even per solution, is short. This characteristic is called "bottom-wideness" and, when present, makes *basic amortization* suitable to be used, even if it is not enough to obtain good amortized bounds ([Marino, 2015]). Anyway, when the structure of the recursion tree is regular (i.e., when every internal node has two children, every leaf has the same depth, and the cost of each node is proportional to the height of the node), then the computation time constantly decreases and the amortized cost is reduced. When the recursion tree is more biased, we focus more on analysing local structures. For instance, the cost of a node can be charged to it and its children. In this case, we talk about *children amortization*. In the *push-out amortization* by [Uno, 2016], the computation of a node is directly charged to its children. When this technique can be applied and the amortized cost per iteration is low, it means that there are nodes with many descendants. On the contrary, the average computation time will be long with only a few descendants.

Chapter 2

Listing the bonds of a graph in $\widetilde{O}(n)$–delay

"In practical life, time is a form of wealth with which we are stingy [...] Saving time is a good thing because the more time we save, the more we can afford to lose. Quickness of style and thought means above all agility, mobility, and ease, all qualities that go with writing where it is natural to digress, to jump from one subject to another, to lose the thread a hundred times and find it again after a hundred more twists and turns."

Italo Calvino, *Quickness*, in *Six Memos for the Next Millennium*

In this chapter, I investigate the problem of listing the bonds of an undirected graph. By starting from the state of the art, I propose two new algorithms.

2.1 Introduction

Consider a connected graph $\mathcal{G} = (V, E)$ with $n := |V|$ vertices and $m := |E|$ edges. When S is a subset of V such that neither S nor $\overline{S} := V \setminus S$ is empty, then $\{S, \overline{S}\}$ is called a *bipartition* of V. The set of those edges in E having one endpoint in S and the other in $\overline{S}$, denoted by $\delta_{\mathcal{G}}(S, \overline{S})$, is called a *cut* of $\mathcal{G}$. Since $\mathcal{G}$ is connected, $\delta_{\mathcal{G}}(S, \overline{S})$ is nonempty. Moreover, for every cut $E' \subseteq E$ there exists a unique bipartition $\{S, \overline{S}\}$ such that $E' = \delta_{\mathcal{G}}(S, \overline{S})$, in which case S and $\overline{S}$ are called the *shores* of the cut E'.

The cuts of $\mathcal{G}$ are precisely those sets of edges having even intersection with every cycle of $\mathcal{G}$. A *cycle* of $\mathcal{G}$ is any $F \subseteq E$ such that the graph (V, F) is Eulerian, i.e., every vertex in V is incident to an even number of edges in F. This number is either 2 or 0 for every node when F is a minimal nonempty cycle. Cycles and cuts are two orthogonal subspaces of $GF(2)^E$ that, together with minimal cycles and *bonds*, have been studied since the birth of graph theory, also in connection with Kirchhoff's laws, planar graphs, binary matroids and clutters.

A *minimal cut*, or *bond*, is a cut $E' \subset E$ such that no proper subset $E'' \subsetneq E'$ is itself a cut of $\mathcal{G}$. Equivalently, $\delta_{\mathcal{G}}(S, \overline{S})$ is a bond if and only if the two induced subgraphs $\mathcal{G}[S]$ and $\mathcal{G}[\overline{S}]$ are both connected, i.e., $\mathcal{G} \setminus E'$ has precisely two connected components. We denote by $\mu = \mu(\mathcal{G})$ the number of different bonds in $\mathcal{G}$. When $\mathcal{G}$ is a path, then $\mu = n - 1$; when $\mathcal{G}$ is a clique, then $\mu = 2^n - 2$. A set $S \subsetneq V$ is called a *bond-shore* when $\delta_{\mathcal{G}}(S, \overline{S})$ is a bond. When s and t are two vertices with $s \in S$ and $t \in \overline{S}$, then $\delta_{\mathcal{G}}(S, \overline{S})$ is called an *s, t-bond* and S is called an *s, t-bond-shore*.

In optimization, we may be interested in finding a bond where, for instance, $|\delta_{\mathcal{G}}(S, \overline{S})|$ is minimum, as in PROBLEM 2.1.

PROBLEM 2.1 (Compute a bond of $\mathcal{G}$ with minimum number of edges). Given a connected graph $\mathcal{G} = (V, E)$, compute a bond of $\mathcal{G}$, either in form of an edge-set $\delta_{\mathcal{G}}(S, \overline{S})$ or in form of a bond-shore S, such that the number of edges $|\delta_{\mathcal{G}}(S, \overline{S})|$ is minimum.

If the graph is weighted, we may adapt the objective function of PROBLEM 2.1 to consider the sum of the weights instead of the cardinality of the edges in the bond.

In this work, I focus on listing all bonds, i.e., ignoring the objective function. In particular, I present new algorithms to address the following two fundamental problems.

PROBLEM 2.2 (Listing all bonds of $\mathcal{G}$). Given a connected graph $\mathcal{G} = (V, E)$, output all bonds of $\mathcal{G}$, each one either in form of an edge-set $\delta_{\mathcal{G}}(S, \overline{S})$ or in form of a bond-shore S.

PROBLEM 2.3 (Listing all s, t-bonds of $\mathcal{G}$). Given a connected graph $\mathcal{G} = (V, E)$ and two distinct vertices $s, t \in V$, output all s, t-bonds of $\mathcal{G}$, each one either in form of an edge-set $\delta_{\mathcal{G}}(S, \overline{S})$ or in form of an s, t-bond-shore S.

The current state-of-the-art algorithm to solve PROBLEM 2.2 and PROBLEM 2.3 is the one by [Tsukiyama et al., 1980]. This exploits binary partition to output each s, t-bond in $O(m)$ per bond, being thus classified as an $O(m)$-delay algorithm.

Observing this result, we can ask ourselves: how can it be improved it? Is it possible to develop an enumeration algorithm whose bound, rather than depending on the number of edges m, is expressed only in terms of the number of vertices n? The answer is yes. Here, I present the first $\widetilde{O}(n)$-delay algorithm for PROBLEM 2.2 and PROBLEM 2.3, assuming only the two bond-shores are output for every bond. In case the entire edge-set of every bond is explicitly provided, then I can claim the following performance: every time the procedure finishes outputting a bond $\delta_G(S, \overline{S})$, then $|\delta_G(S, \overline{S})| + \widetilde{O}(n)$ is the time elapsed since the previous output was complete or the algorithm started. Disregarding the polylogarithmic factors hidden in the $\widetilde{O}$ notation, this is the first output-linear algorithm to list bonds. To obtain these bounds, I exploit a slightly different branching strategy than [Tsukiyama et al., 1980], by focusing more on the role of cut-vertices, and I rely on dynamic data structures, some described by [Holm et al., 2001].

This chapter is structured as follows. In Section 2.2, I motivate the study of these problems by presenting a few relevant applications. In order to fully describe the two algorithms, in Section 2.3, I give some graph-theory definitions and notions on biconnectivity, needed in the rest of the chapter. In Section 2.4, I recall the main related work. In Section 2.5, first I show how to reduce the two problems above to the one effectively tackled by [Tsukiyama et al., 1980]. Then, by recalling the key lemmas used in [Tsukiyama et al., 1980], I introduce the main ideas, lemmas, and invariants at the base of the approach I propose, and I illustrate in what these differ from [Tsukiyama et al., 1980]. Only then, I illustrate the first algorithm, which has the same complexity as [Tsukiyama et al., 1980]. In order to improve it, in Section 2.6, I recall some relevant results on dynamic data structures by [Holm et al., 2001], to support connectivity and biconnectivity queries in graphs subject to change in the fully-dynamic paradigm. I also introduce a third ad hoc and relatively simple dynamic data structure to operate over a tree and a given subset of its vertices. In Section 2.7, I describe the second algorithm and analyse its complexity. Finally, in Section 2.8, I discuss other possible research directions concerning the enumeration of bonds.

2.2 Applications

These problems find application in many research areas, starting of course from connectivity in *graph theory* (see, e.g., [Tutte, 1966] or [Diestel, 2017]). Recently, [Duarte et al., 2021] addressed the problem of computing the largest bond in general graphs and also in specific classes, such as planar, bipartite, and split graphs. Graphs are widely used to represent networks. Typically, in this context, bonds have been studied to assess the structural vulnerability and reliability of a *network*, i.e., the probability of a given network to succeed in performing its tasks, even when some of its components, such as nodes and links, may fail (e.g., [Van Slyke and Frank, 1971], [Colbourn, 1987], [Shier, 1991], [Lin et al., 2003], and [Gaur et al., 2021]).

Reliability is the motivation that also brought researchers from other different disciplines to study minimal cuts.

In *logistics and supply chain management*, distribution networks are the main instrument to deal with the storage and transportation of products. In this context, as described in [Niu et al., 2017] and [Nguyen and Lin, 2021], we may want to compute the probability that the nodes of the network remain connected, or that the destination can be reached within a specified deadline, or also that a given flow demand can be transported from a given source to a given destination.

In *production networks and process management*, we can be interested in identifying the parts of the systems that may crash and interrupt a process. This is the case, for example, of a subsea system for oil and gas production ([Cheliyan and Bhattacharyya, 2018]).

In *bioinformatics*, minimal cuts have been used by [Klamt and Gilles, 2004] in biochemical reaction networks, in order to identify a set of reactions whose inactivation would lead to a failure in certain network functions. Later, [Hädicke and Klamt, 2010] generalized their approach by specifying not only the functionalities to be disabled but also those to be preserved in the metabolic networks. Starting from these results, [Gerstl et al., 2016] investigated the robustness of a metabolic network, that is, its ability to perform normally under the presence of perturbations. Moreover, [Apaolaza et al., 2017] applied minimal cuts to predict and exploit synthetic lethality in cancer metabolism.

In *engineering*, another application sees minimal cuts at the basis of a software fault-tree analysis, utilized to investigate the safety of digital reactor protection systems in nuclear power plants ([Jung et al., 2020]). In *computer vision*, minimal cuts are exploited in some schemes designed for image segmentation ([Eriksson et al., 2006], [Peng et al., 2013]).

2.3 Preliminaries

Here I recall a few needed notions and lemmas from graph theory, by especially focusing on connectivity and biconnectivity.

We work with finite undirected graphs without self-loops. Consider a connected graph $\mathcal{G} = (V, E)$ with $n := |V|$ *vertices* and $m := |E|$ undirected pairs of vertices called *edges*. We often write uv as shorthand for an edge $(u, v) \in E$. To avoid ambiguity, we always use the term *node* to indicate a subproblem of a recursion tree built by an enumeration algorithm, whereas the term *vertex* refers to an element of the set V in $\mathcal{G}$.

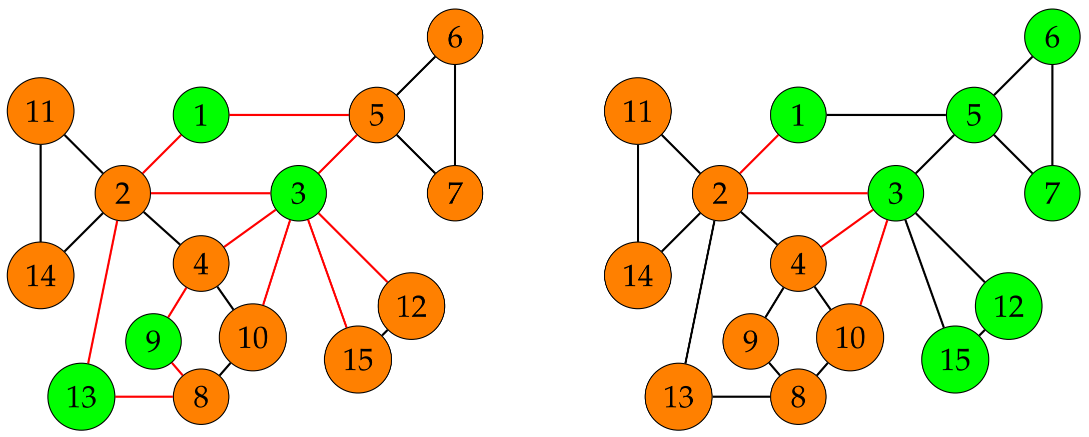
Connectedness is a monotone property in the sense that, if $E' \subseteq E$, then the connectedness of (V, E') implies the connectedness of (V, E). A graph $\mathcal{G}' = (V', E')$ is called a *subgraph* of $\mathcal{G}$ if $V' \subseteq V$ and $E' \subseteq E$. The subgraph is called *spanning* if $V' = V$. Given a set of edges $F \subseteq E$, we denote by $\mathcal{G} \setminus F$ the spanning subgraph of $\mathcal{G}$ having $E \setminus F$ as its edge set and $V(\mathcal{G} \setminus F) = V$. For $e \in E$, we write $G \setminus e$ as a shorthand for $\mathcal{G} \setminus \{e\}$. When $S \subseteq V$, $\mathcal{G}[S]$ denotes the subgraph of $\mathcal{G}$ having S as its vertex set and all edges in E with both endpoints in S as its edge set. This is the maximal subgraph of $\mathcal{G}$ with $V(\mathcal{G}[S]) = S$ and is called *the subgraph of $\mathcal{G}$ induced by S*. Being also the maximal subgraph of $\mathcal{G}$ avoiding the vertices in $\overline{S}$, it is also denoted by $G \setminus \overline{S}$ and considered as *the subgraph obtained from $\mathcal{G}$ after removing the vertices in $\overline{S}$*. For $v \in V$, we write $G \setminus v$ as a shorthand for $\mathcal{G} \setminus \{v\}$. Given a vertex $u \in V$, we denote by $N(u) := \{v \in V \mid uv \in E\}$ the *neighborhood* of u. For any subset S of V, $N(S) := \{v \in \overline{S} \mid uv \in E, u \in S\}$.

A *connected component* of $\mathcal{G}$ is a maximal set $S \subseteq V$ such that $\mathcal{G}[S]$ is connected. The connected components of $\mathcal{G}$ form a partition of V. The number of connected components increases by at most one when one single edge is removed. As such, $m \geq n - 1$ for every connected graph. A *forest* is an *acyclic* graph $\mathcal{F}$, that is, $|E(\mathcal{F}[S])| \leq |S| - 1$ for every $S \subseteq V(\mathcal{F})$. In particular, $m = n - k \leq n - 1$ for every forest, where k is the number of connected components of $\mathcal{F}$. A *tree* is a connected forest. Equivalently, trees are those forests for which $m = n - 1$. A *spanning tree* τ of $\mathcal{G}$ is a spanning subgraph of $\mathcal{G}$ which is a tree; there is included at least one if and only if $\mathcal{G}$ is connected. Given a vertex v of $\mathcal{G}$, we denote by τ_v the outward-rooted orientation of τ.

When V can be partitioned into the two sets S and $\overline{S}$, such that both $\mathcal{G}[S]$ and $\mathcal{G}[\overline{S}]$ have no edges, then $\mathcal{G}$ is called *bipartite of color classes S and $\overline{S}$*. Given two disjoint vertex sets $S, T \subseteq V$, then $\delta_{\mathcal{G}}(S, T)$ denotes the edge set $\{uv \in E : u \in S, v \in T\}$. Notice that the spanning subgraph $(V, \delta_{\mathcal{G}}(S, \overline{S}))$ is bipartite with color classes S and $\overline{S}$ and that the cut $C := \delta_{\mathcal{G}}(S, \overline{S})$ intersects every spanning tree of $\mathcal{G}$. Therefore, $\mathcal{G} \setminus C$ is not connected and every connected component of $\mathcal{G} \setminus C$ is either contained in S or in $\overline{S}$. As defined in the introduction, a bond is a minimal non-empty cut. When a cut $F = \delta_{\mathcal{G}}(S, \overline{S})$ is a bond of $\mathcal{G}$, then both S and $\overline{S}$ are connected components of $\mathcal{G} \setminus F$ (for otherwise $\mathcal{G} \setminus (F \setminus \{f\})$ could not be connected for any $f \in F$). Therefore, every bond is a cut whose shores induce two connected subgraphs, as proved by [Jensen and Bellmore, 1969].

Lemma 2.3.1 (Double connectivity – [Jensen and Bellmore, 1969]). *Assume $\mathcal{G} = (V, E)$ is connected. Then, a cut $C := \delta_{\mathcal{G}}(S, \overline{S})$ of $\mathcal{G}$ is a bond if and only if the induced subgraphs $\mathcal{G}[S]$ and $\mathcal{G}[\overline{S}]$ are both connected (i.e., $\mathcal{G} \setminus C$ has only two connected components).*

Proof. Assume $\mathcal{G}[S]$ and $\mathcal{G}[\overline{S}]$ are both connected and let T (resp., $\overline{T}$) be a spanning tree of $\mathcal{G}[S]$ (resp., $\mathcal{G}[\overline{S}]$). We need to show that $\mathcal{G} \setminus C'$ is connected for every $C' \subsetneq C$. Take any edge $e \in C \setminus C'$ and notice that e together with T and $\overline{T}$ comprise a spanning tree for $\mathcal{G} \setminus C'$. □



(a) Given $\{S_1, \overline{S_1}\}$, $\delta_{\mathcal{G}}(S_1, \overline{S_1})$ is a *cut*. (b) Given $\{S_2, \overline{S_2}\}$, $\delta_{\mathcal{G}}(S_2, \overline{S_2})$ is a *bond*.

FIGURE 2.1: Difference between a generic cut and a minimal cut (i.e., a bond).

A vertex v of a connected graph $\mathcal{G} = (V, E)$ is called a *cut-vertex* of $\mathcal{G}$ if $\mathcal{G} \setminus v$ is not connected. When no cut-vertex exists, the graph is called *biconnected*. A *biconnected component* (or *block*) of $\mathcal{G}$ is a maximal set $S \subseteq V$ such that $\mathcal{G}[S]$ is biconnected. When S is a (bi)connected component of $\mathcal{G}$, we may also refer to the graph $\mathcal{G}[S]$ as a (bi)connected component of $\mathcal{G}$. While the connected components of $\mathcal{G}$ partition V, the biconnected components overlap in that the cut-vertices of $\mathcal{G}$ belong to more than one biconnected component. Notice, however, that every edge of $\mathcal{G}$ belongs to precisely one biconnected component of $\mathcal{G}$. As an example, Figure 2.2 shows a connected graph, its three cut-vertices in red, and each of its four biconnected components shadowed with a different color. Notice that the edges of a cycle (e.g., $(4,9) - (9,8) - (8,10) - (10,4)$ in Figure 2.2), all belong to a same block since the removal of a single vertex cannot break a cycle apart.

A vertex incident with precisely one edge is called *a leaf*.

Lemma 2.3.2 (Leaf of a tree). *Given a connected graph $\mathcal{G} = (V, E)$, let τ be a spanning tree of $\mathcal{G}$. Then, no leaf of τ is a cut-vertex of $\mathcal{G}$.*

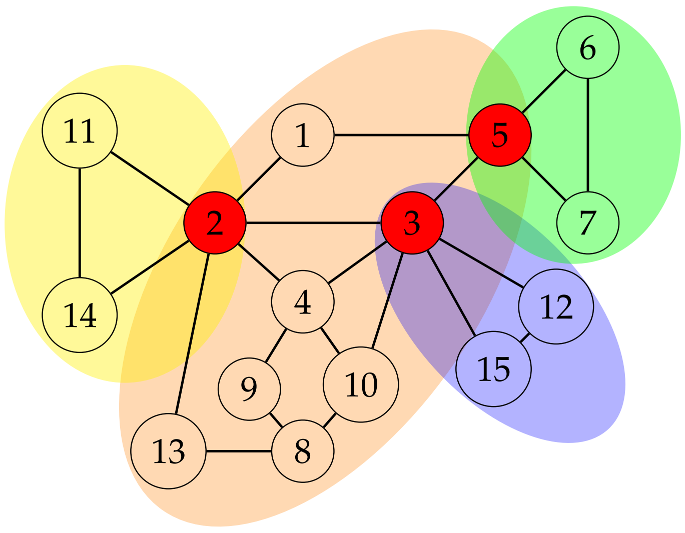


Figure 2.2: The connected graph of Figure 2.1, its three cut-vertices ($\{2, 3, 5\}$), and its four biconnected components ($\{1, 2, 3, 4, 5, 8, 9, 10, 13\}$, $\{2, 11, 14\}$, $\{3, 12, 15\}$, and $\{5, 6, 7\}$).

Proof. Consider any leaf x of τ. Then, $\tau \setminus x$ is still a tree, spanning and certifying the connectivity of $\mathcal{G} \setminus x$. □

A well-known linear-time algorithm by [Hopcroft and Tarjan, 1973] computes the cut-vertices of a graph by implementing a depth-first search visit, as described by [Tarjan, 1971]. Consider a depth-first search tree of the graph, as described in Algorithm 2.1. Then there are two possible conditions for a vertex u to be a cut-vertex: either it is the root of the tree and has at least two children that correspond to two disjoint subgraphs, or it has a child v such that no vertex in the subtree rooted at v has a back-edge to one of the ancestors of u. The former case is straightforward to check, whereas the latter requires two additional attributes to be defined during the traversal of the graph. The algorithm by [Hopcroft and Tarjan, 1973] keeps track of the order of vertices discovered, from the earliest to the latest: every time a new vertex is reached, it is marked with a *discovery* time (i.e., a counter) needed to detect back-edges. Then, to each vertex, they assign a *lowpoint* value, in order to maintain the earliest possible vertex accessible from that vertex by traversing any back-edge. Hereafter, Algorithm 2.2 wraps Algorithm 2.1in order to return all cut-vertices of $\mathcal{G}$.

Algorithm 2.1: DFS($\mathcal{G}$, u, $lowpoint$, $discovery$, $time$) – [Hopcroft and Tarjan, 1973].

```
1. Set $discovery[u]$ and $lowpoint[u]$ to $time$. For each child $v$ of $u$:
  1.1. If $discovery[v] = -1$, then:
    1.1.1. Increment the number of children of $u$.
    1.1.2. Make the recursive call DFS($\mathcal{G}$, $v$, $lowpoint$, $discovery$, $time + 1$).
    1.1.3. Update $lowpoint[u] := \min\{lowpoint[u], lowpoint[v]\}$.
    1.1.4. If $u$ is the root of the DFS tree and has at least two children, or if
           $lowpoint[v] \geq discovery[u]$, then mark $u$ as a cut-vertex of $\mathcal{G}$.
  1.2. Otherwise, update $lowpoint[u] := \min\{lowpoint[u], discovery[v]\}$.
```

Algorithm 2.2: Computing the cut-vertices of a connected graph $\mathcal{G}$ – [Hopcroft and Tarjan, 1973].

```
1. For each vertex $u$ in $V$, initialize $lowpoint[u]$ and $discovery[u]$ to $-1$.
   Set $time$ to 0.
2. Given a vertex $u \in V$, call DFS($\mathcal{G}$, $u$, $lowpoint$, $discovery$, $time + 1$).
3. Return all vertices of $\mathcal{G}$ marked as cut-vertices.
```

Starting from a vertex u, for each child v of u the recursive algorithm checks if v has already been visited. If not, it increments the number of children of u and makes a recursive call on v. When the subproblem of v is closed, the algorithm updates the lowpoint value of u by taking the minimum between the current lowpoint of u and the lowpoint of v. The algorithm marks u as a cut-vertex if it has more than two children or if the lowpoint of v is greater or equal to the discovery time of u. Otherwise, if v has been already discovered in a previous recursive call, the algorithm updates the lowpoint of u by taking the minimum between the lowpoint of u and the discovery time of v. The algorithm keeps track of new vertices reached on a stack. When a cut-vertex is found, all the edges examined during the search are placed on a different stack in order to retrieve the biconnected component reachable through that cut-vertex. Since the algorithm performs a single traversal of the graph, the required time complexity is $O(n + m)$.

For more details, see also [Cormen et al., 2009] (problem 22-2).

2.4 Related work

Although the literature is full of works about cuts in graphs, not so many are about the computation and enumeration of bonds (as mentioned in Section 1.2, see [Wasa, 2019] for an online and periodically updated catalogue of enumeration algorithms). Moreover, different communities tend to use different terms, or to associate the same term with various notions. Indeed, beyond minimal cuts, bonds are also referred as *minimal cutsets* (see, e.g., [Martelli, 1976], [Arunkumar and Lee, 1979], [Abel and Bicker, 1982], [Yan et al., 1994], and [Lin et al., 2003]) or *cocycles* (see, e.g., [Harary, 1969]); sometimes the adjective *minimal* is avoided (see, e.g., [Liu and Edelberg, 1968], [Wilson, 1972], [Deo, 2017], and [Tsukiyama et al., 1980]), or *proper* is used instead (see, e.g., [Jensen and Bellmore, 1969] and [Bellmore and Jensen, 1970]). The term *bond*, as we mean it in this work, was used by [Bondy and Murty, 1976]. From this point onwards, we will only use this word to refer to the minimal cuts of a given connected graph $\mathcal{G} = (V, E)$.

The literature about listing bonds dates back to the 1960s, when [Jensen and Bellmore, 1969] were among the first authors to discuss minimal cuts in the context of reliability of complex systems. They proved by contradiction that a cut must induce two connected subgraphs in order to be minimal. In a successive work, [Bellmore and Jensen, 1970] also considered the number of bonds, stating that the maximum is reached with the clique graph, where there are 2^n ways to partition the vertices of $\mathcal{G}$, that become 2^{n-2} if we consider s, t-bonds (i.e., forcing s and t to be in different partitions). [Bellmore and Jensen, 1970] first developed a brute-force algorithm to check all possible $2^n - 2$ bipartitions. Then, they also proposed an enumeration algorithm based on the construction of a binary-search tree (i.e., a precursor of binary partition). At each recursive call, they checked connectivity requirements for the two induced subgraphs, in order to obtain a bond at each leaf.

[Martelli, 1976] defined a suitable algebra for bonds by using a semi-ring composed of two operations, that were, sum and multiplication. Then, he showed how to reduce the problem of enumerating all bonds to solving a system of linear equations by using Gaussian elimination. The method computed simultaneously the bonds between all pairs of vertices, but its efficiency depended on the implementation of the two operations. Nevertheless, the time required to output all bonds was output-linear, assuming the number of bonds μ was $\Omega(2^n)$.

Until now, the state of the art for listing bonds is the work by [Tsukiyama et al., 1980], which relied on the proof by [Jensen and Bellmore, 1969] on the two induced subgraphs generated by a bond. [Tsukiyama et al., 1980] designed and compared three versions of an algorithm to enumerate all s,t-bonds separating two given vertices s and t, based on the partition of the vertices of V. To output all the bonds in the graph, the most efficient version of their algorithms offers a linear delay $O(m)$ and requires $O(m)$ also in terms of the storage needed for computation.

Later, [Abel and Bicker, 1982] provided a $O(n^3 \cdot \mu)$ procedure to generate all μ s,t-bonds exactly once, by blocking any solutions that could be yielded from other bonds already computed.

[Yan et al., 1994] implemented the partitioning principle of [Bellmore and Jensen, 1970] without constructing a systematic binary-search tree. They presented a recursive labelling algorithm adapted from a dynamic-programming approach for the classical shortest route problem, for both undirected and directed graphs. Their method produced all s,t-bonds by eliminating redundancies through comparison. Moreover, they showed how the required time per bond decreases exponentially with the density of the graph. Unfortunately, even if applicable to both undirected and directed graphs, in the former case their approach was not completely correct by also generating non-minimal cuts.

Based on [Tsukiyama et al., 1980], [Lin et al., 2003] enumerated all s,t-bonds through a simple recursive algorithm. This approach merges the source s one by one with its adjacent vertices, and absorbs all vertices that cannot reach t without going through any vertex in the source set. They combined this procedure with Binary Decision Diagrams, in order to evaluate networks reliability and availability. Their approach was sped up by [Debieux et al., 2017], who had the idea to exploit cut-vertices to run the recursive merge algorithm separately on the biconnected components of the graph. They did not provide neither an implementation nor a complexity analysis, but just showed experimental results.

More recently, [Conte et al., 2018] investigated the number of s,t-bonds in an undirected connected graph $\mathcal{G}$, proving that there are $\Omega(m)$ s,t-bonds in any biconnected graph $\mathcal{G}$ for any choice of distinct vertices s,t.

2.5 Yet another $O(m)$–delay algorithm

In this section, I propose an alternative $O(m)$-delay algorithm for Problem 2.2 and Problem 2.3.

2.5.1 From bonds to S,T-bonds

Consider the following problem.

PROBLEM 2.4 (Listing all S,T-bonds of $\mathcal{G}$). Given a connected graph $\mathcal{G} = (V, E)$ and two disjoint sets $S, T \subsetneq V$, list the S,T-bonds of $\mathcal{G}$.

A bond $\delta_{\mathcal{G}}\left(S, \overline{S}\right)$ is called an *S', T'-bond* for every pair (S', T') with $S' \subseteq S$ and $T' \subseteq \overline{S}$. In this way, PROBLEM 2.4 has PROBLEM 2.2 as its special case with $S = T = \emptyset$, and PROBLEM 2.3 as its special case with $S = \{s\}$ and $T = \{t\}$.

However, this is not the right joint generalization to go for, since we next show that the corresponding decision problem is NP-complete.

PROBLEM 2.5 (Checking the existence of an S,T-bond of $\mathcal{G}$). Given a connected graph $\mathcal{G} = (V, E)$ and two disjoint sets $S, T \subsetneq V$, decide whether there exists an S,T-bond of $\mathcal{G}$.

The reduction is from the following problem.

PROBLEM 2.6 (NAE-3-SAT). Given a Boolean formula ϕ in 3-CNF, composed of a set of m clauses over the variables $\{x_1, \ldots, x_n\}$, decide whether there exists a truth assignment $\alpha \rightarrow \{\textbf{true}, \textbf{false}\}^n$ such that for no clause all three literals evaluate to the same truth value.

By the dichotomy theorem provided in [Schaefer, 1978], NAE-3-SAT is NP-complete even when all literals are positive. This special case is known as MONOTONE NAE-3-SAT.

Theorem 2.5.1. PROBLEM 2.5 *is NP-complete.*

Proof. We show that MONOTONE NAE-3-SAT $\leq_p$ PROBLEM 2.5.

Consider an instance ϕ of MONOTONE NAE-3-SAT with m clauses defined over the set of n variables. Construct the corresponding graph $\mathcal{G} = (V, E)$ as follows:

- for each variable x_i, $i = 1, \ldots, n$, add the corresponding vertex x_i;
- add two vertices s and t and connect them with all other vertices;
- for each clause c_i in ϕ, $i = 1, \ldots, m$, add the two vertices c_i and $\overline{c_i}$; then, add the edges (x, c_i) and $(x, \overline{c_i})$ if x appears in c_i.

Define the sets $S := \{s, c_1, \ldots, c_m\}$ and $T := \{t, \overline{c_1}, \ldots, \overline{c_m}\}$.

Example 2.1 Let $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_5) \wedge (x_4 \vee x_5 \vee x_6)$. Figure 2.3 shows its corresponding graph $\mathcal{G}$ and the sets S and T.

If α is a truth assignment certifying that ϕ is a YES-instance for PROBLEM 2.6, then an S,T-bond is obtained by augmenting S and T with all the vertices corresponding to the variables respectively set to **true** and to **false**. Conversely, if

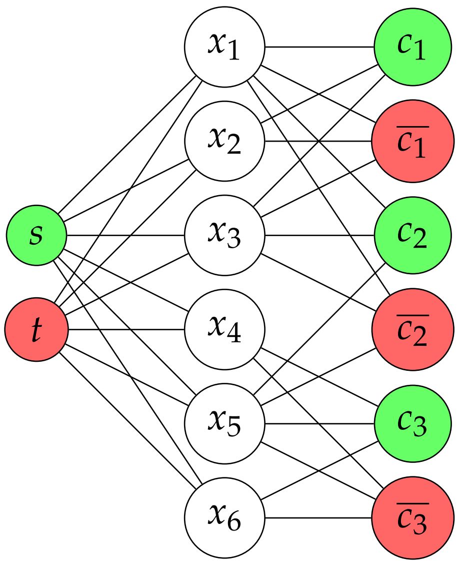


FIGURE 2.3: The graph $\mathcal{G}$ corresponding to $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_5) \wedge (x_4 \vee x_5 \vee x_6)$ in Example 2.1. The vertices in green (in red) belong to S (to T).

$\delta_{\mathcal{G}}\left(X, \overline{X}\right)$ is an S,T-bond for $\mathcal{G}$, with $S \subseteq X$ and $T \subseteq \overline{X}$, then consider the truth assignment α setting to **true** all variables $x_i \in X$ and to **false** all those in $\overline{X}$. This α certifies that ϕ is a YES-instance for PROBLEM 2.6. □

PROBLEM 2.7 (Listing all S,T-bonds of $\mathcal{G}$ with $\mathcal{G}[S]$ and $\mathcal{G}[T]$ connected). Given a connected graph $\mathcal{G} = (V, E)$ and two disjoint sets $S, T \subsetneq V$, such that both $\mathcal{G}[S]$ and $\mathcal{G}[T]$ are connected, list the S,T-bonds of $\mathcal{G}$.

PROBLEM 2.2 and PROBLEM 2.3 are still two special cases of PROBLEM 2.7, when $S = T = \emptyset$, and when $S = \{s\}$ and $T = \{t\}$, respectively. However, the associated decision problem analogous to PROBLEM 2.5 is now obviously in P. Starting with an assigned pair (S, T), and facing the decision to put an *unassigned vertex* $v \in V \setminus (S \cup T)$ either in S or in T, is exactly what suggests a binary-partition approach. However, this recursive decomposition rule cannot always be applied. Indeed, when assigning v to S, we may lose the connectedness of the vertices in T. Requiring that both $\mathcal{G}[S]$ and $\mathcal{G}[T]$ are connected will not be under our control in general. What is the nature and the complexity of the associated decision problem when we relax this double condition by assuming connectedness only for $\mathcal{G}[S]$? As noticed by [Tsukiyama et al., 1980], an S,T-bond will exist if and only if all the vertices in T end up contained within a same connected component of $\mathcal{G} \setminus S$, which can be checked in linear time (in the following this single component of $\mathcal{G}[\overline{S}]$ will be called W). Thus, we arrive at the following problem as identified by [Tsukiyama et al., 1980].

Problem 2.8 (Listing all S, T-bonds of $\mathcal{G}$ with $\mathcal{G}[S]$ and $\mathcal{G}[T]$ connected and $S, T \neq \emptyset$). Given a connected graph $\mathcal{G} = (V, E)$ and two disjoint sets $S, T \subsetneq V$, such that $\mathcal{G}[S]$ is connected and all vertices in T are contained in a same connected component (W) of $\mathcal{G}[\overline{S}]$, output all S, T-bonds of $\mathcal{G}$. We are assured this family of bonds is not empty.

As for the assumption that $S, T \neq \emptyset$, we deliberately introduced Problem 2.8 in order to avoid having to deal with spurious cases that would otherwise bother us with not much construct. Though Problem 2.2 is not any longer a special case, it can still be reduced to Problem 2.8 by means of Algorithm 2.3, that can be easily implemented as to take only an output-linear overhead.

Algorithm 2.3: Listing all bonds of a graph $\mathcal{G}$

```
0. Assert $\mathcal{G}$ is connected.
1. Let $S := \{s\}$, with $s$ an arbitrary vertex of $\mathcal{G}$, and consider any vertex
   $t \in N(S)$.
2. Solve Problem 2.8 over the pair $(S, \{t\})$.
3. $S := S \cup \{t\}$.
4. While $N(S) \neq \emptyset$:
    4.1. Choose any $t \in N(S)$.
    4.2. Solve Problem 2.8 over the pair $(S, \{t\})$.
    4.3. $S := S \cup \{t\}$.
```

Lemma 2.5.1. *Algorithm 2.3 correctly lists all bonds of $\mathcal{G}$ without duplicates.*

Proof. Each set of solutions output by Algorithm 2.3 at Steps 2 and 4.2 is a set of S, T-bonds (and thus also a set of bonds). Since there are no other steps where something is output, then every object returned by Algorithm 2.3 is an S, T-bond. At Steps 1 and 4.1, any vertex in $N(S)$ is first considered as t and then included into S. Thus, all sets of solutions output by Algorithm 2.3 are disjoint, each one differing from the others in at least one element t. Each S, T-bond of $\mathcal{G}$ belongs to exactly one set of solutions. □

By relying on any spanning tree of $\mathcal{G}$, we address Problem 2.2 by solving $O(n)$

instances of PROBLEM 2.8. Thus, in the following, we will focus on efficiently tackling PROBLEM 2.8.

2.5.2 Main idea, invariants, and base cases

Let $\mathcal{X}$ be the set of all bond-shores of $\mathcal{G}$. A bond-shore $X \in \mathcal{X}$ and the bond $\delta_{\mathcal{G}}\left(X, \overline{X}\right)$ are said *to comply* two disjoint sets $S, T \subsetneq V$ if $S \subseteq X$ and $T \subseteq \overline{X}$. Similarly to [Tsukiyama et al., 1980], we define:

$$\mathcal{X}(S,T) := \left\{X \in \mathcal{X} \mid S \subseteq X, T \subseteq \overline{X}\right\}. \tag{2.1}$$

To address PROBLEM 2.8, first we discuss an $O(m)$-delay algorithm to list all S,T-bond-shores in $\mathcal{X}$, which immediately implies an $O(m)$-delay algorithm to list their related bonds $\delta_{\mathcal{G}}\left(X, \overline{X}\right)$, that is, all S,T-bonds of $\mathcal{G}$. Then, these bounds are improved in later sections, but the high-level description of the algorithm remains the one introduced in this section.

We develop a recursion over the pair of parameters (S,T). Like [Tsukiyama et al., 1980], we propose an implicit enumeration algorithm based on a binary partition of the space of solutions for the subproblem at hand, yielding two subproblems.

Lemma 2.5.2 (Binary partition – [Tsukiyama et al., 1980]). *Given a connected graph $\mathcal{G} = (V,E)$, let $S,T \subsetneq V$ be disjoint, with $S \neq \emptyset$ and $\mathcal{G}[S]$ connected. For any vertex $v \in V \setminus (S \cup T)$, we introduce the following decomposition:*

$$\mathcal{X}(S,T) = \mathcal{X}\left(S \cup \{v\}, T\right) + \mathcal{X}\left(S, T \cup \{v\}\right), \tag{2.2}$$

where "+" denotes the disjoint union operation.

The main difference with the algorithm of [Tsukiyama et al., 1980] is in that we take into account also the biconnectivity structure of the graph at hand, in particular we are going to exploit cut-vertices. The algorithm maintains three invariants, i.e., all subproblems issued in the recursive calls will respect them:

1. the subproblem (S,T) is *fertile* in the sense that $\mathcal{X}(S,T) \neq \emptyset$:
   1a. the induced subgraph $\mathcal{G}[S]$ is connected;
   1b. $\overline{S} \neq \emptyset$ and T is contained within the same connected component W in $\mathcal{G}[\overline{S}]$.
2. $T \subseteq N(S)$.

Invariants 1 are the critical ones: we should concentrate first on these; Invariant 2 just plays useful. One first natural and convenient trick we stick to is to choose the pivot vertex v out from $(N(S) \cap W) \setminus T$. This guarantees that Invariant 1a is maintained when v is added to S. Clearly, Invariant 1a is always maintained when v is added to T and S is left unaffected. Also Invariant 1b is always maintained when S is left unaffected, given that $v \in W$.

Once Invariant 1a holds, Invariant 1b characterizes fertility: the delicate issues are how to maintain Invariant 1b when v is added to S and how to detect when $\mathcal{X}(S \cup \{v\}, T) = \emptyset$. Indeed, $\overline{S} = \emptyset$ immediately implies $\mathcal{X}(S, T) = \emptyset$. Moreover, if t_1 and t_2 were two vertices in T belonging to two different connected components of $\mathcal{G}[\overline{S}]$, then for no $X \subseteq V \setminus T$ with $S \subseteq X$ it might hold that $\mathcal{G}[\overline{X}]$ is connected, being $\mathcal{G}[\overline{X}]$ a subgraph of $\mathcal{G}[\overline{S}]$ with $t_1, t_2 \in \overline{X}$. Conversely, both $\mathcal{G}$ and $\mathcal{G}[S]$ are connected by assumption and by Invariant 1a, respectively. Then, the bond-shore $X := \overline{W}$ or its corresponding bond is the *incumbent solution* (i.e., the current best solution found), since $\mathcal{G}[\overline{X}] = \mathcal{G}[W]$ and $\mathcal{G}[X]$ are connected. Figure 2.4 below shows an example.

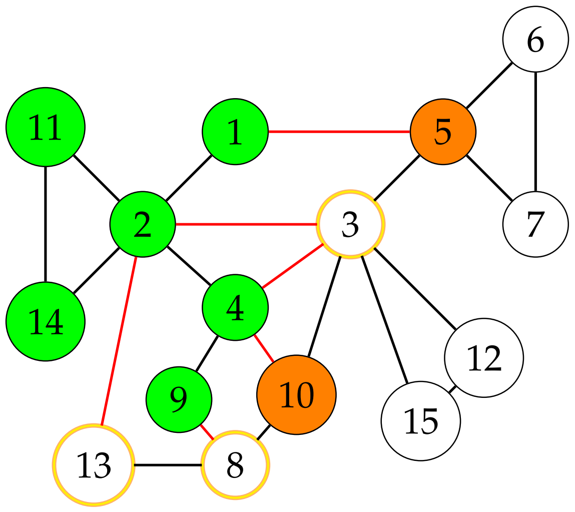


FIGURE 2.4: Example of a subproblem (S, T), where $S = \{1, 2, 4, 9, 11\}$ (filled-green vertices), $T = \{5, 10\}$ (filled-orange vertices), $W = \{3, 5, 6, 7, 8, 10, 12, 13\}$, and $(N(S) \cap W) \setminus T = \{3, 8, 13\}$ (orange circles). The bond-shore S, or its corresponding bond $\delta_{\mathcal{G}}(S, W)$ (red edges), is the incumbent solution.

Lemma 2.5.3 (Easy fertility)**.** *If v is not a cut-vertex of $\mathcal{G}[W]$, then $\mathcal{X}(S \cup \{v\}, T) \neq \emptyset$ and Invariant 1b holds for subproblem $(S \cup \{v\}, T)$.*

Lemma 2.5.4 (Fertility on cut-vertices)**.** *Assume v is a cut-vertex. Let $\tau(\mathcal{G}[W])$ be any spanning tree of $\mathcal{G}[W]$. Invariant 1b holds for subproblem $(S \cup \{v\}, T)$ and $\mathcal{X}(S \cup \{v\}, T) \neq \emptyset$ if and only if for no two vertices $t_1, t_2 \in T$, v is on the unique path between t_1 and t_2 in $\tau(\mathcal{G}[W])$.*

Proof. Invariant 1b holds for subproblem $(S \cup \{v\}, T)$ if and only if, for no two vertices $t_1, t_2 \in T$, v belongs to every path between t_1 and t_2 in $\mathcal{G}[W]$. This last condition is equivalent to the one formulated in the statement of the lemma. □

When v is a cut-vertex of $\mathcal{G}[\overline{S}]$ separating any pair of vertices in T, we say that v is *critical* (*noncritical* otherwise).

Figure 2.5 shows the three possible cases to choose the pivot vertex v among $(N(S) \cap W) \setminus T$ in order to put it into S. When v is not a cut-vertex of $\mathcal{G}[\overline{S}]$ (Figure 2.5a), we can add it to S without any worries. Instead, if v is a critical cut-vertex of $\mathcal{G}[\overline{S}]$ (Figure 2.5b), then it cannot be put into S, otherwise Invariant 1b would not hold by Lemma 2.5.4, thus the subproblem $(S \cup \{v\}, T)$ would not be fertile. Finally, when v is a noncritical cut-vertex of $\mathcal{G}[\overline{S}]$ and is added to S (Figure 2.5c), the only way to generate a fertile subproblem is to put in S also all other components of $\mathcal{G}[\overline{S}]$ connected to v and not containing any vertex in T.

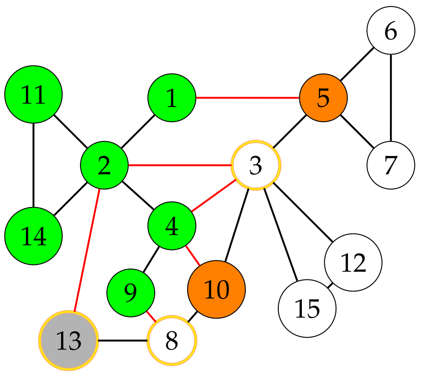
The base cases of the recursion (i.e., the leaves of the recursion tree) are the following, different from what in [Tsukiyama et al., 1980] as a consequence of the fact that our branching differs:

1. $|W| = 1$;
2. $(N(S) \cap W) \subseteq T$.

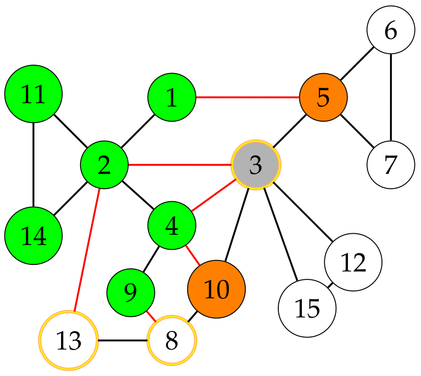
In both cases, the incumbent solution $\overline{W}$, with its corresponding bond $\delta_{\mathcal{G}}\left(\overline{W}, W\right)$, is the only element in $\mathcal{X}(S, T)$: we output it and return to the calling subproblem.

2.5.3 Algorithm

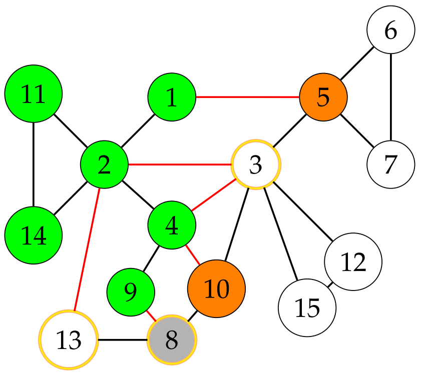
Based on all previous lemmas, we can define Algorithm 2.4, a recursive procedure based on binary partition. In the very first call, we are given as input the two nonempty disjoint sets $S, T \subsetneq V$ and the connected component W containing all vertices in T.



(a) $v = 13$ is not a cut-vertex of $\mathcal{G}[\overline{S}]$: W is updated by removing $\{13\}$; $\mathcal{X}(S \cup \{v\}, T) \neq \emptyset$.



(b) $v = 3$ is a critical cut-vertex of $\mathcal{G}[\overline{S}]$: the vertices in T would not be in the same connected component anymore; $\mathcal{X}(S \cup \{v\}, T) = \emptyset$.



(c) $v = 8$ is a noncritical cut-vertex of $\mathcal{G}[\overline{S}]$: W is updated by removing $\{8, 13\}$; $\mathcal{X}(S \cup \{v\}, T) = \mathcal{X}(\overline{W}, W) \neq \emptyset$.

FIGURE 2.5: Possible choices of the pivot vertex v (filled-grey vertex) in the example subproblem (S, T) shown in Figure 2.4.

Algorithm 2.4: Listing S, T-bonds(S, T, W)

```
1. If $|W| = 1$ or $(N(S) \cap W) \subseteq T$, output the $S, T$-bond-shore $\overline{W}$ and the
   $S, T$-bond $\delta_{\mathcal{G}}(\overline{W}, W)$.
2. Choose any $v \in (N(S) \cap W) \setminus T$.
3. Call the subproblem $(S, T \cup \{v\}, W)$.
4. If $v$ is a noncritical cut-vertex of $\mathcal{G}[W]$, then let $W'$ be the connected
   component of $\mathcal{G} \setminus (S \cup \{v\})$ that contains $T$ and call the subproblem
   $\left(\overline{W'}, T, W'\right)$.
5. Else if $v$ is not a cut-vertex of $\mathcal{G}[\overline{S}]$, call the subproblem $(S \cup \{v\}, T, W \setminus \{v\})$.
```

Theorem 2.5.2. *Algorithm 2.4 correctly lists all S, T-bonds of $\mathcal{G}$ without duplicates.*

Proof. In Step 1, the algorithm outputs a new S, T-bond-shore $\overline{W}$ and/or the corresponding S, T-bond $\delta_{\mathcal{G}}(\overline{W}, W) = \delta_{\mathcal{G}}(S, W) = \delta_{\mathcal{G}}(S, T)$, where $\mathcal{G}[\overline{W}]$ is connected by Invariant 1 and $G[W]$ is connected since either $|W| = 1$ or $W = T$. Thus, $\delta_{\mathcal{G}}(\overline{W}, W)$ is a bond by Lemma 2.3.1. Since there are no other steps where something is output, then every object returned by Algorithm 2.4 is an S, T-bond. By Lemma 2.5.2, no S, T-bond is output more than once, being this branching a special case of [Tsukiyama et al., 1980] decomposition (Equation 2.2). By the same lemma, every bond is output, because we omit to explore the second term of the decomposition only when the vertices in T would get irreversibly separated by placing v in S, which assures this second term is actually empty. □

Theorem 2.5.3. *Algorithm 2.4 takes $O(m)$ time to output a new S, T-bond-shore (or a new S, T-bond) of $\mathcal{G}$.*

Proof. We analyse the cost of each type (leaf, unary, and binary) of nodes in the recursion tree of Algorithm 2.4 separately.

Leaf nodes A leaf node corresponds to either Base case 1 or Base case 2, both handled at Step 1. In the former, $\delta_{\mathcal{G}}(\overline{W}, W)$ comprises the edges incident with the only vertex in W, thus it is output in $O\left(|\delta_{\mathcal{G}}(\overline{W}, W)|\right)$. In the latter, $\delta_{\mathcal{G}}(\overline{W}, W) = \delta_{\mathcal{G}}(S, W) = \delta_{\mathcal{G}}(S, T)$ is output in $O(|\delta_{\mathcal{G}}(S, T)|)$ time. In both cases, S is output in $O(n)$.

Unary nodes These nodes appear in the recursion tree when the pivot vertex v cannot be put into S, since v is a critical cut-vertex of $\mathcal{G}[\overline{S}]$. Thus, Algorithm 2.4

performs only Steps 2–3. Choosing any $v \in (N(S) \cap W) \setminus T$ at Step 2 costs $O(1)$. To state that v is a cut-vertex of $\mathcal{G}[W]$, we use the algorithm in [Tarjan, 1971], that takes $O(n+m)$. Then, we compute a spanning tree τ of $\mathcal{G}[W]$ in $O(n+m)$ and we check Lemma 2.5.4 in $O(n)$. Step 3 generates a subproblem by adding v to T without changing the incumbent solution. We can attribute the cost of the incumbent solution to the current node in the recursion tree, which pays $O(1) + O(n+m) + O(n+m) + O(n) = O(m)$.

Binary nodes In case of binary nodes, the first child is obtained by generating the same subproblem as in a unary node (Steps 2–3). The second child depends on the nature of the pivot vertex v.

- If v is a noncritical cut-vertex of $\mathcal{G}[\overline{S}]$ (Step 4), finding the connected component W' costs $O(n+m)$. The incumbent solution $\delta_{\mathcal{G}}(S, W)$ is updated with $\delta_{\mathcal{G}}\left(\overline{W}', W'\right)$ in $O(m)$ (i.e., by removing all edges between $\overline{W'} \setminus S$ and S and by adding all those between $\overline{W'}$ and W).
- Otherwise (Step 5), computing $W \setminus \{v\}$ costs $O(1)$. The incumbent solution is updated in $O(n)$ (i.e., by removing all edges between v and S and by adding all those between v and $W \setminus \{v\}$).

As in the previous case, we can attribute both the costs of the incumbent solution and of the second child to the current node in the recursion tree, which pays $O(m) + \max\{O(n+m) + O(m), O(1) + O(n)\} = O(m)$.

□

To achieve these bounds, we can use the following data structures:

- a spanning tree τ^W of $\mathcal{G}[W]$ (rooted at a vertex in T);
- the set V^W_{cut}, which offers the cut-vertices of $\mathcal{G}[W]$;
- the set D^W, which offers those vertices in τ^W with a strict descendant in T.

When choosing an unassigned vertex v, we place in D^W all its ancestors in τ^W found by going up in the tree until the first vertex already in D^W. We can always generate at least one subproblem by putting v in T. If $v \in \left(V^W_{cut} \setminus D^W\right)$ (i.e., v is a noncritical cut-vertex of W) or $v \notin V^W_{cut}$, then we can also put it into S and generate a second subproblem. However, in this case the graph $\mathcal{G}[W]$ changes and we have to update the incumbent solution $\delta_{\mathcal{G}}(S, W)$ and to recompute τ^W, V^W_{cut} and D^W. This is the main bottleneck of Algorithm 2.4. In the following sections, we see how to avoid this recomputation by efficiently keeping these structures updated.

2.6 The dynamic data structures employed

After a brief introduction on dynamic graph problems, this section offers the data structures needed to improve the bounds of Algorithm 2.4.

2.6.1 Dynamic graph problems

A *dynamic graph problem* asks to answer queries about a graph subject to updating operations (e.g., insertion and/or deletion of edges). The goal is to deal as efficiently as possible with an arbitrary and unpredictable sequence of updates and queries over the graph. When the only operation allowed to update a graph is the insertion (resp., deletion) of edges, then we speak of *incremental* (resp. *decremental*) dynamic graph problems. If both updating operations are allowed, the graph problem is called *fully-dynamic*. In order to completely specify the problem as an abstract data-structure interface, we must also indicate the kind of queries supported, together with the cost of each kind of operation, both queries and updates. We consider *deterministic dynamic data structures*: starting from a set of input objects and an empty structure, a sequence of updates communicated by the user is performed, dynamically changing the data structure. Each time the sequence is performed, the same resulting structure will be produced. This is not guaranteed when using *probabilistic* data structures, where the sequence of operations depends on random bits.

To improve the bounds of Algorithm 2.4, we are going to introduce three dynamic data structures. The first two, defined by [Holm et al., 2001], maintain connectivity and biconnectivity, respectively. In later sections, we are going to exploit their results as black-boxes to avoid recomputing the tree τ^W and the set of cut-vertices V_{cut}^W when an unassigned vertex v is added to S. The third data structure operates over a tree where some vertices are *lit up* (i.e., highlighted). Given a vertex of the tree, the data structure checks whether it is on the path between any pair of lit-up vertices.

For more details about dynamic graph algorithms, the reader is referred to [Cormen et al., 2009] (in particular, Section 10.4, Chapters 12–14, and also Chapter 17 for the basics of amortized analysis).

2.6.2 Maintaining connectivity

Consider the graph $\mathcal{H} = (V, E)$, with $|V| = n$ and $|E| = m$. We speak of a *fully-dynamic graph connectivity* problem when queries concern the connectedness of $\mathcal{H}$. In particular, to maintain a maximal forest $\mathcal{F}$ of $\mathcal{H}$, [Holm et al., 2001] built a deterministic data structure which offered an $O(\log^2 n)$ amortized update time and an $O(\log n / \log \log n)$ query time. [Thorup, 2000] provided a data structure with a randomized expected amortized update time of $O(\log n (\log \log n)^3)$ and a query time of $O(\log n / \log \log \log n)$. This is very close to the cell-probe lower bound of $\Omega(\log n / \log \log n)$ in [Henzinger and Fredman, 1998] and [Miltersen et al., 1994]. In 2013, starting from [Thorup, 2000], [Wulff-Nilsen, 2013] improved [Holm et al., 2001], by obtaining an $O\left(\log^2 n / \log \log n\right)$ update time in the deterministic case. For a description of the data structure, I refer the reader to [Holm et al., 2001].

[Holm et al., 2001] employed *Euler Tour (ET) trees* introduced by [Henzinger and King, 1999]. They offered the following operations:

update in $O\left(\log^2 n\right)$ amortized time:

- Insert(u, v): inserts the edge (u, v) in $\mathcal{H}$;
- Delete(u, v): deletes the edge (u, v) from $\mathcal{H}$.

query in $O(\log n / \log \log n)$ amortized time:

- Connected(u, v): tells whether the two vertices u and v are connected in $\mathcal{H}$.

The occupied memory varies with the changing of E, but it is always $O(m + n \log n)$.

Lemma 2.6.1 (Maintaining a connected component given T)**.** *Assume given a connected graph $\mathcal{H} = (V, E)$, with $n := |V|$ and $m := |E|$, and a maximal forest $\mathcal{F}$ of $\mathcal{G}$. Let $T \subsetneq V$ be a set of vertices contained in the same connected component W of $\mathcal{H}$. Let $v \in W \setminus T$ be not a critical cut-vertex of $\mathcal{H}[W]$. If v is removed from W, then we can find the new connected component W' containing T with at most $n - 1$ calls to the query* Connected(u, v).

Proof. If v is not a cut-vertex of $\mathcal{H}[W]$, then we can find $W' := W \setminus \{v\}$ in $O(1)$. Otherwise, if v is a noncritical cut-vertex of $\mathcal{H}[W]$, then to find W' we remove from W all those vertices that are in a different component of $\mathcal{H}[W] \setminus \{v\}$ than T. We choose any $u \in T$ and we call at most $n - 1$ times the query Connected(u, v), with $v \in W \setminus T$. Thus, we can find W' in $(n-1) \cdot \widetilde{O}(\log n / \log \log n) = \widetilde{O}(n)$. $\square$

2.6.3 Maintaining biconnectivity

In the *fully-dynamic graph biconnectivity* problem, the edge updates may be interspersed with queries asking whether two given vertices are biconnected (i.e., they are in the same biconnected component). In this case, [Holm et al., 2001] used *Top Trees* from [Alstrup et al., 1997]. Top trees can be implemented both with the topology trees defined in [Frederickson, 1997] and with the s, t-trees in [Sleator and Tarjan, 1983], as shown in [Alstrup et al., 2005].

[Holm et al., 2001] remarked that biconnectivity is not a transitive relation over vertices but over edges, i.e., if the edges e and f are in the same biconnected component, and the same holds for f and g as well, then also e and g belong to the same biconnected component.

Lemma 2.6.2 (Biconnectivity – [Holm et al., 2001])**.** *Consider a graph $\mathcal{H} = (V, E)$.*

(i) Biconnectivity is a transitive relation over the neighbours of a vertex $v \in V$ and if two neighbours of v are in the same biconnected component, then also v is in the biconnected component containing them.

(ii) A vertex $v \in V$ is a cut-vertex if and only if there exists a pair of its neighbours that are not in the same biconnected component.

Proof. To prove (i), consider two biconnected neighbours x and y of the vertex v. Since they are in the same biconnected component, either there exists the edge $xy \in E$ or there are two distinct paths from x to y with no common vertex other than x and y. Now take a path not containing v and add the edges xv and vy to form a cycle. This cycle certifies that all three vertices v, x and y are in the same biconnected component. Moreover, the two edges xv and vy both enjoy the property of having the endpoints belonging to a same biconnected component, from which also transitivity follows.

In (ii), if v is not a cut-vertex, then by definition its removal would not disconnect the graph and any two of its neighbours x and y would still be connected. Thus, there would exist two distinct paths connecting x and y. Conversely, if v is a cut-vertex, then two of its neighbours would belong to different components of $\mathcal{H} \setminus v$, hence to different biconnected components of $\mathcal{H}$. □

The algorithm proposed by [Holm et al., 2001] maintains a maximal spanning forest in a top tree data structure, which offers the following operations:

update in $O\left(\log^5 n\right)$ amortized time:

- INSERT(u,v): inserts the edge (u,v) in $\mathcal{H}$;
- DELETE(u,v): deletes the edge (u,v) from $\mathcal{H}$.

query in $O(\log^5 n)$ amortized time:

- BICONNECTED(u,v): tells whether the two vertices u and v are biconnected in $\mathcal{H}$.

The space used is $O(m + n\log^2 n)$.

In the following lemma, we provide the details on how to use what in [Holm et al., 2001] in order to maintain an updated knowledge of the cut-vertices.

Lemma 2.6.3 (Spanning tree and biconnectivity)**.** *Assume given a connected graph $\mathcal{H} = (V, E)$, with $n := |V|$ and $m := |E|$, and a spanning tree τ of $\mathcal{H}$. Then, we can identify all the cut-vertices of $\mathcal{H}$ with at most n calls to the query* BICONNECTED(u,v).

Proof. By Lemma 2.3.2, we can avoid checking the leaves of τ. Consider any internal vertex v of τ, let $N(v)$ be the set of its neighbours and fix $x \in N(v)$. For each $y \in N(x) \setminus x$, use the query BICONNECTED(x,y) by paying $\widetilde{O}(\log^5 n)$. When x and y are not in the same biconnected component, by Lemma 2.6.2 v is a cut-vertex of $\mathcal{H}$, and we move to test another internal vertex v'. Thus, for each vertex v, we check at most $|N(v)| - 1$ pairs. Since the sum of the degrees of the vertices in a tree is $2m = 2(n-1)$, we can discover all cut-vertices of $\mathcal{H}$ in $2(n-1) \cdot \widetilde{O}\left(\log^5 n\right) = \widetilde{O}(n)$. □

2.6.4 Checking cut-vertices

Given a connected graph $\mathcal{H} = (V, E)$, let L be the set of *lit-up* vertices of V, i.e., those vertices satisfying a certain property. We assume that L can only increase and it is initialized when a vertex $r \in V$ satisfies the required property. No vertex gets ever removed from L. We define a dynamic data structure called *lit-up tree* to efficiently keep L updated and to answer queries about the vertices, by operating over a spanning tree τ^L of $\mathcal{H}$. The data structure offers the following three operations:

setup in $O(n)$:

- INITIALIZE(τ^L): takes in charge the tree τ^L with n vertices and rooted in r, and sets $L := \{r\}$.

update in $O(1)$ amortized time:

- INSERT(v): inserts v into L.

query in $O(1)$ time:

- INTERMEDIATE(v): tells whether there exist two vertices $a, b \in L$ such that v is an internal vertex of the unique path between a and b in the tree τ^L.

The lit-up tree requires $O(n)$ space.

The setup operation is called only once, as the very first operation on the data structure. About the update operation, when a vertex v is inserted into L, actually a sequence of INSERT operations is performed. Indeed, all the ancestors of v found by going up in the tree (until the first one already in L) are inserted into L. Since no vertex gets ever removed from L, the INSERT operation is called at most $n-1$ times. Thus, the average cost of each operation is $O(n)/n = O(1)$. Then, the query INTERMEDIATE(v) corresponds to just checking whether $v \in L$ or not.

Lemma 2.6.4. *Assume given a connected graph $\mathcal{H} = (V, E)$, with $n := |V|$ and $m := |E|$ and a subset $T \subset V$. Let τ^L be a lit-up tree of $\mathcal{H}$, rooted at a chosen $t \in T$, with $L = T$. Let $V_{cut} \subset V$ be the nonempty set of cut-vertices of $\mathcal{H}$. Then, we can check whether $v \in V_{cut}$ separates any pair of vertices in T in $O(1)$.*

Proof. It directly follows from the query INTERMEDIATE(v). □

2.6.5 Summary

Table 2.1 summarizes the operations allowed by each dynamic data structure described in the previous subsections.

Goal	Data structure	Update time	Query time	Space
Maximal forest to maintain connectivity	ET-tree [Holm et al., 2001]	INSERT(u,v) DELETE(u,v) in $O(\log^2 n)^*$	CONNECTED(u,v) in $O(\log n / \log\log n)^*$	$O(m + n\log n)$
Spanning tree to maintain biconnectivity	Top tree [Holm et al., 2001]	INSERT(u,v) DELETE(u,v) in $O(\log^5 n)^*$	BICONNECTED(u,v) in $O\left(\log^5 n\right)^*$	$O\left(m + n\log^2 n\right)$
Spanning tree to check cut-vertices	Lit-up tree [Raffaele et al., 2021]	INSERT(v) in $O(1)^*$	INTERMEDIATE(v) in $O(1)$	$O(n)$

* amortized time.

TABLE 2.1: Dynamic data structures exploited.

2.7 An $\widetilde{O}(n)$–delay algorithm

Here we see how to improve Algorithm 2.4 in order to obtain the following deterministic performance guarantees:

- listing each S, T-bond-shore in $\tilde{O}(n)$-delay time;
- when contextually listing also each S, T-bond $\delta_{\mathcal{G}}\left(S, \overline{S}\right)$ as an edge-set, the delay time to output it becomes $\tilde{O}(n) + O\left(\left|\delta_{\mathcal{G}}\left(S, \overline{S}\right)\right|\right)$.

We stick to the high-level description of Algorithm 2.4 based on Invariants 1 and 2, and Lemmas 2.5.2, 2.5.3, and 2.5.4. What differ is that we exploit the dynamic data structures introduced in Section 2.6 to maintain the connected component W of $\mathcal{G}[\overline{S}]$, its spanning tree τ^W, its set of cut-vertices V^W_{cut} and the incumbent solution $\delta_{\mathcal{G}}(S, W)$.

We use a maximal forest $\mathcal{F}$ of $\mathcal{G}[\overline{S}]$ to maintain connectivity of τ^W. To efficiently update W and V^W_{cut}, we rely on the following two lemmas.

Lemma 2.7.1 (Updating W)**.** *Given a connected graph $\mathcal{G} = (V, E)$, with $n := |V|$ and $m := |E|$, let $\mathcal{F}$ be a maximal forest of $\mathcal{G}[\overline{S}]$. Let $S, T \subsetneq V$ be disjoint, with $S \neq \emptyset$, $\mathcal{G}[S]$ connected, and the vertices of T contained in the same connected component W of $\mathcal{G}[\overline{S}]$. Assume a vertex $v \in (N(S) \cap W) \setminus T$ is assigned to S. Then, we can update W in $\widetilde{O}(n)$.*

Proof. It directly follows from Lemma 2.6.1. □

Lemma 2.7.2 (Updating V^W_{cut})**.** *Given a connected graph $\mathcal{G} = (V, E)$, with $n := |V|$ and $m := |E|$, let $S, T \subsetneq V$ be disjoint, with $S \neq \emptyset$, $\mathcal{G}[S]$ connected, and the vertices of T contained in the same connected component W of $\mathcal{G}[\overline{S}]$. Let τ^W be a spanning tree of $\mathcal{G}[W]$. Then, we can identify all the cut-vertices of $\mathcal{G}[W]$ in $\widetilde{O}(n)$.*

Proof. It directly follows from Lemma 2.6.3. □

To distinguish whether a cut-vertex in V^W_{cut} is critical or not, we use a lit-up tree.

Lemma 2.7.3 (Checking critical cut-vertices)**.** *Given a connected graph $\mathcal{G} = (V, E)$, with $n := |V|$ and $m := |E|$, let $S, T \subsetneq V$ be disjoint, with $S \neq \emptyset$, $\mathcal{G}[S]$ connected, and the vertices of T contained in the same connected component W of $\mathcal{G}[\overline{S}]$. Let τ^L be a lit-up tree of $\mathcal{G}[W]$ with $L = T$ and let V^W_{cut} be the set of cut-vertices of $\mathcal{G}[W]$. Then, we can check whether $v \in V^W_{cut}$ separates any pair of vertices in T in $O(1)$.*

Proof. It directly follows from Lemma 2.6.4. □

Now we analyse the cost to compute and output a new S,T-bond-shore (or a new S,T-bond) by running Algorithm 2.4 implementing these data structures. For the amortized analysis, we use the accounting method (see [Cormen et al., 2009], Chapter 17.2). At the very first call of Algorithm 2.4, we deposit a credit for each edge in the incumbent solution $\delta_{\mathcal{G}}(S,W)$. These will be needed during the execution of the algorithm to keep $\delta_{\mathcal{G}}(S,W)$ updated within the desired bounds. We notice that the delay to output both the very first bond-shore S and its related bond $\delta_{\mathcal{G}}\left(S,\overline{S}\right)$ will be $\Theta(m)$, as needed to take in the input.

Theorem 2.7.1 (Improving Algorithm 2.4)**.** *Assume given a connected graph $\mathcal{G} = (V,E)$, with $n := |V|$ and $m := |E|$, let $S,T \subsetneq V$ be disjoint, with $S \neq \emptyset$, $\mathcal{G}[S]$ connected, and the vertices of T contained in the same connected component W of $\mathcal{G}[\overline{S}]$. Then, by using a maximal forest $\mathcal{F}$ of $\mathcal{G}[\overline{S}]$, a spanning tree τ^W and a lit-up tree τ^L of $\mathcal{G}[W]$, Algorithm 2.4 takes $\widetilde{O}(n)$ time to output a new bond-shore S and $\widetilde{O}(n) + |\delta_{\mathcal{G}}(S,\overline{S})|$ to output a new S,T-bond $\delta_{\mathcal{G}}\left(S,\overline{S}\right)$ of $\mathcal{G}$.*

Proof. We analyse the cost of each type (leaf, unary, and binary) of nodes in the recursion tree of Algorithm 2.4 separately.

Leaf nodes A leaf node corresponds to either Base case 1 or Base case 2, both handled at Step 1. In the former, $\delta_{\mathcal{G}}(\overline{I},I)$ comprises the edges incident with the only vertex in I, thus it is output in $O(|\delta_{\mathcal{G}}(\overline{I},I)|)$. In the latter, $\delta_{\mathcal{G}}(\overline{I},I) = \delta_{\mathcal{G}}(S,I) = \delta_{\mathcal{G}}(S,T)$ is output in $O(|\delta_{\mathcal{G}}(S,T)|)$ time. The bond-shore S is output in $O(n)$.

Unary nodes In this case, Algorithm 2.4 performs only Steps 2–3. Step 2 costs $O(1)$. In Step 3, before generating a subproblem by adding v to T, the only operation to perform is to check, when $v \in V^W_{cut}$, whether v is a critical cut-vertex. By Lemma 2.7.3, this is done in $O(1)$. W is unchanged, as well as the incumbent solution.

Binary nodes In case of binary nodes, the first child is obtained by generating the same subproblem as in a unary node (Steps 2–3) and the second child depends on the nature of the pivot vertex v. We attribute both the costs of the incumbent solution and of the second child to the current node in the recursion tree.

- When v is a noncritical cut-vertex of $\mathcal{G}[W]$ (Step 4), updating W takes $\widetilde{O}(n)$ by Lemma 2.7.1. Also τ^W is updated by removing v and all its edges from, relying on the primitive Delete(u,v). This operation costs at most $n \cdot O(\log^5 n)$ ([Holm et al., 2001]). Updating V_{cut}^{W} costs $\widetilde{O}(n)$ by Lemma 2.7.2. The incumbent solution $\delta_{\mathcal{G}}(S, W)$ is updated by removing all edges between v and S and adding all those between v and the updated W in $O(n)$. To also remove all edges between $\overline{W} \setminus S$ and S, we use the corresponding credits anticipated in the first call of the algorithm.

- When v is not a cut-vertex of $\mathcal{G}[W]$ (Step 5), removing v from W costs $O(1)$ by Lemma 2.7.1. As in the previous case, updating τ^{W} costs at most $n \cdot O(\log^5 n)$ ([Holm et al., 2001]) and updating $V_{cut}^{W'}$ costs $\widetilde{O}(n)$ by Lemma 2.7.2. The incumbent solution changes by removing all edges between v and S and adding all those between v and the updated W, which is done in $O(n)$.

Thus, the current node in the recursion tree pays $\widetilde{O}(n)$.

□

2.8 Further developments

As for future work concerning bonds, we can wonder whether it is possible to avoid paying the $\widetilde{O}(n)$ part when listing bonds as edge-sets, thus paying only $|\delta_{\mathcal{G}}(S, \overline{S})|$. Also, it would be interesting to investigate the possibility of developing an algorithm output-linear in the number of vertices and not exploiting dynamic data structures. Moreover, one could be curious to discover whether there exists an enumeration algorithm for listing bonds that does not rely on minimal s, t-cuts.

Another small improvement may be about not choosing any vertex from $N(S) \cap W \setminus T$ for branching, but instead identifying the most convenient one, in order to make the decomposition faster. One could be interested in focusing on particular classes of graphs (e.g., planar graphs, k-degenerate graphs).

Furthermore, Algorithm 2.4 could be parallelized in case of binary nodes, being the two generated subproblems independent from each other. Indeed, a processor managing a current binary node could pass the first subproblem (generated in Step 3) to another available processor, together with a copy of all dynamic data structures at that point. Then, it could handle by itself the second subproblem (generated either in Step 4 or in Step 5). The only interaction among processors would be just an output collection process, which would concatenate the output into a single stream.

More generally, one could focus on other enumeration problems for which it is possible to develop and utilize dynamic data structures specifically designed for listing. This is a research direction not so explored so far, since there are just a few examples of listing problems tackled by using dynamic graph algorithms (e.g., [Eppstein et al., 2010], [Eppstein and Strash, 2011]).

Chapter 3

A new decomposition for the Monotone Boolean Duality problem

"The fact is, my writing has always found itself facing two divergent paths that correspond to two different types of knowledge. One path goes into the mental space of bodiless rationality, where one may trace lines that converge, projections, abstract forms, vectors of force. The other path goes through a space crammed with objects and attempts to create a verbal equivalent of that space by filling the page with words, involving a most careful, painstaking effort to adapt what is written to what is not written, to the sum of what is sayable and not sayable. [...] I think we are always searching for something hidden or merely potential or hypothetical, following its traces whenever they appear on the surface."

Italo Calvino, *Exactitude*, in *Six Memos for the Next Millennium*

This chapter is dedicated to some of the most interesting and still open problems in enumeration and combinatorics, that are the *Monotone Boolean Duality* problem and the *Monotone Boolean Dualization* problem.

I warn the reader that some notations of this chapter overlap with those used in Chapter 2. For instance, the letters $\mathcal{G}$ and $\mathcal{F}$, adopted to respectively indicate a graph and a forest of it, now represent two set families. I decided not to change the letters in either the two chapters because in graph theory $\mathcal{G}$ is conventionally used to indicate a graph, and the two symbols $\mathcal{F}$ and $\mathcal{G}$ have been used in many previous works on the Monotone Boolean Duality and Dualization problems.

3.1 Introduction

Any family $\mathcal{F}$ of incomparable subsets of $V := \{1, 2, \dots, n\}$ essentially encodes a positive DNF (i.e., disjunctive normal form) of a Boolean function $f : \{0,1\}^V \to \{0,1\}$:

$$f(x) = \bigvee_{F \in \mathcal{F}} \bigwedge_{i \in F} x_i =: \mathrm{DNF}(\mathcal{F}) . \tag{3.1}$$

Clearly, f is monotone given that $\mathrm{DNF}(\mathcal{F})$ involves no negative literals.

Conversely, any monotone Boolean function f can be written in this way where $\mathcal{F}$ comprises those minimal subsets of V whose characteristic vectors evaluate to true under f; these are also called *the prime implicants* of f. Thus, $\mathrm{DNF}(\mathcal{F})$ is the *unique irredundant DNF* of the monotone Boolean function f.

Dually, any monotone Boolean function $g : \{0,1\}^V \to \{0,1\}$ can be described instead by its *unique irredundant CNF* (i.e., conjunctive normal form) in which no negation occurs:

$$g(x) = \bigwedge_{G \in \mathcal{G}} \bigvee_{i \in G} x_i =: \mathrm{CNF}(\mathcal{G}) , \tag{3.2}$$

where $\mathcal{G}$ comprises the *prime implicates* of g. These are the complementary sets of the maximal subsets of V whose characteristic vectors evaluate to false under g. Indeed, a point x evaluates to true if and only if it misses every possible occasion to evaluate to false, that is, if and only if it falls out from every set in $\overline{\mathcal{G}} := \{\overline{G} : G \in \mathcal{G}\}$.

The following two problems arise naturally.

Problem 3.1 (Monotone Boolean Duality using Boolean functions). Given a pair (f, g) of monotone Boolean functions, where f and g are expressed through a DNF and a CNF, respectively, decide the equivalence of f and g.

Problem 3.2 (Monotone Boolean Dualization using Boolean functions). Given a monotone Boolean function f, expressed through a DNF, compute its equivalent monotone Boolean function g expressed through a CNF.

Given a subset of V, we say that it is a *transversal* of $\mathcal{G}$ when it shares at least one element with each set of $\mathcal{G}$. Problem 3.1 consists in checking whether $f(x) = g(x)$, $\forall\ x \in \{0,1\}^V$. This is the case if and only if, considering the corresponding pair of set families $(\mathcal{F}, \mathcal{G})$, $\mathcal{F}$ comprises precisely the *minimal* transversals of $\mathcal{G}$. Indeed, every point $x \in \{0,1\}^V$ containing an $F \in \mathcal{F}$ (hence $f(x) = 1$) intersects every $G \in \mathcal{G}$ (the contrary would imply $g(x) = 0$), and every point x strictly contained in some $F \in \mathcal{F}$ (hence $f(x) = 0$, since x contains no one of the incomparable sets in $\mathcal{F}$) misses some $G \in \mathcal{G}$ (the contrary would imply $g(x) = 1$). Equivalently, it holds that $\mathcal{G}$ comprises the minimal transversals of $\mathcal{F}$. Indeed, given any $x \in \{0,1\}^V$ intersecting all $F \in \mathcal{F}$, its complement $\overline{x}$ contains no F (hence $f(\overline{x}) = 0$) and there exists some $G \in \mathcal{G}$ disjoint from $\overline{x}$, that is, contained in x. Thus, the dual of $\mathcal{G}$ is $\mathcal{F}$ and the relationship between $\mathcal{F}$ and $\mathcal{G}$ is symmetric.

All authors in the related literature (see Section 3.3) have always preferred to work with the set families $\mathcal{F}$ and $\mathcal{G}$, rather than with the monotone Boolean functions, for symmetry reasons. This is the approach also used to tackle Problem 3.2. Indeed, one could try to apply elementary Boolean laws to compute the CNF function corresponding to a given DNF, but generally this would not be efficient ([Eiter et al., 2008]). Thus, we slightly reformulate Problem 3.1 and Problem 3.2 as follows.

Problem 3.3 (Monotone Boolean Duality using set families). Given a pair $(\mathcal{F}, \mathcal{G})$ of set families, both defined over a vertex set $V := \{1, 2, \ldots, n\}$, decide whether $\mathcal{F}$ and $\mathcal{G}$ are dual of each other (i.e., whether $\mathcal{F}$ comprises the minimal transversals of $\mathcal{G}$ and viceversa).

Problem 3.4 (Monotone Boolean Dualization using set families). Given a set family $\mathcal{F}$, defined over a vertex set $V := \{1, 2, \ldots, n\}$, compute its dual set family $\mathcal{G}$ (i.e., $\mathcal{G}$ comprises the minimal transversals of $\mathcal{F}$).

Problem 3.4, also known as the *Hypergraph Transversal Generation* problem, can be reduced to Problem 3.3. Indeed, given a set family $\mathcal{F}$, corresponding to a DNF function f, we start by considering an empty $\mathcal{G}$. Given this input pair $(\mathcal{F}, \mathcal{G})$, we solve Problem 3.3. If the answer is positive, we stop. Otherwise, we can find a transversal in $\mathcal{G}$ that is not minimal or a minimal transversal of $\mathcal{F}$ not present in $\mathcal{G}$. This is added to $\mathcal{G}$ and the whole procedure is repeated, until the duality testing is satisfied (i.e., until $\mathcal{G}$ is composed of all minimal transversals of $\mathcal{F}$). Therefore, checking that the two set families $\mathcal{F}$ and $\mathcal{G}$ are dual (i.e., solving Problem 3.3) cannot be harder than computing the dual of $\mathcal{F}$ (i.e., solving Problem 3.4).

Currently, given $\sigma := |\mathcal{F}| + |\mathcal{G}|$, the best known algorithms to solve Problem 3.3 run in quasi-polynomial $\sigma^{o(\log \sigma)}$ time, or use $O\left(\log^2 \sigma\right)$ nondeterministic bits ([Fredman and Khachiyan, 1996], [Eiter et al., 2003], [Kavvadias and Stavropoulos, 2003]). This is strong evidence that the problem is not coNP-complete, but no polynomial time algorithm has been discovered yet. Despite this compelling positive result, and even if polynomial-time algorithms for many special classes are known, the exact complexity of the general problem is still an open question.

The main purpose of this chapter is to present another possible decomposition for Problem 3.3 based on both the remarkable result by [Fredman and Khachiyan, 1996] and the concept of full covers introduced by [Boros and Makino, 2009]. The approach I define provides a strong bound for Problem 3.3 which, however, in the worst case is the same as the one of [Fredman and Khachiyan, 1996]. Anyway, the problem is described under another light, with more commitment to the symmetry between $\mathcal{F}$ and $\mathcal{G}$. I hope this might foster simplifications and further comprehension. I also express an analogy with classical implicit enumeration approaches, like those widely used in operations research or in constraint programming.

This chapter is organized as follows. In Section 3.2, I present some applications of the two problems, with a particular focus on a fundamental question in linear programming. In Section 3.3, I review the most relevant works on the two problems. In Section 3.4, I provide a few preliminary definitions and lemmas on set families and clutters, as well as some operations and decompositions. In Section 3.5, I introduce the main ideas at the base of my approach. To define it, I need the concept of full covers, which I recall in Section 3.6. Then, in Section 3.7, I describe the new decomposition to solve Problem 3.3 and analyse its time complexity. Successively, I shift my attention to Problem 3.4. In Section 3.8, after discussing a few delicate issues regarding memory, I show how to exploit the decomposition proposed to solve Problem 3.4 by using only polynomial space. Finally, Section 3.9 illustrates some further developments and future work.

3.2 Applications

Problem 3.3 and Problem 3.4 have relevance in many research areas, such as theoretical computer science, artificial intelligence, combinatorial optimization, and mathematical programming. Hereafter I propose a few examples of applications studied in the literature.

[Eiter and Gottlob, 1991] and [Eiter and Gottlob, 2000] provided a rich list of related problems in clause satisfiability, Boolean switching theory, model-based diagnosis, design of relational databases, and updates in distributed databases.

Regarding data mining and knowledge discovery, [Gunopulos et al., 1997] refers to problems such as computing maximal frequent and minimal infrequent sets, finding episodes from sequences, and finding keys or inclusion dependencies from relation instances.

More recently, in the context of query-oriented extractive summarization, [Van Lierde and Chow, 2019] made a connection between the problem of sentence retrieval and the extraction of a transversal in a hypergraph.

In terms of data profiling, [Bläsius et al., 2022] solved the discovery problem of minimal unique column combinations of several real-world and artificially generated databases by providing an enumeration algorithm to list all minimal bounded-size transversals of an hypergraph, also improving the result of [Eiter and Gottlob, 1995].

In matroid theory, given a matroid defined on a ground set V and two nonempty disjoint sets $A, B \subset V$, consider the problem to enumerate all maximal subsets $X \subseteq A$ such that the $Span(X) \cap B \neq \emptyset$, where $Span(X)$ is the set of all subsets Y of V such that the rank of $(X \cup Y)$ is equal to the rank of X itself. [Khachiyan et al., 2005] underlined that, in case of binary matroids, this includes as special case Problem 3.4. Indeed, it corresponds to listing all maximal independent sets for a given hypergraph defined over 2^V, where A and B are the sets of the characteristic vectors of all vertices and edges, respectively.

3.2.1 The Vertex (Facet) Enumeration problem

Problem 3.3 and Problem 3.4 can also find application in linear programming (e.g., [Boros et al., 2002], [Boros et al., 2007]). Consider the following theorem.

Theorem 3.2.1 (Minkowski-Weyl's theorem [Ziegler, 2012]). *Any convex polyhedron P can be described through two different equivalent formulations:*

- ***$\mathcal{H}$-representation**, as the intersection of closed halfspaces:*

$$P = P(A, b) = \{Ax \leq b \mid x \in \mathbb{R}^n\},$$

where $A \in \mathbb{R}^{m \times n}$ is an $m \times n$ real matrix and $b \in \mathbb{R}^m$ is an m-dimensional real vector;

- ***$\mathcal{V}$-representation**, as the Minkowski sum of the convex hull of a finite set of points plus a conical combination of vectors $\mathbb{R}^n$:*

$$P = conv\{v_1, \dots, v_r\} + cone\{d_1, \dots, d_s\},$$

where $\mathcal{V}(P) = \{v_1, \dots, v_r\} \subseteq \mathbb{R}^n$ is the set of vertices or extreme points of P and $\{d_1, \dots, d_s\} \subseteq \mathbb{R}^n$ is the set of extreme directions of P.

Theorem 3.2.1 naturally leads to the fundamental question: *is there a way to pass from one representation to the other?* The two following enumeration problems arise.

Problem 3.5 (Vertex enumeration). Given a $\mathcal{H}$*-representation* of a polyhedron, compute all of its vertices and extreme directions, to obtain its $\mathcal{V}$*-representation*.

Problem 3.6 (Facet enumeration). Given a $\mathcal{V}$*-representation* of a polyhedron, compute all of its halfspaces, to obtain its $\mathcal{H}$*-representation*.

[Lovász, 1992] denoted Problem 3.5 and Problem 3.6 by the name of "*Polytope-Polyhedron problem*", and underlined the structural similarity between these and Problem 3.3 and Problem 3.4. To better see how these are connected, let $A \in \{0,1\}^{m\times n}$ and let $\underline{\mathbf{1}}$ and $\underline{\mathbf{0}}$ be two m-dimensional vectors of all ones and all zeros, respectively. Then $P(A,\underline{\mathbf{1}}) := \{\mathbb{R}^n \mid x \in Ax \geq \underline{\mathbf{1}}, x \geq \underline{\mathbf{0}}\}$ is the set-covering polyhedron with only integral vertices, having A as ideal matrix (see, e.g., [Boros et al., 2009]). These vertices are in bijection with the minimal transversals of the hypergraph $\mathcal{H}$ associated to the matrix A. Indeed, the columns and the rows of A correspond to the vertices and the characteristic vectors of the hyperedges of $\mathcal{H}$, respectively. Thus, the problem of enumerating the vertices of $P(A,\underline{\mathbf{1}})$ is equivalent to the problem of computing the minimal transversals of $\mathcal{H}$, that is Problem 3.4. Then, an efficient procedure to solve Problem 3.4 would lead to an efficient procedure to also solve this special restriction of Problem 3.5 (and vice versa), but may not hold for the general case. Anyway, given the celebrated result by [Fredman and Khachiyan, 1996], the vertices of $P(A,\underline{\mathbf{1}})$ can be enumerated in quasi-polynomial time, which makes Problem 3.5 unlikely to be NP-Hard.

In the literature, the main approaches to solve Problem 3.5 and Problem 3.6 are pivoting algorithms (e.g., reverse search by [Avis and Fukuda, 1992] – see Subsection 1.4.3) or incremental algorithms (e.g., the double-description method by [Motzkin et al., 1953], later described by [Fukuda and Prodon, 1995]). In case of 0/1-polytopes, [Bussieck and Lübbecke, 1998] showed that Problem 3.5 is strongly P-enumerable. [Khachiyan et al., 2006] proved that generating the vertices of a polyhedron is NP-Hard. Anyway, the problem of generating the vertices and extreme rays of a polyhedron and the problem of generating the vertices of a polytope (i.e., a bounded polyhedron) are still open.

We can also consider tackling the Problem 3.5 and Problem 3.6 jointly. Let $M := \{1,\ldots,m\}$. Given the $\mathcal{H}$-representation $\{a_i^T x \leq b_i, \mid i \in M,\ a_i, x \in \mathbb{R}^n\}$ of a polytope P and a subset $S \subseteq M$, let $P(S) := \{a_i^T x \leq b_i \mid i \in M \setminus S,\ a_i, x \in \mathbb{R}^n\} \cap \{a_i^T x = b_i \mid i \in S,\ a_i, x \in \mathbb{R}^n\}$. Let $\mathcal{W}$ be the set family comprising the subsets of 2^M such that $P(S)$ is nonempty. Then, there exists a 1-1 correspondence between the vertices $\mathcal{V}(P)$ and the maximal subsets of $\mathcal{W}$ ([Boros et al., 2007]. Similarly, there exists a 1-1 correspondence between the facets of P and the maximal subsets X of $\mathcal{V}(P)$, for which the following system, in the variables $(a, b) \in \mathbb{R}^n \times \mathbb{R}$, is feasible:

$$\begin{cases} a^T v \leq b, & \forall v \in \mathcal{V}(P), \\ a^T v = b, & \forall v \in X. \end{cases} \tag{3.3}$$

Also, we can define the set family $\mathcal{B}$, consisting of the subsets of $2^M \setminus \mathcal{W}$ such that $P(S)$ is empty. The set of the maximal subsets of $\mathcal{W}$ and the one of the minimal subsets of $\mathcal{B}$ are dual of each other, and quasi-polynomial time algorithms are known to produce both, jointly ([Boros et al., 2004]). Indeed, we can develop a polynomial satisfiability oracle that, given a subset S of 2^M, classifies it by putting into $\mathcal{W}$, if the corresponding polytope $P(S)$ is nonempty, or into $\mathcal{B}$, otherwise. This oracle could be then used in a recursive algorithm that, receiving as input the two set families $\mathcal{W}$ and $\mathcal{B}$, either confirms their duality or returns an unclassified point.

By considering a generic monotone property, we can define the following problem.

Problem 3.7 (Joint Generation – [Khachiyan et al., 2006]). Given a monotone property, represented by a satisfiability oracle, and two set families $\mathcal{W}$ and $\mathcal{B}$ of subsets of $M := \{1, 2, \ldots, m\}$, either find an unclassified subset $S \in 2^M$ that belongs to $\mathcal{W}$ or $\mathcal{B}$, or prove that $\mathcal{W}$ and $\mathcal{B}$ are complete (and dual of each other).

Studying Problem 3.5, Problem 3.6, and Problem 3.7 may be relevant to provide an alternative way to the mainstream operations-research approach to combinatorial optimization, that is, formulating integer linear programming (ILP) models that are managed by an industrial solver (e.g., CPLEX by [IBM, nd] and [Gurobi Optimization, nd]). Solving such an ILP is known to be NP-Hard in general, therefore formulating the problem in this way might impede solvability. Usually, the approach is to relax the integrality constraints and solve an LP relaxation, which is a polynomial problem. However, this introduces fractional vertices. The truth is that, for all NP-Hard optimization problems, there will not be any compact description of the facets (unless NP = coNP [Papadimitriou and Yannakakis, 1984]). For a given family of combinatorial problems (and for their associated polytopes), polyhedral combinatorics struggles in the task to understand their minimal description, being already happy to identify some important families of defining inequalities.

3.3 Related work

This section briefly presents the most relevant works in the literature about the definition, the study, and the resolution of Problem 3.3 and Problem 3.4.

[Bioch and Ibaraki, 1995] showed that Problem 3.3 and Problem 3.4 are polynomially equivalent, in the sense that an incremental-polynomial algorithm for the former exists if and only if the latter is in P. With $V = \{a_1, ..., a_n, b_1, ..., b_n\}$, the dual of $\mathcal{F} = \{\{a_i, b_i\} : i = 1, ..., n\}$ is $\mathcal{G} = \bigotimes_{i=1}^{n}\{a_i, b_i\}$, where $\bigotimes$ indicates the Cartesian product. There exists hence an infinite sequence of set families $\mathcal{F}_k, k \in \mathbf{N}$, such that $|\mathcal{G}_k|$ is exponential in $|\mathcal{F}_k|$. With this, the algorithm for the Problem 3.4 would be polynomial time in the input and output sizes. The existence of such an algorithm is a challenging open problem.

[Fredman and Khachiyan, 1996] obtained a remarkable result by using intersection properties of clutters (i.e., set families of pairwise incomparable sets – see Subsections 3.4.2–3.4.4) and the concept of *frequency* of an element of the vertex set (see Subsection 3.4.5). These allowed them to put the decision problem in an intermediate class between P and coNP, presenting two quasi-polynomial-time algorithms. The former, known as Algorithm FK-A, solves Problem 3.3 in $\sigma^{O\left(\log^2 \sigma\right)}$, whereas the latter, known as Algorithm FK-B, solves it in $\sigma^{o(\log \sigma)}$ time (in our notation, $\sigma = |\mathcal{F}| + |\mathcal{G}|$ is the input size).

[Tamaki, 2000] remastered the decompositions by [Fredman and Khachiyan, 1996] to obtain an algorithm streaming out the dual form of the input formula f, which requires only polynomial internal space. His algorithm carefully manages lexicographic orderings in order to avoid multiple outputs of the same clause.

[Kavvadias and Stavropoulos, 2003] modified [Fredman and Khachiyan, 1996]'s algorithms to solve the problem in deterministic polynomial time plus $O\left(\log^2 n\right)$ non-deterministic guesses, thus placing the problem in the coNP-$\left[\log^2 \sigma\right]$ class, according to our notation.

[Eiter et al., 2003] investigated some subclasses of the decision problem and provided new polynomial-time cases (e.g., degenerate CNFs, read-k CNFs, acyclic CNFs). Independently from [Kavvadias and Stavropoulos, 2003], they also used the results of [Fredman and Khachiyan, 1996] to show that the duality of a pair of monotone Boolean functions can be checked in polynomial time with limited nondeterminism.

[Gaur and Krishnamurti, 2004] studied the problem of determining the self-duality of a DNF, a special case equivalent to the general Problem 3.3, since $h =$

$xf \vee yg \vee xy$ is self-dual if and only if f and g are dual ([Eiter and Gottlob, 2000]). They developed an algorithm that, according to our notation, generally runs in $O\left(\sigma^{2\log\sigma+2}\right)$ time and in average polynomial time on random instances.

[Khachiyan et al., 2006] moved from Boolean lattices to integrality boxes. For monotone properties over the integral vectors in a rectangular box, they described an implementation of FK-A to jointly generate the families $\mathcal{F}$ and $\overline{\mathcal{G}}$ of minimal satisfying and maximal non-satisfying integral vectors respectively.

[Elbassioni, 2008] generalized [Fredman and Khachiyan, 1996] by considering the following question: *what can we do when more than one variable is frequent?* He proposed two efficient parallel versions of [Fredman and Khachiyan, 1996]'s algorithms, which run in polylogarithmic time on a quasi-polynomial number of processors (in a PRAM model). These two algorithms also offer stronger bounds on the sequential complexity of the problem in the asymmetric case (i.e., when the sizes of the two families $\mathcal{F}$ and $\mathcal{G}$ significantly differ). Moreover, they allow to generate all minimal transversals of a given hypergraph using only input polynomial space.

[Boros and Makino, 2009] developed an approach based on full covers (see Section 3.6) that shows better parallel-time complexity than the algorithms proposed by [Elbassioni, 2008], requiring a smaller number of processors for polylogarithmic-time parallel computation. Furthermore, they also improved the best known bound on the sequential time complexity in the asymmetric case.

[Hagen et al., 2009] provided a detailed description of FK-A and FK-B, also discussing possible variants to be considered in the implementation. They also reported some experimental results on test instances already used in [Khachiyan et al., 2006]. Before their work, FK-B had never been tested in practice: they showed that it is competitive on almost all classes of instances; thus, this version should be included in further experimental studies.

One of the most recent works has been done by [Sedaghat et al., 2018], where they speeded up dualization in FK-B, modifying the algorithm in order to produce multiple certificates of non-duality. Moreover, they showed how to reduce the number of redundant tests performed at the beginning of FK-B (i.e., removing any supersets found comparing every pair of implicates in the input CNF and every pair of implicants in the input DNF). Furthermore, they also implemented a memoization technique to avoid solving the same subproblem more than once.

The interested reader is also referred to [Hagen, 2008] for more details.

3.4 Preliminaries

From now on, I will focus on Problem 3.3. The goal is to propose a new decomposition to solve it. To start, in this section I introduce the notation used in the whole chapter, and I recall the needed background on set families and clutters.

3.4.1 Set families

A family $\mathcal{E}$ of pairwise different sets is called a *set family over* $V(\mathcal{E}) := \bigcup_{E \in \mathcal{E}} E$. Where $V := V(\mathcal{E})$, the pair $(V, \mathcal{E})$ is an *hypergraph*: V is its *vertex set* and the set family $\mathcal{E}$ is its *edge set*.

Definition 3.4.1 (Complement). Given a subset $S \subseteq V$, we denote by $\overline{S} := V \setminus S$ its *complementary set*. Moreover, for a set family $\mathcal{E}$, the family of *complementary sets* is given by $\overline{\mathcal{E}} := \{\overline{E} : E \in \mathcal{E}\}$.

Definition 3.4.2 (Downward closed set). A set family $\mathcal{E}$ is *downward-closed* if the subsets of any set in $\mathcal{E}$ are also in $\mathcal{E}$. We denote by $\mathcal{E}^- := \{S \subseteq V : \exists\, E \in \mathcal{E} \text{ s.t. } S \subseteq E\}$ the *downward closure* of $\mathcal{E}$.

Definition 3.4.3 (Upward closed set). A set family $\mathcal{E}$ is *upward-closed* if $\overline{\mathcal{E}}$ is *downward-closed*. We denote by $\mathcal{E}^+ := \{S \subseteq V : \exists\, E \in \mathcal{E} \text{ s.t. } E \subseteq S\}$ the *upward closure* of $\mathcal{E}$.

Definition 3.4.4 (Minimal/Maximal). A set $E \in \mathcal{E}$ is called *minimal* (resp., *maximal*) in $\mathcal{E}$ when no proper subset (resp., superset) of E belongs to $\mathcal{E}$. The family of such extremal sets in $\mathcal{E}$ is denoted by *Minimals($\mathcal{E}$)* (resp., *Maximals($\mathcal{E}$)*).

Note that $\mathcal{E}^+$ is the largest set family defined over $V(\mathcal{E})$ having the same minimal sets as $\mathcal{E}$. Meanwhile, *Minimals($\mathcal{E}$)* is the smallest such set family. The dual considerations hold for *Maximals($\mathcal{E}$)*.

Example 3.1 A truth assignment t over a set of Boolean variables $x_1, x_2, \ldots, x_n$ can be encoded as a subset of $V := \{1, 2, \ldots, n\}$, namely the set $\{i \in V : t(x_i) = 1\}$. Any Boolean function over $x_1, x_2, \ldots, x_n$ is hence encoded as a set family over V; just list the sets encoding its satisfying truth assignments. Denote by $\mathcal{W}$ this set family, and let $\mathcal{B} := \overline{\mathcal{W}}$. A point $x \in \{0,1\}^V$ (and the corresponding subset of V) is called *white* if it satisfies the function f, *black* otherwise. When the function is monotone, then $\mathcal{W}$ is upward-closed, hence univocally captured by the *white border* $\mathcal{F} := \mathit{Minimals}(\mathcal{W})$. This comprises of the minimal implicants of the function and offers its unique irredundant DNF $\bigvee_{F\in\mathcal{F}} \bigwedge_{i\in F} x_i$. At the same time, $\mathcal{B}$ is downward-closed, hence univocally captured by the *black border* $\overline{\mathcal{G}} := Maximals\,(\mathcal{B})$. Its complementary set $\mathcal{G}$, which clearly conveys the same information, offers the unique irredundant CNF form $\bigwedge_{G\in\mathcal{G}} \bigvee_{i\in G} x_i$. In this way, the family $\mathcal{F}$ of minimal white points and the family $\mathcal{G}$ comprising the complements of the maximal black points offer two alternative representations of the very same function.

For instance, let $V := \{1,2,3,4\}$, $f := (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_3 \wedge x_4)$ and $g := (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_3)$. Then:

- $\mathcal{W} := \{\{1,2\},\{1,3\},\{3,4\},\{1,2,3\},\{1,2,4\}\}$;
- $\mathcal{B} := \{\{1\},\{2\},\{3\},\{4\},\{1,4\},\{2,3\},\{2,4\}\}$.

It follows that:

- $\mathcal{F} := \{\{1,2\},\{1,3\},\{3,4\}\}$;
- $\overline{\mathcal{G}} := \{\{2,4\},\{2,3\},\{1,4\}\}$;
- $\mathcal{G} := \{\{1,3\},\{1,4\},\{2,3\}\}$.

3.4.2 Clutters, blockers, and duality

Definition 3.4.5 (Clutter)**.** Two sets are *incomparable* if neither of the two contains the other. A *clutter* is a set family $\mathcal{E}$ of pairwise incomparable sets.

The family of all minimal (maximal) sets of any set family $\mathcal{E}$ is always a clutter.

Example 3.1 (cont.) This definition holds, for example, for the family $\mathcal{F}$ of the minimal white points that satisfy a monotone Boolean function, and for the family $\overline{\mathcal{G}}$ of the maximal black points on which such a function is false. Given a pair of clutters $\mathcal{F}$ and $\mathcal{G}$, when wondering whether $f \equiv g$, we can readily find out in poly-time if there exists a point x such that $f(x) = 1$ and $g(x) = 0$. For each pair $(F, G) \in \mathcal{F} \times \mathcal{G}$, we check that $F \cap G \neq \emptyset$. Indeed, if $F \cap G = \emptyset$, then $f(F) = 1$ and, by monotonicity, $g(F) \leq g(\overline{G}) = 0$. Conversely, if $f(x) = 1$ and $g(x) = 0$, then $x \supseteq F \in \mathcal{F}$ and $x \subseteq \overline{G} \in \overline{\mathcal{G}}$, that is, $\overline{x} \supseteq G \in \mathcal{G}$, thus $F \cap G = \emptyset$. If every pair (F, G) passes the test, then no such x exists. It is hence standard practice, when dealing with these issues, to assume to work with pairs fulfilling this first necessary condition.

Definition 3.4.6. A pair $(\mathcal{E}, \mathcal{D})$ is *standard* if and only if for no $(E, D) \in \mathcal{E} \times \mathcal{D}$ it holds that $E \cap D = \emptyset$.

Example 3.2 Given $\mathcal{F} := \{\{1,2\},\{1,3\},\{3,4\}\}$ and $\mathcal{G} := \{\{1,4\},\{2,3\}\}$, the pair $(\mathcal{F},\mathcal{G})$ is standard.

Definition 3.4.7 (Transversal)**.** Two sets *intersect* when they have elements in common. A set $T \subseteq V(\mathcal{E})$ is called a *transversal* of a set family $\mathcal{E}$ if it intersects every set of $\mathcal{E}$. A transversal T is called *a minimal transversal* when no proper subset of T is a transversal.

Definition 3.4.8 (Blocker)**.** Given a clutter $\mathcal{E}$, the family of all minimal transversals of $\mathcal{E}$ is always a clutter, called the *blocker of* $\mathcal{E}$ and denoted by $\mathcal{E}^*$.

Definition 3.4.9 (Duality – [Berge, 1989])**.** Given a clutter $\mathcal{E}$ and its blocker $\mathcal{E}^*$, $\mathcal{E}$ and $\mathcal{E}^*$ are called *dual*.

The relationship between a clutter $\mathcal{E}$ and its blocker $\mathcal{E}^*$ is symmetric, as unveiled by the following lemma.

Lemma 3.4.1 ([Edmonds and Fulkerson, 1970], [Lehman, 1964], [Lehman, 1979], [Seymour, 1976], and [Berge, 1989])**.** *If $\mathcal{E}$ is a clutter, then $(\mathcal{E}^*)^* = \mathcal{E}$.*

Proof. Every member E of $\mathcal{E}$ intersects every member of $\mathcal{E}^*$, thus E contains some members of $(\mathcal{E}^*)^*$. Since $\mathcal{E}$ is a clutter, we can now close this proof by showing that each member of $(\mathcal{E}^*)^*$ contains some members of $\mathcal{E}$. Indeed, each $E \in (\mathcal{E}^*)^*$ intersects every member of $\mathcal{E}^*$ and so $\overline{E}$ contains no member of $\mathcal{E}^*$. Thus, $\overline{E}$ is not a transversal of $\mathcal{E}$ and, as such, it does not intersect some members $\widetilde{E}$ of $\mathcal{E}$. Namely, $\widetilde{E} \subseteq E$ for some $\widetilde{E} \in \mathcal{E}$. □

Example 3.1 (cont.) The DNF function f in Eq. 3.1 and the CNF function g in Eq. 3.2 are equivalent if and only if $\mathcal{F}$ and $\mathcal{G}$ are dual. The fact that every $G \in \mathcal{G}$ should be a transversal for $\mathcal{F}$ or that every $F \in \mathcal{F}$ should be a transversal for $\mathcal{G}$ is a must in order for the pair to be standard. Informally, the reason why we want $\mathcal{G}$ ($\mathcal{F}$) to contain all minimal transversals of $\mathcal{F}$ ($\mathcal{G}$) is because we want each single point in $\{0,1\}^V$ to be classified either white or black.

3.4.3 Clean pairs and a first naive approach

As observed in Subsection 3.4.2, we can check in polynomial time that the pair $(\mathcal{E},\mathcal{D})$ is standard, i.e., that every member in $\mathcal{D}$ is a transversal for $\mathcal{E}$ (and every member of $\mathcal{E}$ is a transversal for $\mathcal{D}$, this condition is actually symmetric since two universal quantifiers commute). Actually, we can also check in polynomial time that all these transversals are minimal (just try to remove any possible element). The open problem is how to make sure we are not omitting some minimal transversals.

Definition 3.4.10. A pair of clutters $(\mathcal{E}, \mathcal{D})$ is *clean* if every $D \in \mathcal{D}$ is a minimal transversal of $\mathcal{E}$ and every $E \in \mathcal{E}$ is a minimal transversal of $\mathcal{D}$.

Example 3.2 (cont.) Given $\mathcal{F} := \{\{1,2\},\{1,3\},\{3,4\}\}$ and $\mathcal{G} := \{\{1,4\},\{2,3\}\}$, the pair $(\mathcal{F}, \mathcal{G})$ is clean.

When $(\mathcal{E}, \mathcal{D})$ is clean, then $\mathcal{D} \subseteq \mathcal{E}^*$ and $\mathcal{E} \subseteq \mathcal{D}^*$, but the converse does not hold. It is in this space to the converse that the complexity of the problem is open. On the contrary, the two necessary conditions can both be checked in polynomial time, and any violation to any of them can be promptly turned into a missing transversal (since, informally, it offers a still unclassified point).

Given a clutter $\mathcal{E}$, a simple way to compute its minimal transversals (i.e., to solve PROBLEM 3.4) is provided by the following lemma.

Lemma 3.4.2 (Naive approach). *Let $\mathcal{E} := \mathcal{E}' \cup \{E\}$ be a clutter, with $E \subseteq V(\mathcal{E})$. Then:*

$$\mathcal{E}^* := \mathit{Minimals}\left(\left\{D \cup \{v\} \mid D \in (\mathcal{E}')^*, v \in E\right\}\right). \tag{3.4}$$

Proof. Consider $D \in (\mathcal{E}')^*$. If $D \cap E \neq \emptyset$, then $D \in \mathcal{E}^*$ and it is minimal. Otherwise, D misses every point in E and a minimal transversal of E can be obtained by adding $v \in E$ to D. Conversely, we know that, for every $D \in \mathcal{E}^*$, $D \cap E \neq \emptyset$. Also, when $|D \cap E| \geq 2$, if we remove a single element of $D \cap E$ from D, D would still intersect E in at least another element. Thus, in this case, $D \in (\mathcal{E}')^*$. □

We could recursively rely on Lemma 3.4.2 to decompose the problem, one set $E \in \mathcal{E}$ at a time. The cardinality of $\{D \cup \{v\} \mid D \in (\mathcal{E}'), v \in E^*\}$ is bounded by $|V(\mathcal{E})| \cdot |(\mathcal{E}')^*|$. However, the issue of this first simple procedure is that the minimal transversals of $\mathcal{E}'$ can be more than the ones of $\mathcal{E}$ (i.e., $|(\mathcal{E}')^*| \geq |(\mathcal{E})^*|$). Thus, it would not be guaranteed that generated subproblems have a smaller size. It may seem counter-intuitive, but we provide the following example to solve any doubts.

Example 3.3 Let $V := \{v_1, \ldots, v_n, v_1', \ldots, v_n'\}$, thus $|V| = 2n$.
Let $\mathcal{F}_1 := \{\{v_i, v_i'\} \mid i = 1, \ldots, n\}$ and $\mathcal{F}_2 := \left\{\left\{v_i, v_j'\right\} \mid i, j = 1, \ldots, n\right\}$, where $V(\mathcal{F}_1) = V(\mathcal{F}_2) = V$ (see Figure 3.1). Then, $|\mathcal{F}_1^*| = 2^n$ (for each edge $\{v_i, v_i'\}$, we choose either v_i or v_i') and $|\mathcal{F}_2^*| = \left|\{\{v_1, \ldots, v_n\}, \{v_1', \ldots, v_n'\}\}\right| = 2$.

Now, note that, if we remove one edge at a time from $\mathcal{F}_2$, then we would eventually obtain $\mathcal{F}_2' = \mathcal{F}_1$. The number of the minimal transversals of $\mathcal{F}_2'$ would be 2^n, whereas $\mathcal{F}_2$ would have only 2 minimal transversals.

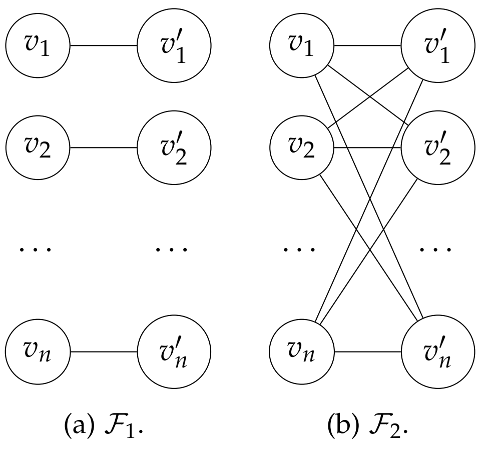


FIGURE 3.1: The clutters $\mathcal{F}_1 := \{\{v_i, v_i'\} \mid i = 1, \dots, n\}$ and $\mathcal{F}_2 := \{\{v_i, v_j'\} \mid i, j = 1, \dots, n\}$ in Example 3.3.

Thus, we need to investigate other ways to more efficiently decompose the original problem. To do so, we first recall some operations on clutters.

3.4.4 Filter and Projection

Deletion and *Contraction* are two basic operations on clutters that have been extensively studied in the literature. Instead of them, in this section I prefer to recall two operations introduced by [Elbassioni, 2008], that turn out to be handy to deal with this problem. I call them *Filter* and *Projection*, and I show the link they have with Deletion and Contraction, respectively.

We consider a clutter $\mathcal{E}$ defined over a vertex set $V(\mathcal{E})$.

Definition 3.4.11 (Filter). *Filtering a clutter $\mathcal{E}$ through a set $S \subseteq V(\mathcal{E})$* yields the clutter $\mathcal{E}_S := \{E \in \mathcal{E} \mid E \subseteq S\}$. That is, only those sets of $\mathcal{E}$ contained in S are considered.

Example 3.1 (cont.) Back to the context of representing monotone Boolean functions, the Filter operation addresses the question: *what would become the white border once all variables outside S were frozen to* 0*?*

Definition 3.4.12 (Projection). *Projecting a clutter $\mathcal{E}$ over a set $S \subseteq V(\mathcal{E})$* yields the set family $\mathcal{E}^S$ comprising of the minimal sets in $\{E \cap S \mid E \in \mathcal{E}\}$.

Example 3.1 (cont.) The question addressed by Projection is: *what would become the maximal black points in case all variables outside S were set to* 1*?*

Example 3.2 (cont.) Given $\mathcal{F} := \{\{1,2\},\{1,3\},\{3,4\}\}$, let $S := \{2,3,4\}$. Then:

- $\mathcal{F}_S := \{F \in \mathcal{F} \mid F \subseteq S\} = \{\{3,4\}\}$;
- $\mathcal{F}^S := Minimals\{F \cap S \mid F \in \mathcal{F}\} = Minimals\{\{2\},\{3\},\{3,4\}\} = \{\{2\},\{3\}\}$.

Remark 3.4.1. *Filter and Projection are just the complementary versions of Deletion and Contraction, respectively. Indeed, the clutter $\mathcal{E}_S$, obtained by filtering the set S on the clutter $\mathcal{E}$, corresponds to the one computed by deleting the set $\overline{S}$ from $\mathcal{E}$. Similarly, the clutter $\mathcal{E}^S$, obtained by projecting the set S on the clutter $\mathcal{E}$, corresponds to the one computed by contracting the set $\overline{S}$ in $\mathcal{E}$. The main reason why we prefer to rely on Filter and Projection is to lighten the notation and, consequently, to simplify the following lemmas.*

Both Filter and Projection can be repeatedly applied to a clutter, by considering different subsets of the vertex set $V(\mathcal{E})$, and they can also be applied in a composed way, as shown in the following lemma.

Lemma 3.4.3. *Given a clutter $\mathcal{E}$ and two disjoint sets $S_1, S_2 \subseteq V(\mathcal{E})$, then:*

(i) $\left(\mathcal{E}_{S_1}\right)_{S_2} = \mathcal{E}_{S_1 \cap S_2}$;

(ii) $\left(\mathcal{E}^{S_1}\right)^{S_2} = \mathcal{E}^{S_1 \cap S_2}$;

(iii) $\left(\mathcal{E}^{S_1}\right)_{S_2} = (\mathcal{E}_{S_2})^{S_1}$.

Moreover, [Seymour, 1976] proved how these two operations link a clutter and its blocker, as reported below.

Lemma 3.4.4 ([Seymour, 1976]). *Given a clutter $\mathcal{E}$ and a set $S \subseteq V(\mathcal{E})$, then:*

(i) $(\mathcal{E}_S)^* = (\mathcal{E}^*)^S$;

(ii) $\left(\mathcal{E}^S\right)^* = (\mathcal{E}^*)_S$.

Proof. Consider the first statement. If $\widetilde{E} \in (\mathcal{E}_S)^*$, then $\widetilde{E} \cup \overline{S}$ intersects each member of $\mathcal{E}$ and contains some member $E \in \mathcal{E}^*$; then $E \setminus \overline{S}$ (and hence $\widetilde{E}$) contains some member of $(\mathcal{E}^*)^S$. In the opposite direction, if $\widetilde{E} \in (\mathcal{E}^*)^S$ and $\widetilde{E} = E \setminus \overline{S}$, where $E \in \mathcal{E}^*$, $\widetilde{E}$ intersects each member of $\mathcal{E}_S$; thus $\widetilde{E}$ contains a member of $(\mathcal{E}_S)^*$.

The second statement is equivalent to the first one by Lemma 3.4.1. Indeed, $\left(\mathcal{E}^S\right)^* = \left(\left((\mathcal{E}^*)^*\right)^S\right)^* = \left(((\mathcal{E}^*)_S)^*\right)^* = (\mathcal{E}^*)_S$. □

Given a clutter $\mathcal{E}$ and a vertex $v \in V(\mathcal{E})$, by applying Filter and Projection, we can uniquely reconstruct $\mathcal{E}$ from $\mathcal{E}_{\overline{\{v\}}}$ and $\mathcal{E}^{\overline{\{v\}}}$.

Lemma 3.4.5 (Reconstructability)**.** *Let $\mathcal{E}$ and $\mathcal{D}$ be two clutters on the same vertex set V, and let $v \in V$. Assume $\mathcal{E}_{\overline{\{v\}}} = \mathcal{D}_{\overline{\{v\}}}$ and $\mathcal{E}^{\overline{\{v\}}} = \mathcal{D}^{\overline{\{v\}}}$. Then, $\mathcal{E} = \mathcal{D}$.*

Proof. $\mathcal{E}$ can be uniquely decomposed as follows. By definition, $\mathcal{E}_{\overline{\{v\}}} = \{E \in \mathcal{E} \mid E \not\ni v\}$. As for $\{E \in \mathcal{E}, v \in E\}$, note that $\{E \setminus \{v\} \mid E \in \mathcal{E}, v \in E\} \subseteq \mathcal{E}^{\overline{\{v\}}}$, since $\mathcal{E}$ is a clutter. Also, $\mathcal{E}^{\overline{\{v\}}} \setminus \{E \setminus \{v\} \mid E \in \mathcal{E}, v \in E\} \subseteq \mathcal{E}_{\overline{\{v\}}}$.

In explicit, $\mathcal{E} = \mathcal{E}_{\overline{\{v\}}} \cup \left\{E \cup \{v\} \mid E \in \mathcal{E}^{\overline{\{v\}}} \setminus \mathcal{E}_{\overline{\{v\}}}\right\}$. □

3.4.5 Fredman and Khachiyan's result and frequency

Given a clean pair $(\mathcal{E}, \mathcal{D})$ of clutters, we can apply Filter and Projection to check the duality between $\mathcal{E}$ and $\mathcal{D}$ as follows.

Lemma 3.4.6 (Variable-based decomposition (see, e.g., [Eiter et al., 2008] and [Hagen, 2008])**.** *Let $\mathcal{E}$ and $\mathcal{D}$ be two clutters and consider an element v of the vertex set V. Then, $\mathcal{D} = \mathcal{E}^*$ if and only if both of the following hold:*

(i) $\mathcal{D}_{\overline{\{v\}}} = \left(\mathcal{E}^{\overline{\{v\}}}\right)^*$*, and*

(ii) $\mathcal{D}^{\overline{\{v\}}} = \left(\mathcal{E}_{\overline{\{v\}}}\right)^*$.

Proof. If $\mathcal{D} = \mathcal{E}^*$, then *(i)* and *(ii)* follow by Lemma 3.4.4. Conversely, we first apply Lemma 3.4.4 to $\left(\mathcal{E}^{\overline{\{v\}}}\right)^*$ in (i) and to $\left(\mathcal{E}_{\overline{\{v\}}}\right)^*$ in (ii) to obtain, respectively:

(i) $\mathcal{D}_{\overline{\{v\}}} = \left(\mathcal{E}^{\overline{\{v\}}}\right)^* = (\mathcal{E}^*)_{\overline{\{v\}}}$, and

(ii) $\mathcal{D}^{\overline{\{v\}}} = \left(\mathcal{E}_{\overline{\{v\}}}\right)^* = (\mathcal{E}^*)^{\overline{\{v\}}}$,

which, by also means of Lemma 3.4.5, imply that $\mathcal{D} = \mathcal{E}^*$. □

This lemma is at the base of the two algorithms proposed by [Fredman and Khachiyan, 1996] to solve Problem 3.3. These algorithms are known as *FK-A* and its improved version *FK-B*, respectively. Indeed, given a clean pair $(\mathcal{E}, \mathcal{D})$ of clutters, defined over the same vertex set V, [Fredman and Khachiyan, 1996] select an element $v \in V$ and recursively call the two smaller subproblems $\left(\mathcal{E}^{\overline{\{v\}}}, \mathcal{D}_{\overline{\{v\}}}\right)$ and $\left(\mathcal{E}_{\overline{\{v\}}}, \mathcal{D}^{\overline{\{v\}}}\right)$. This approach is called *variable-based* decomposition (see, e.g., [Eiter et al., 2008] and [Hagen, 2008]).

However, this procedure is efficient only if the given v appears in a large fraction of the sets in at least one of the clutters $\mathcal{E}$ or $\mathcal{D}$. This led [Fredman and Khachiyan, 1996] to introduce the concept of *frequency*.

Definition 3.4.13 (Frequency)**.** Given a set family $\mathcal{E}$ and an element $v \in V$, we define the *frequency* of v in $\mathcal{E}$ as follows:

$$\omega_{v,\mathcal{E}} := \frac{|\{E \in \mathcal{E} \mid v \in E\}|}{|\mathcal{E}|}. \tag{3.5}$$

Example 3.2 (cont.) Given $\mathcal{F} := \{\{1,2\},\{1,3\},\{3,4\}\}$ and $\mathcal{G} := \{\{1,4\},\{2,3\}\}$, then:

- $\omega_{1,\mathcal{F}} = \omega_{3,\mathcal{F}} := \frac{2}{3}$;
- $\omega_{2,\mathcal{F}} = \omega_{4,\mathcal{F}} := \frac{1}{3}$;
- $\omega_{1,\mathcal{G}} = \omega_{2,\mathcal{G}} = \omega_{3,\mathcal{G}} = \omega_{4,\mathcal{G}} := \frac{1}{2}$.

When $\mathcal{E}$ and $\mathcal{D}$ are dual, [Fredman and Khachiyan, 1996] proved the existence of an element $v \in V$ with frequency in $\mathcal{E}$ or in $\mathcal{D}$ greater or equal than $\frac{1}{\log(|\mathcal{E}|+|\mathcal{D}|)}$. By carefully selecting such an element to apply Lemma 3.4.6, the two subproblems obtained have size at most $\left(1 - \frac{1}{\log(|\mathcal{E}|+|\mathcal{D}|)}\right) |\mathcal{E}| \cdot |\mathcal{D}|$ and $|\mathcal{E}| \cdot |\mathcal{D}| - 1$, respectively. In this way, [Fredman and Khachiyan, 1996] achieved the first quasi-polynomial bound for Problem 3.3, which still represents the state of the art.

3.5 Pursuing symmetry

In this section, I present the main idea guiding the approach I propose, inspired by the fact that two dual clutters are just two representations of one and the same object.

3.5.1 The role of certificates

The bijective relation between two dual clutters is perfectly symmetric. This symmetry is the reason why, similarly to [Berge, 1989] and [Fredman and Khachiyan, 1996], all authors have preferred to work, rather than with the DNF f and the CNF g, with their encoding families $\mathcal{F}$ and $\mathcal{G}$.

Previous algorithms were dedicated to assess duality by recursively checking the duality of a certain number of smaller instances, hinging on lemmas stating the equivalence between the duality of the original input pair and the duality of each pair of some family of subproblems (as in the variable-based decomposition in Subsection 3.4.5). Only in joint generation algorithms, relying either on this idea or on duality checking as a subroutine ([Boros et al., 2004]), attention was also paid to *certificates*, and then the algorithm was aspiring to find a missing transversal for just one of the two families, and thus trying to correct their anomalies.

The problem of deciding whether the two members $\mathcal{E}$ and $\mathcal{D}$ of a clean pair $(\mathcal{E},\mathcal{D})$ are mutually dual is in coNP, by its very definition. As a NO-certificate, one could either provide a set in $\mathcal{E}^* \setminus \mathcal{D}$ or a set in $\mathcal{D}^* \setminus \mathcal{E}$. Concentrating on just searching for a set that hits all the sets in $\mathcal{E}$ and is not contained in any set in $\mathcal{D}$ (i.e., the notion of NO-certificate usually adopted at this point in other works, as also most natural when pursuing the generation of the dual) breaks the perfect symmetry in the problem. In dislike of this double, two-ways, possibility for a certificate, one could observe by pursuing symmetry that a certificate of one form exists if and only if a certificate of the other form also exists (a set X is a transversal of $\mathcal{E}$ not containing any set in $\mathcal{D}$ if and only if $\overline{X}$ is a transversal of $\mathcal{D}$ not containing any set in $\mathcal{E}$).

3.5.2 Bipartitions as certificates

Here we pursue and seek for symmetry also in the language of certificates. For this reason, we prefer and propose to work with NO-certificates having the form of a bipartition of the vertex set $\{S,\overline{S}\}$, where S intersects every member of $\mathcal{E}$ and its complement $\overline{S}$ intersects every member of $\mathcal{D}$.

Fact 3.5.1. *Let $(\mathcal{E},\mathcal{D})$ be a clean pair of clutters. Then, $\mathcal{E}$ and $\mathcal{D}$ are not dual if and only if there exists a bipartition $\{S,\overline{S}\}$ of the vertex set such that:*

(i) S intersects every member of $\mathcal{E}$;

(ii) $\overline{S}$ intersects every member of $\mathcal{D}$.

Proof. If we have such a partition, then S is a transversal of $\mathcal{E}$ not containing any set in $\mathcal{D}$. Any minimal subset of S that is a transversal of $\mathcal{E}$ is hence a member of $\mathcal{E}^* \setminus \mathcal{D}$.

Conversely, if there exists an $S \in \mathcal{E}^* \setminus \mathcal{D}$, then S is a transversal of $\mathcal{E}$ and, moreover, $\overline{S}$ is a transversal of $\mathcal{D}$, since S is incomparable with every other set in $\mathcal{E}^*$ and $\mathcal{D} \subseteq \mathcal{E}^*$. □

Fact 3.5.1 leads to the following.

Fact 3.5.2. *Let $(\mathcal{E},\mathcal{D})$ be a clean pair of clutters and let $\{S,\overline{S}\}$ be a bipartition of their common vertex set V. Then, exactly one of the following three cases occurs:*

(i) there exists a set $E_0 \in \mathcal{E}$ such that $E_0 \subseteq \overline{S}$;

(ii) there exists a set $D_0 \in \mathcal{D}$ such that $D_0 \subseteq S$;

(iii) no set of $\mathcal{E}$ is contained in $\overline{S}$ and no set of $\mathcal{D}$ is contained in S.

Proof. The three cases are pairwise incompatible since: (i) and (ii) together imply $E_0 \cap D_0 = \emptyset$, (i) and (iii) together imply $E_0 \cap S = \emptyset$, (ii) and (iii) together imply $D_0 \cap \overline{S} = \emptyset$. When (i) does not hold, then S is a transversal for $\mathcal{E}$. When (ii) does not hold, then $\overline{S}$ is a transversal for $\mathcal{D}$. Thus, when both (i) and (ii) do not hold, then (iii) holds: S intersects every set in $\mathcal{E}$ and $\overline{S}$ intersects every set in $\mathcal{D}$. □

3.5.3 Bipartitions and frequency

We can exploit the concept of frequency, introduced in Subsection 3.4.5, to provide the following definition.

Definition 3.5.1 (Ω-bipartition)**.** Let $\mathcal{E}$ and $\mathcal{D}$ be two set families with $V(\mathcal{E}) = V(\mathcal{D}) =: V$. Let $\Omega \in [0,1]$ be a threshold value, fixed in advance. A partition $\{S, \overline{S}\}$ is called an Ω-*bipartition* if both of the following two properties hold:

(i) $\omega_{v,\mathcal{E}} < \Omega$ for every $v \in S$;

(ii) $\omega_{v,\mathcal{D}} < \Omega$ for every $v \in \overline{S}$.

Lemma 3.5.1. *Let $\mathcal{E}$ and $\mathcal{D}$ be two set families with $V(\mathcal{E}) = V(\mathcal{D}) =: V$. Let $\Omega \in [0,1]$ be a threshold value, fixed in advance. Let $S := \{v \in V \mid \omega_{v,\mathcal{E}} < \Omega\}$. Then, V admits an Ω-bipartition if and only if for every $v \in V$ either $\omega_{v,\mathcal{E}} < \Omega$ or $\omega_{v,\mathcal{D}} < \Omega$.*

Proof. Two cases are possible: either $\{S, \overline{S}\}$ is an Ω-bipartition, or there exists a $v \in \overline{S}$ such that both $\omega_{v,\mathcal{E}} \geq \Omega$ and $\omega_{v,\mathcal{D}} \geq \Omega$. In the second case, no ω-bipartition exists by definition. □

3.6 Full covers

In this section, I introduce the last concept needed to define a new procedure to solve Problem 3.3. First, I recall the notion of full covers of a clutter and some methods to construct them, as defined by [Boros and Makino, 2009] and implicitly by [Elbassioni, 2008]. Then, I show how to exploit full covers and Ω-bipartitions together, in order to decompose an instance of Problem 3.3 in a family of smaller instances.

3.6.1 Full covers and duality testing

Definition 3.6.1 (Full cover)**.** A set family $\mathcal{C}$ is a *full cover* of a set family $\mathcal{E}$ if for every $E \in \mathcal{E}$ there exists a $C \in \mathcal{C}$ such that $E \subseteq C$.

Examples of (trivial) full covers of $\mathcal{E}$ are $\{V\}$ and $\mathcal{E}$ itself.

Hereafter we consider the definition, given by [Boros and Makino, 2009], that exploits the Filter operation seen in Subsection 3.4.4.

Definition 3.6.2 ([Boros and Makino, 2009])**.** Given a clutter $\mathcal{E}$, a set family $\mathcal{C}$ is a full cover of $\mathcal{E}$ if and only if

$$\mathcal{E} = \bigcup_{C \in \mathcal{C}} \mathcal{E}_C. \tag{3.6}$$

Lemma 3.6.1 (Duality with full covers – [Boros and Makino, 2009])**.** *Let $\mathcal{E}$ and $\mathcal{E}^*$ be dual clutters, let $\mathcal{D} \subseteq \mathcal{E}^*$, and let $\mathcal{C}$ be a full cover of $\mathcal{E}^* \setminus \mathcal{D}$. Then, $\mathcal{D} = \mathcal{E}^*$ if and only if $\mathcal{D}_C = \left(\mathcal{E}^C\right)^*$, for each $C \in \mathcal{C}$.*

Proof. By Lemma 3.4.4, if $\mathcal{D} = \mathcal{E}^*$, then $\mathcal{D}_C = \left(\mathcal{E}^C\right)^*$, for every possible $C \subseteq V$.

Conversely, if $\mathcal{D}$ is missing some member $T \in \mathcal{E}^*$, then consider any $C \in \mathcal{C}$ such that $T \subseteq C$. Clearly, $T \notin \mathcal{D}_C$, whereas $\left(\mathcal{E}^C\right)^* = (\mathcal{E}^*)_C$ contains T. □

We remark that the reformulation of Lemma 3.6.1 given above offers a (very shallow) strengthening with respect to the original one by [Boros and Makino, 2009], in that $\mathcal{C}$ is not required to be a full cover of the whole $\mathcal{E}^*$. This new formulation is slightly more handy, since it does not yield dummy subproblems in the resulting decomposition algorithms and helps in simplifying some general constructions for full covers.

[Boros and Makino, 2009] showed that, conversely, the completeness of the decomposition characterizes in a sense the concept of full cover. Anyway, both in [Boros and Makino, 2009] and here, what given in Lemma 3.6.1 suffices in decomposing the problem into smaller subproblems to allow for a recursive approach. Indeed, Lemma 3.6.1 allows substituting one single instance with a family of smaller instances: $\mathcal{E}$ and $\mathcal{D}$ are dual in the original problem if and only if, for every $C \in \mathcal{C}$, $\mathcal{E}^C$ and $\mathcal{D}_C$ are dual.

[Boros and Makino, 2009] provided two methods to construct a full cover of the dual clutter $\mathcal{E}^*$, that they alternate appropriately during the execution of their algorithm, according to properties satisfied by transversals in $\mathcal{E}$ or in $\mathcal{E}^*$. The former method, which will be used later in this chapter, is described in the following lemma.

Lemma 3.6.2 (Full cover construction – [Boros and Makino, 2009])**.** *Given a clutter $\mathcal{E}$ and a set $E_0 \in \mathcal{E}$, the family*

$$\mathcal{C}(E_0) = \big\{\{i\} \cup \overline{E} \mid E \in \mathcal{E}, i \in E \cap E_0\big\} \tag{3.7}$$

forms a full cover of $\mathcal{E}^*$. By construction, the number of sets in $\mathcal{C}(E_0)$ is at most $\sum_{E\in\mathcal{E}} |E|$.

Example 3.2 (cont.) Given $\mathcal{F} := \{\{1,2\},\{1,3\},\{3,4\}\}$, let $F_0 := \{1,3\}$. Then, $\mathcal{C}(F_0) := \{\{1,2,3\},\{1,2,4\},\{1,3,4\},\{2,3,4\}\}$.

We prove only a slightly stronger version of Lemma 3.6.2, offered in the following lemma.

Lemma 3.6.3 (Stronger full cover construction)**.** *Given a set family $\mathcal{E}$, let $E_0 \in \mathcal{E}$ and fix its elements in a specific order $E_0 = \{e_1, e_2, \ldots, e_t\}$. Then, the following*

$$\mathcal{C}(E_0) := \bigcup_{i=1}^{t} \bigcup_{\{E\in\mathcal{E}|e_i\in E\}} \{\{e_i\} \cup (V \setminus \{e_1, e_2 \ldots e_{i-1}\} \setminus E)\} \tag{3.8}$$

forms a full cover of $\mathcal{E}^*$.

Proof. Let D be any set of $\mathcal{E}^*$. Then, D intersects E_0. Let i be the smallest natural such that $e_i \in D \cap E_0$; from this, $D \subseteq V \setminus \{e_1, e_2 \ldots e_{i-1}\}$. Since D is a minimal transversal of $\mathcal{E}$, then there exists a set $E \in \mathcal{E}$ such that $D \cap E = \{e_i\}$. For such E it holds that $D \setminus \{e_i\} \subseteq V \setminus \{e_1, e_2 \ldots e_{i-1}\} \setminus E$. □

To yield smaller subproblems, in the use of Lemma 3.6.3, it is only natural to sort the elements of E_0 as $e_1, e_2, \ldots, e_t$ in increasing order of their frequency in the sets of the dual $\mathcal{E}^*$.

Example 3.2 (cont.) Given $\mathcal{F} := \{\{1,2\},\{1,3\},\{3,4\}\}$, let $F_0 := \{1,3\}$. Then, $\mathcal{C}(F_0) := \{\{1,2,3\},\{1,2,4\},\{1,3,4\}\}$.

For completeness, we also report the second method described by [Boros and Makino, 2009] to construct a full cover of the dual clutter $\mathcal{E}^*$, even if we are not going to exploit it in this chapter. Actually, [Boros and Makino, 2009] attributed it to [Elbassioni, 2008], which implicitly used it in his own decomposition to solve PROBLEM 3.3.

Lemma 3.6.4 (Full cover construction – [Elbassioni, 2008], [Boros and Makino, 2009]). *Given a clutter $\mathcal{E}$ and a minimal transversal $D_0 \in \mathcal{E}^*$, the family*

$$\mathcal{C}(D_0) = \{V \setminus \{i\} \mid i \in D_0\} \cup \{D_0\} \tag{3.9}$$

forms a full cover of $\mathcal{E}^*$.

Proof. Let D be any set of $\mathcal{E}^*$ other than D_0. Then, there exists an $i \in D_0 \setminus D$ since $\mathcal{E}^*$ is a clutter. □

3.6.2 Full covers and bipartitions

In this subsection, we link the concepts of full covers and Ω-bipartitions, previously defined in Subsection 3.5.3. They are used jointly in the following lemma.

Lemma 3.6.5. *Let $\mathcal{E}$ be a clutter, let $\mathcal{D} \subseteq \mathcal{E}^*$. Let $\{S, \overline{S}\}$ be an Ω-bipartition of $V(\mathcal{E}) = V(\mathcal{D}) =: V$. If there exists $E_0 \in \mathcal{E}$ such that $E_0 \subseteq \overline{S}$, let $\mathcal{C}(E_0)$ be a full cover of $\mathcal{E}^*$. Then, $|\mathcal{D}_C| \leq \Omega \cdot |\mathcal{D}|$, for each $C \in \mathcal{C}(E_0)$.*

Proof. Consider any $E_0 \in \mathcal{E}$ with $E_0 \subseteq \overline{S}$ and any $C \in \mathcal{C}(E_0)$. Then, there exists an $E \in \mathcal{E}$ and an $i \in E_0 \cap E$ such that $C = \{i\} \cup \overline{E}$. The sets of $\mathcal{D}$ can be divided into two groups:

- *those containing i:* these are at most $\Omega \cdot |\mathcal{D}|$ sets, since $i \in E_0 \subseteq \overline{S}$;
- *those not containing i:* these must all intersect E, since $E \in \mathcal{E}$ and $\mathcal{D} \subseteq \mathcal{E}^*$. Thus, each of these sets $D \in \mathcal{D}$ not containing i should have at least one element external to $\overline{E}$ and also external to $\{i\} \cup \overline{E}$; as such, D falls out from $\mathcal{D}_C$.

In conclusion, $|\mathcal{D}_C| \leq \Omega \cdot |\mathcal{D}|$. □

Example 3.2 (cont.) Given $\mathcal{F} := \{\{1,2\}, \{1,3\}, \{3,4\}\}$ and $\mathcal{G} := \{\{1,4\}, \{2,3\}\}$, let $\Omega := \frac{2}{3}$. Then, $\{S, \overline{S}\} := \{\{2,4\}, \{1,3\}\}$ is an Ω-bipartition of $V(\mathcal{F}) = V(\mathcal{G}) =:$ V. Also, there exists $F_0 := \{1,3\} \subseteq \overline{S}$ and $\mathcal{C}(F_0) := \{\{1,2,3\}, \{1,2,4\}, \{1,3,4\}\}$. Let $\Omega := \frac{2}{3}$. Then, $|\mathcal{G}_C| \leq \Omega \cdot |\mathcal{G}|$, for each $C \in \mathcal{C}(F_0)$. Indeed:

- $\mathcal{G}_{C_1=\{1,2,3\}} := \{\{2,3\}\}$, then $|\mathcal{G}_{C_1}| = 1 \leq \frac{2}{3} \cdot 2 = \frac{4}{3}$;
- $\mathcal{G}_{C_2=\{1,2,4\}} := \{\{1,4\}\}$, then $|\mathcal{G}_{C_2}| = 1 \leq \frac{2}{3} \cdot 2 = \frac{4}{3}$;
- $\mathcal{G}_{C_3=\{1,3,4\}} := \{\{1,4\}\}$, then $|\mathcal{G}_{C_3}| = 1 \leq \frac{2}{3} \cdot 2 = \frac{4}{3}$.

To conclude this section, we provide the following lemma to bound the cardinality of a full cover.

Lemma 3.6.6. *Let $(\mathcal{E}, \mathcal{D})$ be a clean pair of clutters defined over the vertex set V. Let $\mathcal{C}$ be a full cover of $\mathcal{E}^*$ constructed by applying Lemma 3.6.2. Then, $|\mathcal{E}| \cdot |\mathcal{D}| \geq |\mathcal{C}|$.*

Proof. Since every member of $\mathcal{E}$ is a minimal transversal of $\mathcal{D}$, for every $E \in \mathcal{E}$ and for every $v \in V$, there exists a $D_v \in \mathcal{D}$ such that $E \cap D_v = \{v\}$. Clearly, for every $u \in E \setminus \{v\}$, there exists $D_u \in \mathcal{D}$ such that $D_u \neq D_v$. Thus, $|\mathcal{E}| \cdot |\mathcal{D}| = \sum_{E \in \mathcal{E}} |\mathcal{D}| \geq \sum_{E \in \mathcal{E}} |E| \geq |\mathcal{C}|$. □

3.7 A new decomposition algorithm

In this section, I describe a new decomposition algorithm, based on all previous lemmas recalled defined in Sections 3.4, 3.5, and 3.6. Indeed, Algorithm 3.2 combines the classical decomposition by [Fredman and Khachiyan, 1996], their concept of frequency, and the notion of full covers, introduced by [Boros and Makino, 2009] and implicitly used also by [Elbassioni, 2008].

3.7.1 Algorithm

Algorithm 3.2 describes a recursive procedure called MissingPair$(\mathcal{E}, \mathcal{D}, \Omega)$, whose structure follows by the three pairwise incompatible cases of Fact 3.5.2. Whenever given a clean pair $(\mathcal{E}, \mathcal{D})$ and a threshold parameter $\Omega \in [0, 1]$, the procedure checks the duality of the two input clutters $\mathcal{E}$ and $\mathcal{D}$. In case they are not dual of each other, then MissingPair$(\mathcal{E}, \mathcal{D}, \Omega)$ returns a missing pair $\{S, \overline{S}\}$ such that S intersects every set in $\mathcal{E}$ and $\overline{S}$ intersects every set in $\mathcal{D}$.

Simple subproblems, where $|\mathcal{E}|$ or $|\mathcal{D}|$ is equal to 1, or when their product is less than 4, are solved by a procedure called SimpleDuality$(\mathcal{E}, \mathcal{D})$ without making any recursive calls. This is described in Algorithm 3.1, reported below, just before Algorithm 3.2.

Algorithm 3.1: SIMPLEDUALITY($\mathcal{E}$,$\mathcal{D}$)

Case 1.1: $|\mathcal{E}| = 1$. Let $\mathcal{E} := \{E\}$ and let $\mathcal{E}^* := \{\{v\} \mid v \in E \in \mathcal{E}\}$.
Then, if any, return a set S in $\mathcal{E}^* \setminus \mathcal{D}$.

Case 1.2: $|\mathcal{D}| = 1$. This is dual of Case 1.1.

Case 2: $|\mathcal{E}| = |\mathcal{D}| = 2$. Let $\mathcal{E} := \{E_1, E_2\}$ and let $\mathcal{D} := \{D_1, D_2\}$.
If E_1 and E_2 are disjoint, then $\mathcal{E}^* := \{(i, j) \mid i \in E_1, j \in E_2\}$. Otherwise, let $\mathcal{E}^* := \{i \mid i \in E_1 \cap E_2\} \cup \{(i, j) \mid i \in E_1 \setminus (E_1 \cap E_2), j \in E_2 \setminus (E_1 \cap E_2)\}$.
Then, if any, return a set S in $\mathcal{E}^* \setminus \mathcal{D}$. Otherwise, similarly compute $\mathcal{D}^*$ and, if any, return a set S in $\mathcal{D}^* \setminus \mathcal{E}$.

Algorithm 3.2: MISSINGPAIR($\mathcal{E}$, $\mathcal{D}$, Ω)

0. Let $V := V(\mathcal{E})$.
   Case 0: $|\mathcal{E}|$ or $|\mathcal{D}| = 1$, or $|\mathcal{E}| \cdot |\mathcal{D}| \leq 4$. Return SIMPLEDUALITY$(\mathcal{E}, \mathcal{D})$.

1. ***Case 1: there exists a*** $v \in V$ ***such that*** $\omega_{v,\mathcal{E}} \geq \Omega$ ***and*** $\omega_{v,\mathcal{D}} \geq \Omega$ ***both hold.***
   Apply Lemma 3.4.6 to recursively call MISSINGPAIR$\left(\mathcal{E}^{\overline{\{v\}}}, \mathcal{D}_{\overline{\{v\}}}, \Omega\right)$ and MISSINGPAIR$\left(\mathcal{E}_{\overline{\{v\}}}, \mathcal{D}^{\overline{\{v\}}}, \Omega\right)$.

2. Otherwise, obtain an Ω-bipartition $\{S, \overline{S}\}$ of the vertex set V by means of Lemma 3.5.1. By Fact 3.5.2, only one of the following cases can occur:

   Case 2.1: there exists a set $E_0 \in \mathcal{E}$ ***such that*** $E_0 \subseteq \overline{S}$. Construct the set family $\mathcal{C}(E_0)$, that is a full cover of the dual $\mathcal{E}^*$ by Lemma 3.6.3. Then, resort on Lemma 3.6.5 to decompose the problem, making $|\mathcal{C}(E_0)|$ recursive calls to the procedure MISSINGPAIR$(\mathcal{E}^C, \mathcal{D}_C, \Omega)$, for each $C \in \mathcal{C}(E_0)$;

   Case 2.2: there exists a set $D_0 \in \mathcal{D}$ ***such that*** $D_0 \subseteq S$. This is dual to Case 2.1. Build a full cover $\mathcal{C}(D_0)$ of the dual $\mathcal{D}^*$ and decompose the problem into $|\mathcal{C}(D_0)|$ recursive calls, again by using Lemma 3.6.3 and Lemma 3.6.5, calling the procedure MISSINGPAIR$(\mathcal{E}_C, \mathcal{D}^C, \Omega)$, for each $C \in \mathcal{C}(D_0)$;

   Case 2.3: no set of $\mathcal{E}$ ***is contained in*** $\overline{S}$ ***and no set of*** $\mathcal{D}$ ***is contained in*** S. A missing pair has already been found. Indeed, S intersects every set in $\mathcal{E}$ and $\overline{S}$ intersects every set in $\mathcal{D}$. Return the bipartition $\{S, \overline{S}\}$ as a NO-certificate of the duality of $\mathcal{E}$ and $\mathcal{D}$.

The following lemma just assures that the preconditions are preserved at each recursive call of Algorithm 3.2.

Lemma 3.7.1. *Let $(\mathcal{E}, \mathcal{D})$ be a clean pair over the vertex set V and let $S \subseteq V$. Then $(\mathcal{E}_S, \mathcal{D}^S)$ and $(\mathcal{E}^S, \mathcal{D}_S)$ are clean pairs over S.*

Proof. By Lemma 3.4.1, it suffices to show that $(\mathcal{E}_S, \mathcal{D}^S)$ is clean. Consider any $(E, D) \in \mathcal{E}_S \times \mathcal{D}^S$. Then, $E \in \mathcal{E}$ with $E \subseteq S$ and $D \cup \overline{S} \in \mathcal{D}^+$. Therefore, E intersects $D \cup \overline{S}$ since $(\mathcal{E}, \mathcal{D})$ is clean and thus standard, and E intersects D as well for $E \cap \overline{S} = \emptyset$. This proves that also $(\mathcal{E}_S, \mathcal{D}^S)$ is standard. Assume $\mathcal{E}_S$ contains a set E that is not a minimal transversal of $\mathcal{D}^S$ and let $E' \subsetneq E$ be a transversal of $\mathcal{D}^S$. Since for every set $D \in \mathcal{D}$ the family $\mathcal{D}^S$ contains some subset of D, then E' is also a transversal of $\mathcal{D}$, contradicting that E is a minimal transversal of $\mathcal{D}$. Finally, assume $\mathcal{D}^S$ contains a set D that is not a minimal transversal of $\mathcal{E}_S$ and let $D' \subsetneq D$ be a transversal of $\mathcal{E}_S$. Then, $D' \cup \overline{S}$ is a transversal of $\mathcal{E}$. But then $\mathcal{D}^S$ should contain a subset of D', contradicting that $\mathcal{D}^S$ is a clutter. □

Lemma 3.7.2 (Correctness of Algorithm 3.2)**.** *Let $(\mathcal{E}, \mathcal{D})$ be a clean pair of clutters defined over the vertex set $V(\mathcal{E}) = V(\mathcal{D}) =: V$. Then, Algorithm 3.2 correctly decides whether $\mathcal{E}$ and $\mathcal{D}$ are dual of each other or not.*

Proof. The decomposition rules applied in Case 1 and in Cases 2.1, 2.2, and 2.3 are correct by Lemma 3.4.6 and Lemma 3.6.1, respectively. Also, by Lemma 3.7.1, any pair given as input to a subproblem generated by Algorithm 3.2 is clean. Moreover, either Algorithm 3.2 does not return anything, when $\mathcal{E}$ and $\mathcal{D}$ are dual, or it reports a NO-certificate of their duality in the form of a bipartition $\{S, \overline{S}\}$ (Case 2.3) or of a missing transversal (Case 0). □

The perspective we adopt here, with Algorithm 3.2, makes more explicit the inherent symmetry of the problem, by looking for a symmetric NO-certificate provided by the pair $\{S, \overline{S}\}$. Besides the fact that having two possible alternative views on something is always intriguing, we point out that seeking for a bipartition $\{S, \overline{S}\}$ with the properties required by Lemma 3.6.5, or excluding that one such exists, lends itself to implicit enumeration approaches (exhaustive or not). Branching on whether an element $v \in V$ should go on one side or the other makes more explicit the analogy with techniques widely used in operations research or in constraint programming. All previous methods could be rewritten under this viewpoint.

3.7.2 Summary

Table 3.1 indicates which main decomposition techniques are used by state-of-the-art algorithms solving Problem 3.3.

	Variable-based decomposition	**Full covers**
[Fredman and Khachiyan, 1996]	✓	
[Elbassioni, 2008]	✓	✓
[Boros and Makino, 2009]		✓
[Raffaele and Rizzi, 2021]	✓	✓

Table 3.1: Comparison among state-of-the-art algorithms to solve the Monotone Boolean Duality problem.

3.7.3 Time complexity

Here we discuss the time complexity of Algorithm 3.2. We use n, σ, and π to indicate the cardinality of $|V|$, the sum $|\mathcal{E}| + |\mathcal{D}|$, and the product $|\mathcal{E}| \cdot |\mathcal{D}|$. In previous works, the last quantity was called *volume* and denoted by v (see, e.g., [Fredman and Khachiyan, 1996] and [Elbassioni, 2008]).

Remark 3.7.1. *At every recursive decomposition of the problem in Algorithm 3.2, all parameters involved (i.e., n, $|\mathcal{E}|$, $|\mathcal{D}|$, and, thus, σ and π) strictly decrease. Therefore, the values of these parameters in the original instance offer a valid upper bound on those in any generated subproblems.*

Remark 3.7.2. *Working on a super-polynomial bound, our goal is to decrease the constant at the exponent. Thus, we focus only on providing an estimate on the number of recursive calls. The estimate on the total running time is then obtained by multiplying this bound with the maximum that can be spent locally in one single call, that is, when disregarding what taken in the recursive calls (i.e., considering only the internal operations performed to create subproblems and handle their answers), clearly polynomial in the parameters. As such, this multiplicative factor is indeed negligible (actually, it could even be entirely disregarded since the vast majority of the subproblems will be leaves of the recursion tree). By Remark 3.7.1, considering the parameters of the original instance yields a safe bound on this maximum. Notice now that the parameters of the instance also affect the branching factor, but again monotonically. Therefore, by Remark 3.7.1, if for some of these parameters we employ their values in the original instance, we anyhow obtain valid upper bounds. We are going to resort on such convenient simplifications in the analyses that follow.*

As in [Fredman and Khachiyan, 1996] and [Elbassioni, 2008], we exploit the following equation, where $\chi(a,b)$ is the unique positive root:

$$\left(\frac{\chi(a,b)}{b}\right)^{\chi(a,b)} = a. \tag{3.10}$$

We observe that, when $\log a = \omega(b)$, $\chi(a,b) \approx \frac{\log a}{\log\log a}$, whereas, if $\log a = O(b)$ and $a \geq 1$, then $\chi(a,b) = \Theta(b)$.

Theorem 3.7.1 (Number of recursive calls of Algorithm 3.2)**.** *Let $(\mathcal{E},\mathcal{D})$ be a clean pair of clutters defined over the vertex set V, such that $|\mathcal{E}|\cdot|\mathcal{D}| := \pi$ and $\left(\frac{\chi(\pi,2)}{2}\right)^{\chi(\pi,2)} = \pi$. Then,* Problem *3.3 can be solved with at most $\pi^{k\chi(\pi,2)}$ recursive calls of Algorithm 3.2, where $k := \max\left\{\ln 2, 1-\log_{\chi(\pi,2)} 2\right\}$.*

Proof. Let $P(\pi)$ be the number of recursive calls of Algorithm 3.2 needed to solve the instance $(\mathcal{E},\mathcal{D})$ of Problem 3.3. We introduce $\Omega := \frac{1}{\chi(\pi,2)}$ as a threshold. Similarly to [Fredman and Khachiyan, 1996] and [Elbassioni, 2008], we prove by induction that $P(\pi) \leq \pi^{k\cdot\chi(\pi,2)}$, for some $k \in \mathbb{R}^+$.

Base case This corresponds to Case 0, where $|\mathcal{E}|$ or $|\mathcal{D}|$ is equal to 1, or when $|\mathcal{E}|\cdot|\mathcal{D}| = \pi \leq 4$. Actually, we can assume $\pi < 6$, since $\pi = 5$ would imply either $|\mathcal{E}|$ or $|\mathcal{D}|$ trivial. Since we do not make any recursive calls, $P(\pi) = 1 \leq \pi^{k\cdot\chi(\pi,2)}, \forall k > 0, k \in \mathbb{R}^+$.

Inductive step We assume that $\pi \geq 6$ and analyse each remaining case of Algorithm 3.2.

Case 1. Lemma 3.4.6 yields two subproblems. Thus, we get the following recurrence:

$$P(\pi) \leq 1 + 2P\left((1-\Omega)\pi\right). \tag{3.11}$$

By induction, we obtain:

$$\begin{aligned}
P(\pi) &\leq 1 + 2\left((1-\Omega)\pi\right)^{k\cdot\chi((1-\Omega)\pi,2)} \\
&\leq 2\left((1-\Omega)\pi\right)^{k\cdot\chi(\pi,2)}, \text{ for } k \geq 0.32 \\
&= 2\left(1-\Omega\right)^{k\cdot\chi(\pi,2)} \pi^{k\cdot\chi(\pi,2)} \\
&\leq 2\left(e^{-\Omega}\right)^{k\cdot\chi(\pi,2)} \pi^{k\cdot\chi(\pi,2)}, \text{ since } 1-x \leq e^{-x}, \text{ for all } x \in \mathbb{R} \\
&\leq 2\left(\frac{1}{e}\right)^{\frac{1}{\chi(\pi,2)}\cdot k\cdot\chi(\pi,2)} \pi^{k\cdot\chi(\pi,2)} \\
&= \left(\frac{2}{e^k}\right) \pi^{k\cdot\chi(\pi,2)} \leq \pi^{k\cdot\chi(\pi,2)}, \text{ for } k \geq \ln 2.
\end{aligned}$$

Case 2.1. There exists a set $E_0 \in \mathcal{E}$ such that $E_0 \subseteq \overline{S}$, thus we construct a full cover $\mathcal{C}(E_0)$ containing at most $\sum_{E \in \mathcal{E}} |E|$ sets.
This translates in the following recurrence:

$$P(\pi) \leq 1 + \left(\sum_{E \in \mathcal{E}} |E| \right) P(\Omega\pi). \tag{3.12}$$

By induction and monotonicity of $\chi(\cdot, 2)$, we obtain:

$$\begin{aligned} P(\pi) &\leq 1 + \left(\sum_{E \in \mathcal{E}} |E| \right) (\Omega\pi)^{k \cdot \chi(\Omega\pi, 2)} \\ &\leq 1 + \pi(\Omega\pi)^{k \cdot \chi(\Omega\pi, 2)} \text{ by Lemma 3.6.6} \\ &\leq \pi(\Omega\pi)^{k \cdot \chi(\pi, 2)}, \text{ for } k \geq 0.23 \\ &\leq \pi\Omega^{k \cdot \chi(\pi, 2)} \pi^{k \cdot \chi(\pi, 2)}. \end{aligned}$$

We want $\pi\Omega^{k \cdot \chi(\pi,2)} \pi^{k \cdot \chi(\pi,2)} \leq \pi^{k \cdot \chi(\pi,2)}$, thus we set $\pi\Omega^{k \cdot \chi(\pi,2)} \leq 1$.
Since $\Omega := \frac{1}{\chi(\pi,2)}$, we get:

$$\begin{aligned} \left(\frac{1}{\chi(\pi, 2)} \right)^{k \cdot \chi(\pi, 2)} &\leq \frac{1}{\pi} \\ \chi(\pi, 2)^{k \cdot \chi(\pi, 2)} &> \pi \\ k \cdot \chi(\pi, 2) &\geq \log_{\chi(\pi,2)} \pi. \end{aligned}$$

By definition, $\left(\frac{\chi(\pi,2)}{2} \right)^{\chi(\pi,2)} = \pi$.
Thus:

$$\begin{aligned} k &\geq \frac{1}{\chi(\pi, 2)} \log_{\chi(\pi,2)} \left(\frac{\chi(\pi, 2)}{2} \right)^{\chi(\pi,2)} \\ k &\geq \frac{1}{\chi(\pi, 2)} \chi(\pi, 2) \log_{\chi(\pi,2)} \left(\frac{\chi(\pi, 2)}{2} \right) \\ k &\geq \log_{\chi(\pi,2)} \left(\frac{\chi(\pi, 2)}{2} \right) \\ k &\geq 1 - \log_{\chi(\pi,2)} 2. \end{aligned}$$

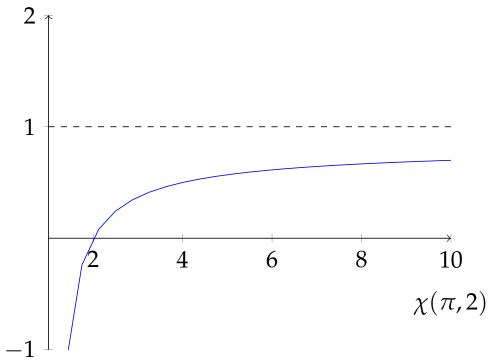
Case 2.2. This is symmetric to Case 2.1 by clutter duality from Lemma 3.4.1.

Case 2.3. This is a lucky case in which the computation ends without calling any subproblems.

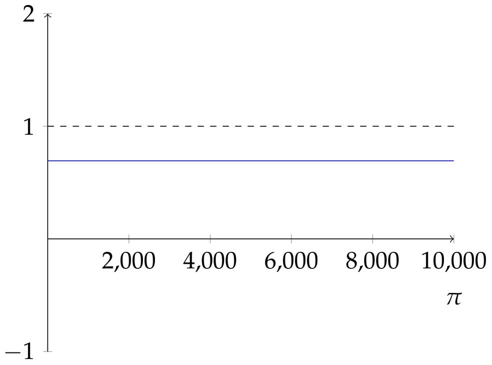
Summarizing, we obtain the following bounds for π and k:

- Case 0: $\pi < 6$, $k \geq 0$;
- Case 1: $\pi \geq 6$, $k \geq 0.32$ and $k \geq \ln 2$;
- Case 2: $\pi \geq 6$, $k \geq 0.23$ and $k \geq 1 - \log_{\chi(\pi,2)} 2$.

Figure 3.2a shows that, for any $\chi(\pi, 2) \geq 2$ (i.e., $\pi \geq 1$), the minimum value of k required in Case 2 is always greater than 0 but generally also less than 1. Only when $\pi \to \infty$, then also $\chi(\pi, 2) \to \infty$, and $k = 1$.



(a) $1 - \log_{\chi(\pi,2)} 2$.



(b) $k = \max\left\{\ln 2, 1 - \log_{\chi(\pi,2)} 2\right\}$.

FIGURE 3.2: Trends of $1 - \log_{\chi(\pi,2)} 2$ and $k = \max\left\{\ln 2, 1 - \log_{\chi(\pi,2)} 2\right\}$ as $\chi(\pi, 2)$ and π increase, respectively.

Since the value of k cannot be fixed in advance for any π, to finally get to our claimed bound, we just take the maximum value between $\ln 2$ and $1 - \log_{\chi(\pi,2)} 2$ (Figure 3.2b). $\square$

Example 3.4 Let $(\mathcal{E}, \mathcal{D})$ be a clean pair of clutters such that $\pi := |\mathcal{E}| \cdot |\mathcal{D}| = 2^{20}$. Then, $\chi(\pi, 2) = 9.13$ and $k := \max\{\ln 2, 1 - \log_{9.13}(2)\} = \ln 2$.

Theorem 3.7.2 (Running time of Algorithm 3.2). *Let $(\mathcal{E}, \mathcal{D})$ be a clean pair of clutters defined over the vertex set V, such that $|\mathcal{E}| + |\mathcal{D}| =: \sigma$, $|\mathcal{E}| \cdot |\mathcal{D}| =: \pi$, and $\left(\frac{\chi(\pi,2)}{2}\right)^{\chi(\pi,2)} = \pi$. Then, PROBLEM 3.3 can be solved in $\sigma^{4k\chi(\sigma,2)}$ time, where $k := \max\left\{\ln 2, 1 - \log_{\chi(\pi,2)} 2\right\}$.*

Proof. By monotonicity of $\chi(\cdot, 2)$, $\chi(\sigma^2, 2) > \chi(\sigma, 2)$.

Since $\sigma^2 = \left(\frac{\chi(\sigma^2,2)}{2}\right)^{\chi(\sigma^2,2)} = \left(\frac{\chi(\sigma,2)}{2}\right)^{2\chi(\sigma,2)}$, we have $\chi(\sigma^2, 2) < 2\chi(\sigma, 2)$.

By definition, $\pi = |\mathcal{E}| \cdot |\mathcal{D}| \leq \left(\frac{(|\mathcal{E}|+|\mathcal{D}|)}{4}\right)^2 \leq (|\mathcal{E}| + |\mathcal{D}|)^2 \leq \sigma^2$.

Hence, by Theorem 3.7.1, the number of recursive calls of Algorithm 3.2 is at most $\pi^{k\chi(\pi,2)} \leq (\sigma^2)^{k\chi(\sigma^2,2)} < \sigma^{4k\chi(\sigma,2)}$. □

Thus, even if the bounds obtained seem promising, in the worst case Algorithm 3.2 offers the same performance guarantee as the state-of-the-art algorithms by [Fredman and Khachiyan, 1996].

3.8 The space issues in dualization

Now, after providing a new decomposition approach to solve PROBLEM 3.3 (i.e., duality), we shift our attention to PROBLEM 3.4 (i.e., dualization). In particular, in this section, I first provide a simple procedure to solve PROBLEM 3.4 by relying on Algorithm 3.2, and I underline the main issues in terms of space. To avoid them, first I recall some implementation techniques used by [Tamaki, 2000]. Then, I consequently adapt the decomposition rules applied in Algorithm 3.2. Finally, I propose an algorithm to solve PROBLEM 3.4 by using only polynomial space.

3.8.1 A naive procedure to dualize

Since PROBLEM 3.3 is a decision problem, we do not worry about the memory occupied by Algorithm 3.2 when solving it. Indeed, the memory used by a subproblem already handled can be exploited again by another subproblem still to be examined. Moreover, given an input pair $(\mathcal{E}, \mathcal{D})$, what we are interested in is just computing one missing pair $\{S, \overline{S}\}$ such that S is a transversal of $\mathcal{E}$ and $\overline{S}$ is a transversal of $\mathcal{D}$. When we find the first NO-certificate of such a form, we return it and conclude that $\mathcal{E}$ and $\mathcal{D}$ are not dual of each other.

Instead, space has to be carefully taken into account when solving PROBLEM 3.4. In this case, given an input clutter $\mathcal{E}$, we want to return its dual $\mathcal{E}^* = \mathcal{D}$, i.e., we want to return all the minimal transversals of $\mathcal{E}$. Also, we would like each minimal transversal to be output exactly once.

As mentioned in Section 3.1, PROBLEM 3.4 can be reduced to PROBLEM 3.3. In particular, given a clutter $\mathcal{E}$ and a threshold parameter $\Omega \in [0, 1]$, we can rely on Algorithm 3.2 to obtain the following procedure.

Algorithm 3.3: NaiveDualize($\mathcal{E}, \Omega$)

```
1. Let $\mathcal{D} := \emptyset$.
2. While MissingPair($\mathcal{E}, \mathcal{D}, \Omega$) $= \{S, \overline{S}\}$:
     Add $S$ to $\mathcal{D}$.
3. Return $\mathcal{D}$.
```

Algorithm 3.3 is pretty simple, but it has a main issue in Step 2. Indeed, it requires that all minimal transversals generated are stored in memory and used as input in successive iterations. In terms of space, this could be prohibitive, e.g., when a clutter has an exponential number of transversals, as in Example 3.3.

3.8.2 Reporting transversals one by one by using processes

[Tamaki, 2000] faced the same issue of Algorithm 3.3 when applying the decompositions by [Fredman and Khachiyan, 1996] to solve Problem 3.4. To overcome it, he first reformulated FK-B and then proposed a *process*-based algorithm, which occupies polynomial space only. This approach was also used by [Elbassioni, 2008].

The key idea implemented is the following: "*Each process reports transversal one by one, rather than returning the set of transversals as a whole*" ([Tamaki, 2000], page 5). To be able to do so, [Tamaki, 2000] introduces a (sub)process for each recursive call of his dualization algorithm. A subprocess is in charge of the execution of the subproblem corresponding to that recursive call. The first call of the algorithm, corresponding to the original input problem, is associated with the so-called *root* process. A subprocess knows its parent (sub)process. In a given moment, only one subprocess, called *current*, is active and being executed, whereas all other subprocesses are either suspended (waiting to be resumed) or have already terminated their execution. The root process, as well as a generic subprocess, can initiate some child subprocesses, according to the decomposition rule that can be applied in the recursive algorithm. In this case, the current process saves its state and is suspended, and one of the child subprocesses becomes active. When there is a *return* statement, the subprocess returns to its parent (sub)process either a new minimal transversal or a termination signal. In the former case, the subprocess is suspended until the transversal is output. In the latter case, it is terminated and not resumed anymore, thus the storage it has occupied can be freed and used by another subprocess not yet terminated.

Thus, together with the recursion tree of the algorithm, where nodes are associated with subproblems of the original input problem, we can assume to have a *process tree*. The algorithm developed by [Tamaki, 2000] schedules processes and keeps all those that are ready and waiting for execution in a stack. Hereafter we report the main invariants maintained:

(I1) except for the current, all ready processes are in the stack, where entries are ordered according to the right-to-left ordering in the process tree;

(I2) if a process is waiting, it is waiting for its rightmost child to respond;

(I3) the current process is always on the path in the process tree from the root to the rightmost leaf.

3.8.3 Reformulating decomposition rules

In order to avoid storing all minimal transversals as in Algorithm 3.3, we can return transversals one by one by using processes as in [Tamaki, 2000]. To do so, we need to reformulate the two decomposition rules used in Algorithm 3.2, i.e., the variable-based decomposition and the full-cover decomposition.

To adapt the variable-based decomposition in Lemma 3.4.6, we can proceed as follows. Given $\mathcal{E}$, we can decompose $\mathcal{E}$ as:

$$\mathcal{E} = \left\{ E \cup \{v\} \mid E \in \mathcal{E}^{\overline{\{v\}}} \right\} \cup \mathcal{E}_{\overline{\{v\}}}.$$

Lemma 3.8.1 (Decomposition of $\mathcal{E}^*$). *Let $\mathcal{E}$ be a clutter defined over the vertex set V. Then*

$$\mathcal{E}^* = \mathit{Minimals}\left(\left(\mathcal{E}^{\overline{\{v\}}}\right)^* \cup \left\{ \{v\} \cup D \mid D \in \left(\mathcal{E}_{\overline{\{v\}}}\right)^* \setminus \left(\mathcal{E}^{\overline{\{v\}}}\right)^* \right\} \right).$$

Proof. A set $S \subseteq V$ is a minimal transversal of $\mathcal{E}$ if and only if either $v \notin S$ and S is a transversal of $\mathcal{E}^{\overline{\{v\}}}$, or $S \setminus \{v\}$ is a transversal of $\mathcal{E}_{\overline{\{v\}}}$. □

The two clutters $\left(\mathcal{E}^{\overline{\{v\}}}\right)^*$ and $\left\{ \{v\} \cup D \mid D \in \left(\mathcal{E}_{\overline{\{v\}}}\right)^* \setminus \left(\mathcal{E}^{\overline{\{v\}}}\right)^* \right\}$ are disjoint. Thus, we can ensure that each transversal of $\mathcal{E}$, obtained through this variable-based decomposition, is output only once.

When we apply a full-cover decomposition, actually we do not need to work on disjoint clutters, but we can add an easy testing procedure based on the following lemma.

Lemma 3.8.2. *Let $\mathcal{E}$ be a clutter defined over the vertex set V. Let $\mathcal{C} := \left\{C_1, \ldots, C_{|\mathcal{C}|}\right\}$ be a full cover of $\mathcal{E}^*$ and let $\mathcal{D} \subseteq \mathcal{E}^*$. Let $T \subseteq V$. Then, given $C_i \in \mathcal{C}$, $T \in \left(\mathcal{E}^{C_i}\right)^* \setminus \mathcal{D}_{C_i}$ and $T \nsubseteq C_j$, for all $C_j \in \mathcal{C}$ such that $j < i$, if and only if $T \in \mathcal{E}^* \setminus \mathcal{D}$ and $T \subseteq C_i$.*

Proof. By Lemma 3.6.1, $T \in \left(\mathcal{E}^{C_i}\right)^* \setminus \mathcal{D}_{C_i}$ is a transversal of $\mathcal{E}$. Since $T \in \left(\mathcal{E}^{C_i}\right)^*$, then $T \subseteq C_i$. Given this and the fact that $T \notin \mathcal{D}_{C_i}$, then $T \notin \mathcal{D}$.

Conversely, clearly $T \notin \mathcal{D}_{C_i}$, because $T \notin \mathcal{D}$ and the filter operation computes a subset of it. Also, $T \in \left(\mathcal{E}^{C_i}\right)^*$ because $T \subseteq C_i$ and, by the definition of the projection operation, $\mathcal{E}^{C_i} := Minimals\,\{E \cap C_i \mid E \in \mathcal{E}\}$. Thus, for each set in $\mathcal{E}^{C_i}$, there exists at least one element in T intersecting it. Finally, since T is a minimal transversal of $\mathcal{E}$, it is also a minimal transversal of $\mathcal{E}^{C_i}$. $\square$

We build a full cover $\mathcal{C} := \{C_1, \ldots, C_{|\mathcal{C}|}\}$ of $\mathcal{E}^*$ and we apply Lemma 3.6.5 to decompose the input problem in $|\mathcal{C}|$ subproblems. In a given subproblem $\mathcal{E}^{C_i}$, by Lemma 3.8.2 we avoid generating a transversal T of $\mathcal{E}$ more than once by simply checking whether T is a subset of any $C_j \in \mathcal{C}$, with $j < i$. Indeed, if T is not a new transversal, then it has already been generated in a previous subproblem as a transversal of $\mathcal{E}^{C_j}$, $j < i$, where $V\left(\mathcal{E}^{C_j}\right) = C_j$ and $T \subseteq C_j$. Thus, it is enough to store all the sets in the full cover $\mathcal{C}$ and consider the related subproblems one by one, solving them sequentially. Differently from [Tamaki, 2000], we do not need to sort sets in a lexicographic or any specific order, to perform this polynomial check.

3.8.4 A polynomial-space algorithm for dualization

In this subsection, I propose Algorithm 3.4, a polynomial-space procedure to solve Problem 3.4. Algorithm 3.4 improves Algorithm 3.3 by integrating the variable-based decomposition and the full-cover decomposition as described in Lemma 3.8.1 and in Lemma 3.8.2, respectively.

To keep the pseudocode of Algorithm 3.4 simple, we do not explicitly write all initiated, suspended, and resumed subprocesses. Actually, we assume the procedure is implemented as a generator and that transversals are reported one by one by using the keyword *yield* (as allowed by some programming languages, such as Python[1]).

[1]See, e.g., https://docs.python.org/3/reference/expressions.html#yieldexpr.

Algorithm 3.4: DUALIZE($\mathcal{E}$)

```
0. Let $V := V(\mathcal{E})$.
   *Case 0:* $|\mathcal{E}| = 1$. Let $E$ be the only set in $\mathcal{E}$. For each $v \in E$, yield $\{v\}$.
1. *Case 1: there exists a* $v \in V$ *such that* $\omega_{v,\mathcal{E}} \geq \Omega$.
   Apply Lemma 3.8.1. For each $D$ in DUALIZE$\left(\mathcal{E}^{\overline{\{v\}}}\right)$, yield $D$.
   Then, for each $D$ in DUALIZE$\left(\mathcal{E}_{\overline{\{v\}}}\right)$, if $D$ is not a transversal of $\mathcal{E}^{\overline{\{v\}}}$, yield $D$.
2. *Case 2: there does not exist a* $v \in V$ *such that* $\omega_{v,\mathcal{E}} \geq \Omega$.
   Apply Lemma 3.6.1. Let $E_0 \in \mathcal{E}$ and use Lemma 3.6.3 to construct the set
   family $\mathcal{C}(E_0) := \left\{C_1, \dots, C_{|\mathcal{C}(E_0)|}\right\}$, that is a full cover of $\mathcal{E}^*$.
   Then, resort on Lemma 3.6.5 to make $|\mathcal{C}(E_0)|$ recursive calls by considering
   each $C_i \in \mathcal{C}(E_0)$ and applying Lemma 3.8.2:
   for each $T$ in DUALIZE$(\mathcal{E}^{C_i})$, if $T \nsubseteq C_j$ for each $j < i$, yield $T$.
```

Lemma 3.8.3 (Correctness and uniqueness of solutions of Algorithm 3.4)**.** *Let $\mathcal{E}$ be a clutter defined over the vertex set V. Then, Algorithm 3.4 correctly lists all minimal transversals of $\mathcal{E}$ without duplicates.*

Proof. In Case 0, Algorithm 3.4 returns $|E|$ minimal transversals of $\mathcal{E}$. In Case 1, by Lemma 3.8.1, the two recursive calls of the variable-based decomposition are disjoint, return all minimal transversals only, and no minimal transversal is output more than once. In Case 2, by Lemma 3.6.1, all minimal transversals of $\mathcal{E}$ are output and, by Lemma 3.8.2, no minimal transversal is output more than once. □

3.8.5 Space complexity

In this subsection, I show that Algorithm 3.4 can be implemented by occupying polynomial space only.

Given a clutter $\mathcal{E}$, defined over a vertex set V, we consider all the pairs (v, E) where v is an element of V and E is a set of $\mathcal{E}$, such that $v \in E$.

Lemma 3.8.4 (Storage cost of an instance)**.** *Given a clutter $\mathcal{E}$, defined over the vertex set V, the cost to represent $\mathcal{E}$ by using doubly linked list is $O(\eta)$, where $\eta := \sum_{E \in \mathcal{E}} |E|$.*

This is a natural measure that arises, corresponding to what we would like to pay, in terms of space, to represent $\mathcal{E}$. Indeed, we can introduce a doubly linked list for each $v \in V$ and a doubly linked list for each $E \in \mathcal{E}$, which contain links to the related pairs (v, E).

Through its list, an element v can easily get information about all the sets $E \in \mathcal{E}$ in which it appears. Similarly, a set E is able to retrieve all the elements $v \in V$ it contains. It is also easy to remove a set E from $\mathcal{E}$, by removing each pair (v, E) in its doubly linked list and updating the corresponding list of v. Likewise, when removing an element v from V, we remove each pair (v, E) in its doubly linked list and we update the list of the related E. (Instead of removing a pair, we can alternatively associate a Boolean value to each pair, that states whether it has to be considered or discharged). Anyway, since removing an element from a doubly linked list requires $\theta(1)$ time and $O(1)$ space, this kind of operation on $\mathcal{E}$ or on V is linear in η.

Also other procedures, which can occur several times when solving PROBLEM 3.3 and PROBLEM 3.4, can be performed linearly as well by adopting this measure.

Lemma 3.8.5 (Transversal testing)**.** *Let $\mathcal{E}$ be a clutter defined over a vertex set V, and let $D \subseteq V$. Then, we can test whether D is a transversal of $\mathcal{E}$ in $O(\eta)$ time, where $\eta := \sum_{E \in \mathcal{E}} |E|$.*

For each $v \in D$, we assign a Boolean value set to true to every pair (v, E) in the doubly linked list of v. Then, for each $E \in \mathcal{E}$, we check that in its doubly linked list there exists at least one pair set to true. In the worst case, this checking is linear in the number of pairs (v, E).

Lemma 3.8.6 (Minimal-transversal testing)**.** *Let $\mathcal{E}$ be a clutter defined over a vertex set V, and let $D \subseteq V$. Then, we can test whether D is a minimal transversal of $\mathcal{E}$ in $O(\eta)$ time, where $\eta := \sum_{E \in \mathcal{E}} |E|$.*

In this case, instead of a Boolean value for each pair (v, E), we can use a counter for each $E \in \mathcal{E}$. First, we iterate over the elements $v \in D$: for each pair (v, E), we increment the counter of the related E and we keep a record of v as the last element of D hitting E. Then, we consider $\mathcal{E}'$ as the sets $E \in \mathcal{E}$ hit only once: if every element of D is the last hitting element of at least one set of $\mathcal{E}'$, then D is a minimal transversal of $\mathcal{E}$. Also this checking is linear in the number of pairs (v, E).

Now we can proceed analysing the storage required by Algorithm 3.4. Similarly to [Tamaki, 2000], we consider two types of storage: the *temporary* storage and the *process* storage. The former is used to test whether a given subset of the vertex set V is a (minimal) transversal of a given hypergraph.

Lemma 3.8.7 (Temporary storage of Algorithm 3.4)**.** *Let $\mathcal{E}$ be a clutter defined over a vertex set V and let $\mathcal{E}$ be the original input of Algorithm 3.4. Then, the temporary storage* *required by Algorithm 3.4 to test whether S is a (minimal) transversal of $\mathcal{E}$ (or of some subproblem of $\mathcal{E}$) is $O(\eta)$, where $\eta := \sum_{E\in\mathcal{E}} |E|$.*

Proof. This follows directly from Lemma 3.8.4, Lemma 3.8.5, and Lemma 3.8.6. □

Now we consider the process storage. We note that, at any time during the execution of Algorithm 3.4, the nodes in the recursive tree are generated only when needed. Thus, we actually keep in memory only an *active subtree* composed of all nodes that have been visited but whose execution has not terminated yet. As [Tamaki, 2000] reported, for each subprocess associated with such a node, we store:

(i) its parent (sub)process pointer and its resumption point;

(ii) the storage it needs to represent the clutters it has created and passed as input to a subprocess that is still in the process tree;

(iii) the list cells for which it is the creator and which are still referred to by some subprocesses in the process tree.

For each time step t and a process p in the process tree at time t, let $W_t(p)$ be the number of words of process storage that p is responsible for. Let $\mathcal{E}$ be the input clutter for p. If Case 0 applies to p, as process storage we need only $O(1)$ for the parent process pointer and the resumption point. If Case 1 applies to p, we need this $O(1)$ plus $O(\eta)$ to represent the two subproblems $\mathcal{E}_{\overline{\{v\}}}$ and $\mathcal{E}^{\overline{\{v\}}}$. Finally, if Case 2 applies to p, we need $O(1)$ plus $O(\eta)$ to represent the full cover $\mathcal{C}(E_0)$, and $O(\eta)$ to represent the subproblem $\mathcal{E}^{C_i}$, with $C_i \in \mathcal{C}(E_0)$. Thus:

$$W_t(p) \leq \begin{cases} O(1) \text{ in Case 0,} \\ O(\eta) \text{ in Case 1 and Case 2.} \end{cases}$$

Let $M_t(p)$ denote the sum of $W_t(q)$ over all the processes q in the process tree rooted at p. Let $M(\eta)$ denote the worst possible value of $M_t(p)$, over all possible input $\mathcal{E}$ to the process p, with $\sum_{E\in\mathcal{E}} |E| \leq \eta$ over all possible steps t.

Lemma 3.8.8 (Process storage of Algorithm 3.4)**.** *Let $\mathcal{E}$ be a clutter defined over the vertex set V, such that $\sum_{E\in\mathcal{E}} |E| =: \eta$. Then, $M(\eta)$, i.e., the number of words of process storage required by Algorithm 3.4 to list all minimal transversals of $\mathcal{E}$, is $O(\eta^2)$.*

Proof. We analyse each possible case.

Case 0: $M(\eta) \leq O(1)$.

Case 1 and Case 2: $M(\eta) \leq O(\eta) + M(\eta - 1)$.

Let c be a large enough constant such that $M(1) \leq c$ and $M(\eta) \leq c\eta + M(\eta - 1)$, for $\eta > 1$. We show by induction that $M(\eta) \leq c\eta^2$, for every $\eta \geq 1$. The base case $\eta = 1$ is trivial. Let $\eta > 1$ and suppose $M(k) \leq ck^2$ holds for every $k < \eta$. When $M(\eta) \leq c\eta + M(\eta - 1)$ holds, then we have

$$\begin{aligned} M(\eta) &\leq c\eta + c(\eta - 1)^2 \\ &= c(\eta + \eta^2 - 2\eta + 1) \\ &= c(\eta^2 - \eta + 1) \\ &\leq c\eta^2, \text{ for } \eta \geq 1. \end{aligned}$$

□

Theorem 3.8.1 (Storage of Algorithm 3.4)**.** *Let $\mathcal{E}$ be a clutter defined over the vertex set V, such that $\sum_{E \in \mathcal{E}} |E| =: \eta$. Then, Algorithm 3.4 lists all minimal transversals of $\mathcal{E}$ by using $O\left(\eta^2\right)$ space.*

Proof. This follows directly from Lemma 3.8.7 and Lemma 3.8.8. □

Finally, we provide the following theorem about the number of processes generated by Algorithm 3.4.

Theorem 3.8.2 (Number of processes of Algorithm 3.4)**.** *Let $\mathcal{E}$ be a clutter defined over the vertex set V. Then, the total number of processes generated by Algorithm 3.4 to solve* Problem *3.4 is equal to the number of recursive calls of Algorithm 3.2 stated in Theorem 3.7.1.*

Proof. By definition, there is a one-to-one correspondence between the processes initiated by Algorithm 3.4 and the nodes in the recursion tree of Algorithm 3.2. □

Could [Tamaki, 2000]'s algorithm and Algorithm 3.4 be executed in parallel? Each ready subprocess, waiting in the stack for execution, could be handled by a different available processor, but we should be careful in maintaining the invariants (I1)–(I3) described in Subsection 3.8.2 (unless of course we modify the structure of the algorithm). However, if we decide to use several processors, then we should be willing to occupy some additional space. Indeed, we would actually keep in memory more than one active subtree. By Theorem 3.8.2, the number of processes generated is equal to the number of recursive calls of Algorithm 3.2. Thus, the maximum number of processors to be used should be delicately set, in order not to compromise too much the advantage achieved in terms of space.

3.9 Further developments

To conclude this chapter, I discuss some further developments and interesting research lines for future work.

3.9.1 Generalizations and extensions of full-covers methods

Here I provide some other methods to construct full covers, which lead to other possible experimental decompositions. The main goal to keep in mind is to try to define a full cover with a small size (i.e., the number of subproblems that would be generated by its application) and such that the cardinality of its sets is also not big (i.e., that would allow obtaining smaller subproblems).

Lemma 3.6.3 can be generalized as follows.

Lemma 3.9.1 (Generalized full cover construction)**.** *Let $V := \{1, 2, \ldots, n\}$. Let $\mathcal{E}$ be a set family over V, and $\mathcal{E}' \subseteq \mathcal{E}$. Then, the following is a full cover of $\mathcal{E}^*$:*

$$\bigcup_{K=\{k_1,\ldots,k_{|K|}\}\in(\mathcal{E}')^*} \left(\bigcup_{k\in K} \left\{ K \cup \left(V \setminus \bigcup_{\{E\in\mathcal{E}:k\in E\}} E \right) \right\} \right). \tag{3.13}$$

Proof. Let D be any set of $\mathcal{E}^*$. Then, D contains some minimal transversal of $\mathcal{E}'$. Let K be a minimal transversal of $\mathcal{E}'$ contained in D. For every $k \in K$, $k \in D$ and, since D is a minimal transversal for $\mathcal{E}$, then there exists an $E \in \mathcal{E}$ such that $D \cap E = \{k\}$. Therefore, $D \setminus K \subseteq \left(V \setminus \bigcup_{\{E\in\mathcal{E}:k\in E\}} E \right)$. □

Here it is an extension of Lemma 3.6.3.

Lemma 3.9.2 (Stronger full cover construction (extended))**.** *Given a set family $\mathcal{E}$, let $E_0 = \{e_1, e_2, \ldots, e_p\}$ and $F_0 = \{f_1, f_2, \ldots, f_q\}$ be two disjoint sets in $\mathcal{E}$. Then, the following is a full cover of $\mathcal{E}^*$:*

$$\bigcup_{i=1}^{p} \bigcup_{j=1}^{q} \bigcup_{\{E\in\mathcal{E}|e_i\in E\}} \bigcup_{\{F\in\mathcal{E}|f_j\in F\}} \Big\{ \{e_i, f_j\} \cup (V \setminus \{e_1, e_2 \ldots e_{i-1}\} \setminus \{f_1, f_2 \ldots f_{j-1}\} \setminus E \setminus F) \Big\}. \tag{3.14}$$

Proof. Let D be any set of $\mathcal{E}^*$. Then, D intersects both E_0 and F_0. Let i (resp., j) be the smallest natural such that $e_i \in D \cap E_0$ (resp., $f_j \in D \cap F_0$); from this $D \subseteq V \setminus \{e_1, e_2 \ldots e_{i-1}\} \setminus \{f_1, f_2 \ldots f_{j-1}\}$. Since D is a minimal transversal of $\mathcal{E}$, then there exists a set $E \in \mathcal{E}$ such that $D \cap E = \{e_i\}$ and a set $F \in \mathcal{E}$ such that $D \cap F = \{f_j\}$. Therefore, $D \setminus \{e_i, f_j\} \subseteq V \setminus \{e_1, e_2 \ldots e_{i-1}\} \setminus \{f_1, f_2 \ldots f_{j-1}\} \setminus E \setminus F$. □

[Boros and Makino, 2009] attributed a first example of the use of full covers to [Elbassioni, 2008], which implicitly used the following kind.

Lemma 3.9.3 (Full cover construction – [Elbassioni, 2008], [Boros and Makino, 2009])**.** *Given a clutter $\mathcal{E}$ and a minimal transversal $D_0 \in \mathcal{E}^*$, the family*

$$\mathcal{C}(D_0) = \{V \setminus \{i\} \mid i \in D_0\} \cup \{D_0\} \tag{3.15}$$

forms a full cover of $\mathcal{E}^*$.

Proof. Let D be any set of $\mathcal{E}^*$ other than D_0. Then, there exists an $i \in D_0 \setminus D$ since $\mathcal{E}^*$ is a clutter. □

Lemma 3.9.3 can be regarded as a special case of the following.

Lemma 3.9.4. *Given a set family $\mathcal{E}$, let $\mathcal{D}_0 \subseteq \mathcal{E}^*$. Then, the following is a full cover of $\mathcal{E}^* \setminus \mathcal{D}_0$:*

$$\mathcal{D}_0 \cup \left\{ \overline{P} \mid P \in \mathcal{D}_0^* \right\}. \tag{3.16}$$

Proof. Let D be any set in $\mathcal{E}^* \setminus \mathcal{D}_0$. Since $\mathcal{E}^*$ is a clutter, then $\overline{D}$ is a transversal of $\mathcal{E}^* \setminus D$, and also a transversal of $\mathcal{D}_0 \setminus D = \mathcal{D}_0$, since $D \notin \mathcal{D}_0 \subseteq \mathcal{E}^*$. Being a transversal of $\mathcal{D}_0$, $\overline{D}$ contains a set $P \in \mathcal{D}_0^*$. From $P \subseteq \overline{D}$, we get $D \subseteq \overline{P}$. □

We can join Lemma 3.6.3 and Lemma 3.9.3 as described right away.

Lemma 3.9.5 (Full cover construction (two sides))**.** *Given a set family $\mathcal{E}$, let $E_0 \in \mathcal{E}$ and fix its elements in a specific order $E_0 = \{e_1, e_2, \ldots, e_t\}$. Assume given a minimal transversal $D_0 \in \mathcal{E}^*$. Then, the following is a full cover of $\mathcal{E}^*$:*

$$\{D_0\} \cup \bigcup_{d_0 \in D_0} \bigcup_{i=1}^{t} \bigcup_{\{E \in \mathcal{E} \mid e_i \in E\}} \Big\{ \{e_i\} \cup (V \setminus \{e_1, e_2 \ldots e_{i-1}\} \setminus E \setminus \{d_0\}) \Big\}. \tag{3.17}$$

Proof. Let D be any set in $\mathcal{E}^* \setminus \{D_0\}$. Then, there exists a $d_0 \in D_0 \setminus D$ since $\mathcal{E}^*$ is a clutter. Moreover, D intersects E_0. Let i be the smallest natural such that $e_i \in D \cap E_0$; from this, $D \subseteq V \setminus \{e_1, e_2 \ldots e_{i-1}\}$. Since D is a minimal transversal of $\mathcal{E}$, then there exists a set $E \in \mathcal{E}$ such that $D \cap E = \{e_i\}$. For such an E it holds that $D \setminus \{e_i\} \subseteq V \setminus \{e_1, e_2 \ldots e_{i-1}\} \setminus E \setminus \{d_0\}$. □

In a more general form:

Lemma 3.9.6 (Generalized full cover construction (two sides))**.** *Given a set family $\mathcal{E}$, let $E_0 \in \mathcal{E}$ and fix its elements in a specific order $E_0 = \{e_1, e_2, \dots, e_t\}$. Assume $\mathcal{D}_0 \subseteq \mathcal{E}^*$. Then, the following is a full cover of $\mathcal{E}^* \setminus \mathcal{D}_0$:*

$$\mathcal{D}_0 \cup \bigcup_{P \in \mathcal{D}_0^*} \bigcup_{i=1}^{t} \bigcup_{\{E \in \mathcal{E} | e_i \in E\}} \Big\{ \{e_i\} \cup (\overline{P} \setminus \{e_1, e_2 \dots e_{i-1}\} \setminus E) \Big\}. \tag{3.18}$$

Proof. Let D be any set in $\mathcal{E}^* \setminus \mathcal{D}_0$. Since $\mathcal{E}^*$ is a clutter, then $\overline{D}$ is a transversal of $\mathcal{E}^* \setminus D$, and also a transversal of $\mathcal{D}_0 \setminus D = \mathcal{D}_0$, since $D \notin \mathcal{D}_0 \subseteq \mathcal{E}^*$. Being a transversal of $\mathcal{D}_0$, $\overline{D}$ contains a $P \in \mathcal{D}_0^*$. Moreover, D intersects E_0. Let i be the smallest natural such that $e_i \in D \cap E_0$; from this, $D \subseteq V \setminus \{e_1, e_2 \dots e_{i-1}\}$. Since D is a minimal transversal of $\mathcal{E}$, then there exists a set $E \in \mathcal{E}$ such that $D \cap E = \{e_i\}$. For such an E it holds that $D \setminus \{e_i\} \subseteq \overline{P} \setminus \{e_1, e_2 \dots e_{i-1}\} \setminus E$. □

3.9.2 Investigating other measures and techniques

At present, the proposed decomposition does not take advantage of the reductions used in the FK-B algorithm by [Fredman and Khachiyan, 1996]. Thus, a natural development would be to find a way to integrate and exploit them, aiming for the desired stronger bound.

Anyway, all recalled and presented decompositions rely on the concept of frequency of an element $v \in V$ in the sets of $\mathcal{E}$ and $\mathcal{D}$. However, this measure may not entirely express the relevance of v for $\mathcal{E}$ and $\mathcal{D}$. Indeed, a more appropriate measure should take into account not only the percentage of sets containing v, but also the cardinality of these where v appears: is v a crucial element for these sets? A better measure could be the following:

$$\omega_{v,\mathcal{E}} := \sum_{\{E \in \mathcal{E} | v \in E\}} \frac{1}{|E|}.$$

Another interesting direction to investigate is whether these measures could actually be computed, i.e., obtained as a result of the optimization of a linear programming model.

Also, experiments could be used to try to characterize or even tabulate the values of Ω for the branching decisions.

Moreover, one could consider the possibility to implement some machine-learning techniques to learn how to better balance the subproblems sizes, based on what learned from the analysis of enumeration trees from previous instances, or even based on the partial and ongoing exploration of the current enumeration tree.

Chapter 4

Conclusions

"The artist's imagination is a world of potentialities that no work will succeed in realizing. What we experience by living is another world, answering to other forms of order and disorder. [...] A writer carries out operations that involve the infinity of their imagination or the infinity of the contingency that may be attempted, or both, by means of the infinity of linguistic possibilities in writing."

Italo Calvino, *Visibility*, in *Six Memos for the Next Millennium*

In this thesis, I moved from the context of optimization to the one of listing, by investigating some interesting and fundamental enumeration problems that can also have applications and connections with operations research.

In **Chapter 1**, I gave a brief introduction to the research area of enumeration problems and algorithms. In particular, I provided some basic definitions, presented some applications arising from different disciplines, discussed complexity classes used to classify enumeration problems, and illustrated the most common techniques to design and analyse enumeration algorithms.

In **Chapter 2**, I discussed the first problem of my interest: given an undirected and connected graph $\mathcal{G} = (V, E)$, with $n = |V|$ and $m = |E|$, list all its bonds (i.e., minimal cuts). The state of the art by [Tsukiyama et al., 1980] offers an enumeration algorithm which is output-linear in the number of edges of the graph. Thus, I wondered about the possibility of developing an algorithm output-linear in the number of vertices.

I was able to answer affirmatively. Indeed, I reduced the problem of listing bonds and the one of listing all s, t-bonds to the one of listing all S, T-bonds, where S and T are two nonempty disjoint subsets of V. By giving relevance to cut-vertices and biconnectivity, I provided a new recursive procedure which carefully generates only fertile subproblems. I developed an algorithm in two versions that share the same high-level description. The former outputs each bond in $O(m)$, guaranteeing the same time complexity as [Tsukiyama et al., 1980]. The latter relies on dynamic data structures to achieve better bounds. In particular, I exploited two data structures defined by [Holm et al., 2001] to maintain connectivity and biconnectivity through a maximal forest and a spanning tree of the graph, respectively. Moreover, I defined a third data structure operating over a tree to check critical cut-vertices. These techniques, together with amortized analysis, allowed for presenting the first output-linear algorithm to list all S, T-bond shores in $\widetilde{O}(n)$ per bond, and all S, T-bonds as edge-sets in $\widetilde{O}(n) + |\delta_{\mathcal{G}}(S, \overline{S})|$.

In the literature, there are a few examples of enumeration problems tackled by using dynamic graph algorithms. Thus, this is one of the most original aspects of this work, which could enhance applications and improvements in the state of the art of other enumeration problems.

In **Chapter 3**, also motivated by a fundamental question in linear programming about the double representation of a polyhedron, I studied the *Monotone Boolean Duality*: given two set clutters $\mathcal{F}$ and $\mathcal{G}$, we want to establish if they are dual of each other. The remarkable result by [Fredman and Khachiyan, 1996] put the Monotone Boolean Duality problem between P and coNP, by providing a quasi-polynomial time algorithm running in $\sigma^{o(\log \sigma)}$ time, where $\sigma = |\mathcal{F}| + |\mathcal{G}|$. Since their contribution, several attempts have been made to improve this bound and also to determine the exact complexity of the problem. Thus, the research question was the following: what could be another decomposition approach able to improve the state of the art?

I was able to provide a new recursive algorithm to solve the Monotone Boolean Duality problem. This combines steps of a variable-based decomposition with steps based on the full covers, depending on which yields the most promising reduction in terms of the sizes of subproblems. The bound offered by the proposed decomposition is indeed $\sigma^{4k\chi(\sigma,2)}$, where $k \leq 1$. More exactly, $k := \max\left\{\ln 2, 1 - \log_{\chi(\pi,2)} 2\right\}$, where $\pi = |\mathcal{F}| \cdot |\mathcal{G}|$. When $\pi \to \infty$, also $\chi(\pi, 2) \to \infty$, and $k = 1$. Thus, in the worst case, this approach still has the same complexity as [Fredman and Khachiyan, 1996].

I also considered the Monotone Boolean Dualization problem: given a clutter $\mathcal{F}$, we want to compute its dual $\mathcal{G}$. This problem can be reduced to the Monotone Boolean Duality problem. In the last part of the chapter, I showed how to adapt the new decomposition to solve this problem by using polynomial space only.

These results do not exactly match what I hoped for. For sure, they do not settle the intriguing open question about the exact complexity class of the two problems. Anyway, they encourage the study, development, and experimentation of other kinds of decompositions and approaches. The whole exploration process has been symmetrized, and is now cast under a new light and a more standard perspective. This paves the way to the applicability of several standard speeding-up techniques with complete enumeration algorithms.

Appendix A

On communicating operations research

"[...] the contemporary novel as an encyclopedia, as a method of knowledge, and above all as a network of connections between the events, the people, and the things of the world."
Italo Calvino, *Multiplicity*, in *Six Memos for the Next Millennium*

Operations research (OR) has a manifold nature, and so had my doctorate. Indeed, besides enumeration problems and algorithms, I dedicated part of these last four years also to explore other fields. Here I do not list all of them, but I just give a brief overview on two particular side projects, which led me to investigate diverse research directions, and to relate with people with different backgrounds and age, from various contexts and sectors. Mostly, they allowed me to discuss the topic and the relevance of communicating OR to laypeople, which is the main content of this appendix.

The former project, presented in Section A.1, is related to an industrial case study, which required to analyse, evaluate, and possibly improve the current implementation of the interlibrary loan service provided by the public libraries in the provinces of Brescia and Cremona (Italy). Together with a few practitioners of the public company involved, I studied the real situation and identified an optimization problem. Here I just provide the mathematical formulation and I outline the setting of the experimental evaluation performed.

The latter project, described in Section A.2, is instead an educational initiative aimed to introduce OR in higher secondary schools. This is a joint project developed in collaboration with two other young researchers in OR, Dr. Gabriella Colajanni (University of Catania) and Dr. Alessandro Gobbi (University of Brescia), another young researcher in mathematics education, Dr. Eugenia Taranto (University of Catania), and a high-school mathematics teacher, Dr. Marinella Picchi (IIS Antonietti). After studying and comparing the state of the art, we designed some teaching units, which we have been experimenting in a few Italian high schools since March 2021. In this section, I report the origin of the project, its main features, the ongoing teaching experimentations, and some work done also in the field of mathematics education.

In Section A.3, I discuss the higher purpose of these projects, that is, making OR more known, exploited, and appreciated, not only in academia. This is directly linked to the relevance and the difficulty to communicate OR to laypeople, who often do not have an idea of what OR is and how it could be applied. On this, first I bring some considerations by researchers and practitioners. Then, I describe three specific contexts where to promote and communicate OR with different purposes and actions.

Finally, I draw my conclusions in Section A.4.

A.1 The industrial case study of Province of Brescia

In this section, I describe the first side project needed to introduce and discuss the topic of communicating OR.

The *interlibrary loan* is a service that allows the users of a library to borrow or obtain copies of items from other libraries. The term includes the phases of borrowing, lending, and document delivery of the materials ([Boucher, 1997]). Through such a service, libraries can share and exchange their catalogues, making more resources available to their users. This is precious when, for instance, libraries are very small, or located in little towns far from the cities, where there may be more than one library able to satisfy users' requests. Nowadays, different libraries can belong to the same network, sharing the same information system, accessing their own catalogue, as well as the ones of other libraries.

Here we describe the case of *Province of Brescia*, an Italian public company which includes a department called the *Libraries Office*, in charge of the management of the public libraries in the provinces of Brescia and Cremona (see Figure A.1).

The Libraries Office promotes and coordinates the public libraries in the network, for instance by managing the information systems adopted and the data collected, taking care of the training and continuing education of the librarians and all other employees, and managing the interlibrary loan service. The Libraries Office

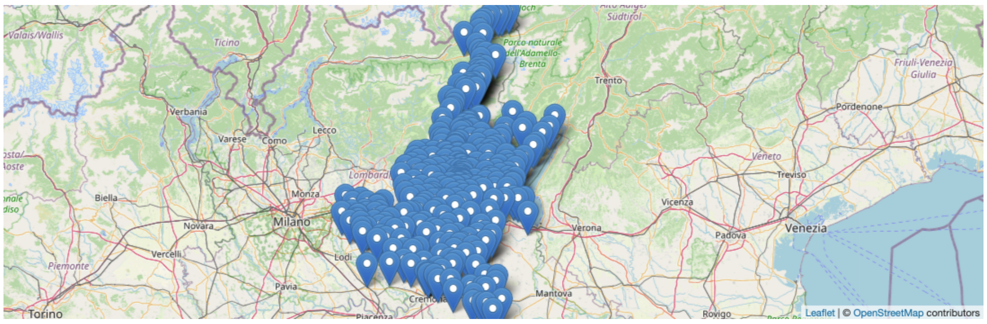


FIGURE A.1: Public libraries located in the provinces of Brescia and Cremona (© OpenStreetMap contributors).

offers an online catalogue where a user, after logging in, can perform several operations, such as checking its current loans or searching for any items owned by any libraries in the network. Once a desired item is found, the user can place a request to obtain a copy of it, by collecting it from a particular library the user explicitly chooses. If the chosen library owns a copy and this is available (i.e., not already on loan), then the item is made ready to be borrowed. Otherwise, the request is satisfied by another library in the network owning that item, which makes it ready for transit. The Libraries Office has an agreement with an external logistics company, which handles the transportation of the items in the whole network through a fleet of vehicles and couriers. When a courier stops in a library, it delivers all items addressed to that library and picks up all items that are instead addressed to other libraries in the network. In the current implementation, pickup and delivery operations are performed at the same time. However, couriers do not deliver right away what they have just picked-up, but they temporarily bring the collected items to the only depot in the network. There, in the following days, items are rearranged and assigned to other couriers to be delivered to their target libraries. In agreement with the Libraries Office, each library is visited by a courier a few times a week, according to a fixed calendar based on historical data. Also, libraries are divided into fixed groups, called *lines*. Figure A.2 shows the lines established for the libraries in the province of Brescia. Each line is associated with a fixed route, scheduled in all its stops. All routes start from the depot, where they also end.

Every year, the Libraries Office spends a fixed amount to rent the vehicles of the logistics company, plus some variable costs related to the length of the routes travelled by couriers. The Libraries Office is thus interested in evaluating the current implementation of the service in terms of routing and transportation costs, but also determining whether their approach with fixed lines and calendar is good or not. This is the main reason why we started collaborating in early 2020. During the first

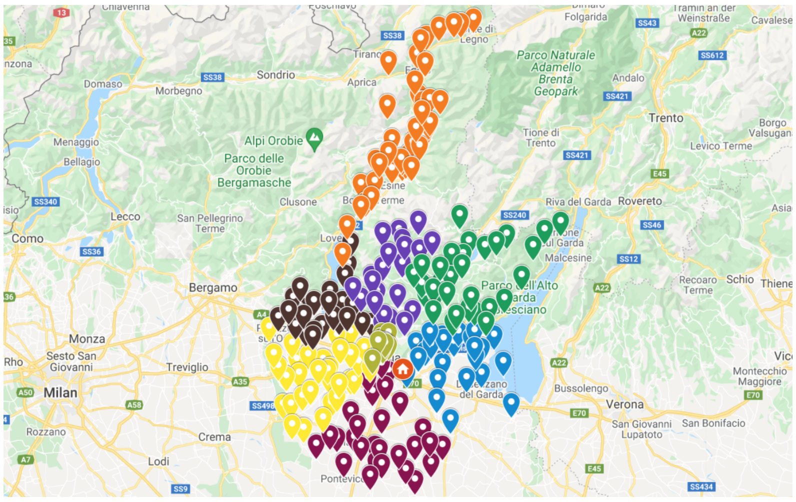


FIGURE A.2: Organization of the public libraries in the province of Brescia into a set of lines, each one represented with a different colour (Google map: https://shorturl.at/axEFR).

meetings, the company introduced us to the service and provided us the raw data related to all interlibrary loans performed in 2019 in the province of Brescia only.

A.1.1 Problem definition

Hereafter I provide a formal description of the problem.

We consider a time horizon of one day. Let $\mathcal{K} := \{1, \ldots, K\}$ be the fleet of available vehicles. Vehicles are heterogeneous, with different capacities q and working times w. All vehicles start their routes from the same depot D, to which they also return at the end.

Let $\mathcal{L} := \{1, \ldots, L\}$ be the set of public libraries to be served. Each library ℓ can have p_ℓ items to be picked-up or d_ℓ items to be delivered. According to the kind of operations performed, the service time s_ℓ to perform the operations varies. Each vehicle is able to do at least one type of operation. Let $\mathcal{K}_\ell \subseteq \mathcal{K}$ be the subset of vehicles in the fleet able to perform the operations required by library ℓ. All items to be delivered are already available at the depot D at the beginning of the day, whereas all items picked-up by vehicles' couriers are brought to the depot D at the end of the day. Each library ℓ has to be served by exactly one vehicle k during its opening window $[a_\ell, b_\ell]$, unless the courier of k owns a copy of the keys of ℓ. In that case, the service can be performed also after hours, by spending an additional time S to exploit the keys. Let $\mathcal{L}' \subseteq \mathcal{L}$ be the set of libraries whose keys are available, through some copies, to the whole fleet of vehicles.

We can model the problem as a complete graph $\mathcal{G} := (\mathcal{V}, \mathcal{A})$, where $\mathcal{V} := \{D\} \cup \mathcal{L}$ and $\mathcal{A}$ is the set of arcs (i, j) such that both i and j belong to $\mathcal{V}$. Every arc is characterized by two values: its travel cost c_{ij} and its travel time t_{ij}, representing the distance from i to j and the time required to go through the arc, respectively. Both these quantities satisfy the triangle inequality.

The main goal is to build an optimal schedule, i.e., to assign a vehicle to each library and to define the route of each vehicle, while satisfying all the constraints and minimizing the total travelled distance, the number of libraries unserved, and the number of vehicles used.

A.1.2 Mathematical formulation

The problem can be modelled as a single-depot pickup-and-delivery vehicle routing problem with time windows and heterogeneous fleet ([Desaulniers et al., 2002], [Montané and Galvão, 2002], and [Salhi et al., 2014]). We introduce:

- a binary variable u_k for each vehicle $k \in \mathcal{K}$, whose value is 1 when k is used, 0 otherwise;
- a binary variable g_ℓ for each library $\ell \in \mathcal{L}$, whose value is 1 when ℓ is not served by any vehicle, 0 otherwise;
- a binary variable $x_{\ell k}$ for each library $\ell \in \mathcal{L}$ and for each vehicle $k \in \mathcal{K}$, whose value is 1 when ℓ is served by k, 0 otherwise;
- a binary variable $h_{\ell k}$ for each library $\ell \in \mathcal{L}$ and for each vehicle $k \in \mathcal{K}$, whose value is 1 when the courier of k uses a copy of the keys of ℓ to provide the service requested, 0 otherwise;
- a binary variable y_{ijk} for each arc $(i, j) \in \mathcal{A}$ and for each vehicle $k \in \mathcal{K}$, whose value is 1 when k goes through the arc (i, j), 0 otherwise;
- a continuous variable l_{ijk} for each arc $(i, j) \in \mathcal{A}$ and for each vehicle $k \in \mathcal{K}$, representing the occupied capacity of k when going through arc (i, j);
- a continuous variable τ_v for each node $v \in \mathcal{V}$, representing the time when service starts in v;
- a continuous variable z_ℓ for each library $\ell \in \mathcal{L}$, representing when service ends (i.e., the lateness) in ℓ.

We get the following mixed-integer linear programming model, where M is a large positive constant.

$$\min \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c_{ij} y_{ijk} + M \sum_{\ell\in\mathcal{L}} g_\ell + M \sum_{k\in\mathcal{K}} u_k \tag{A.1}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} x_{\ell k} \le 1 \qquad \forall \ell \in \mathcal{L} \tag{A.2}$$

$$\sum_{k\in\mathcal{K}_\ell} x_{\ell k} + g_\ell = 1 \qquad \forall \ell \in \mathcal{L} \tag{A.3}$$

$$\sum_{(i,\ell)\in\mathcal{A}} y_{i\ell k} = x_{\ell k} \qquad \forall k \in \mathcal{K}, \forall \ell \in \mathcal{L} \tag{A.4}$$

$$\sum_{(\ell,j)\in\mathcal{A}} y_{\ell jk} = x_{\ell k} \qquad \forall k \in \mathcal{K}, \forall \ell \in \mathcal{L} \tag{A.5}$$

$$\sum_{\ell\in\mathcal{L}} y_{\ell Dk} = u_k \qquad \forall k \in \mathcal{K} \tag{A.6}$$

$$\sum_{\ell\in\mathcal{L}} y_{D\ell k} = u_k \qquad \forall k \in \mathcal{K} \tag{A.7}$$

$$h_{\ell k} = 0 \qquad \forall \ell \in \mathcal{L} \setminus \mathcal{L}', \forall k \in \mathcal{K} \tag{A.8}$$

$$h_{\ell k} \le \sum_{j\in\mathcal{L}} y_{\ell jk} \qquad \forall k \in \mathcal{K}, \forall \ell \in \mathcal{L} \tag{A.9}$$

$$\tau_\ell \ge a_\ell (1 - \sum_{k\in\mathcal{K}_\ell} h_{\ell k}) \qquad \forall \ell \in \mathcal{L} \tag{A.10}$$

$$\tau_\ell \le b_\ell + M \sum_{k\in\mathcal{K}_\ell} h_{\ell k} \qquad \forall \ell \in \mathcal{L} \tag{A.11}$$

$$\tau_\ell \ge \tau_D + t_{D\ell} - M(1 - \sum_{k\in\mathcal{K}} y_{D\ell k}) \qquad \forall (D,\ell) \in \mathcal{A} \tag{A.12}$$

$$\tau_j \ge \tau_\ell + s_\ell + t_{\ell j} + S \sum_{k\in\mathcal{K}} h_{\ell,k} - M(1 - \sum_{k\in\mathcal{K}} y_{\ell jk}) \qquad \forall (\ell,j) \in \mathcal{A} \tag{A.13}$$

$$z_\ell \ge \tau_\ell + s_\ell \qquad \forall \ell \in \mathcal{L} \tag{A.14}$$

$$\sum_{(i,j)\in\mathcal{A}} t_{ij} y_{ijk} + \sum_{\ell\in\mathcal{L}} s_\ell x_{\ell k} \le w_k u_k \qquad \forall k \in \mathcal{K} \tag{A.15}$$

$$l_{ijk} \le q_k y_{ijk} \qquad \forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{K} \tag{A.16}$$

$$\sum_{\ell\in\mathcal{L}} l_{D\ell k} = \sum_{\ell\in\mathcal{L}} d_\ell x_{\ell k} \qquad \forall k \in \mathcal{K} \tag{A.17}$$

$$\sum_{\ell\in\mathcal{L}} l_{\ell Dk} = \sum_{\ell\in\mathcal{L}} p_\ell x_{\ell k} \qquad \forall k \in \mathcal{K} \tag{A.18}$$

$$\sum_{(i,\ell)\in\mathcal{A}} l_{i\ell k} + \sum_{i\in\mathcal{V}} (p_\ell - d_\ell) y_{i\ell k} = \sum_{(\ell,j)\in\mathcal{A}} l_{\ell jk} \qquad \forall \ell \in \mathcal{L}, \forall k \in \mathcal{K} \tag{A.19}$$

$$u_k \in \{0,1\} \qquad \forall k \in \mathcal{K} \tag{A.20}$$

$$g_\ell \in \{0,1\} \qquad \forall \ell \in \mathcal{L} \tag{A.21}$$

$$x_{\ell k}, h_{\ell k} \in \{0,1\} \qquad \forall \ell \in \mathcal{L}, \forall k \in \mathcal{K} \tag{A.22}$$

$$y_{ijk} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{K} \tag{A.23}$$

$$l_{ijk} \ge 0 \qquad \forall (i,j) \in \mathcal{A}, \forall k \in \mathcal{K} \tag{A.24}$$

$$\tau_v \geq 0 \qquad \forall v \in \mathcal{V} \tag{A.25}$$

$$z_\ell \geq 0 \qquad \forall \ell \in \mathcal{L} \tag{A.26}$$

The objective function (A.1) asks to minimize the total travelled distance, the number of libraries unserved, and the number of vehicles used. Constraints (A.2) and (A.3) state that every library is either served by at most one vehicle or is unserved. Constraints (A.4)–(A.7) are needed to form a tour starting and ending in the depot D. A courier cannot exploit a copy of the keys of a library when this is not available (Constraints (A.8)) and, even when it is, the copy may be not used (Constraints (A.9)). Constraints (A.10) and (A.11) ask for a library to be served either during its opening hours or by exploiting a copy of the keys. Constraints (A.12) and (A.13) are needed to compute the start of service in the libraries, whereas Constraints (A.14) express when the service ends. Constraints (A.15) and (A.16) guarantee that a vehicle does not work more than its maximum working time and that its maximum capacity is never overcome, respectively. Constraints (A.17) and (A.18) define the initial and final capacity of each vehicle, whereas Constraints (A.19) update the capacity of a vehicle when it serves a library. Finally, Constraints (A.20)–(A.26) represent the variables introduced.

A.1.3 Experimental evaluation

From the available data and the registry information of the libraries, I derived four sets of instances, by considering (or not) the fixed lines and the fixed calendar implemented by the company. When relaxing these static constraints, the average instance size (i.e., the number of libraries to serve) increases, as shown in Table A.1.

Set name	**Fixed lines**	**Fixed calendar**	**Number of instances**	**Min size**	**Max size**	**Avg size**
`lines_calendar`	✓	✓	1584	1	25	12
`lines_no_calendar`	✓	✗	2485	1	33	18
`days_calendar`	✗	✓	267	1	90	74
`days_no_calendar`	✗	✗	313	2	176	148

TABLE A.1: Sets of instances derived from the data on the interlibrary loans in the province of Brescia in 2019.

I relied on *Open Source Routing Machine*[1] to get shortest distances and times between any two points in the network. I implemented the model in Python and solved it over all instances by using Gurobi 9.5 on a PC with Ubuntu 20.04.2 LTS

[1] `http://project-osrm.org/`

(GNU/Linux 5.4.0-90-generic x86_64), 16 core, 2 GHz, 32 GB RAM[2]. I set a time limit of five minutes for each instance.

Figure A.3 shows just a sample of the results obtained, where we can compare the optimal solutions of the instances in `lines_calendar` with an estimate of the actual costs. The actual costs were not available, but anyway I was able to compute some lower bounds by considering the fixed schedule of each line. From this, we can say that the mixed-integer linear programming model improves the current approach used by the company by at least 20.4%, saving at least $49,020$ km.

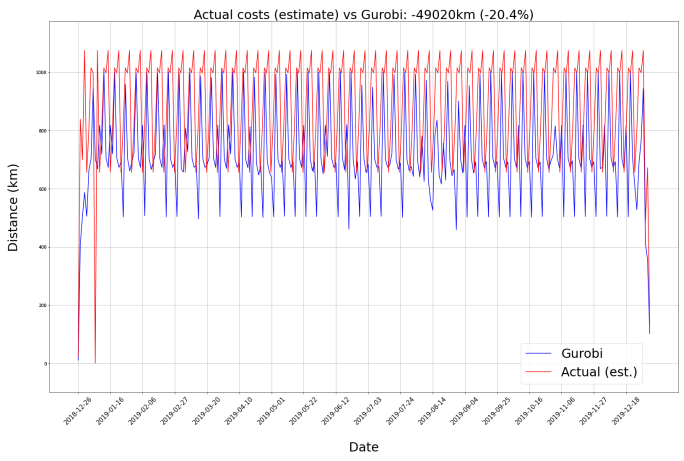


FIGURE A.3: Comparison between an estimate of the actual costs and the optimal solutions of the instances in `lines_calendar`.

A dynamic approach, computing each day with Gurobi the optimal routes to serve the public libraries, could replace the static approach implemented by the company. Even if this would be preferable in terms of savings, relying on a fixed calendar and fixed lines may still be a better method according to couriers or librarians. Indeed, if the routes completely change every day, couriers could not exploit their previous knowledge on roads or towns, and some delays could occur. Also, without a calendar, the librarians would not have a fixed schedule to follow, in order to prepare the items to be picked-up on time. A compromise could be to just relax the fixed schedule of the lines, by keeping the calendar and computing an optimal groups of libraries. On a given day, for each line, the items in the depot waiting to be delivered are known, as well as the items that have to be picked-up in the line. Thus, we can easily obtain an instance similar to the ones in `lines_calendar`, to be solved with Gurobi before the start of the working day. The schedule to follow would not drastically change every day, and couriers would avoid stopping at

[2]The virtual machine is hosted by the *Centro Piattaforme Tecnologiche* (CPT) of the University of Verona (`https://cpt.univr.it/en`).

libraries where there are no items either to be picked-up or delivered.

Moreover, in order to allow the Libraries Office to consider minor or major improvements to make, I performed a simple statistical analysis on the data of 2019. I extracted interesting information which I reported in a set of documents already delivered to the company. For instance, for each library in each line, I computed the average number of items picked-up and delivered by the courier at each stop. Also, by considering the calendar, I estimated the average delay for pickups and deliveries in each library. These data can be useful to the company in order to understand whether and where there are slowdowns in the whole service.

The complete experimental evaluation, as well as some further developments, will be reported in [Raffaele et al., 2022].

A.2 Mathematics education and OR

In this section, I provide an introduction to the second side project needed to deepen the topic of communicating OR.

A.2.1 A literature review

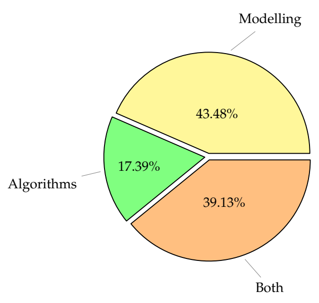
As mentioned, OR has a manifold nature, with plenty of applications in different fields, ranging from management to finance, from transportation to security, and many others. These fields are often related to STEM (i.e., *Science, Technology, Engineering and Mathematics*) disciplines. This is one of the reasons why presenting OR could be stimulating, not only at university level. Indeed, the study of OR problems and methods could be really helpful in increasing pupils' motivation towards mathematics and other scientific subjects, as well as in strengthening modelling and problem-solving skills. Also, Grades 9–12[3] would already have acquired all required mathematical background to learn some basic topics of OR. Despite this, OR is hardly ever included in higher secondary-school curricula.

By reflecting on these, in [Raffaele and Gobbi, 2021] we investigated the existence of specific educational initiatives sharing the aim of introducing OR to Grades 9–12. By using search engines, consulting repositories and journals, and checking programs of recent conferences in Europe, we collected 23 different initiatives[4].

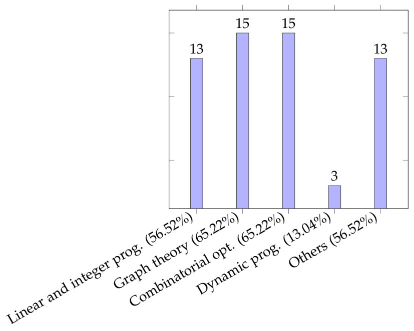
[3]Corresponding to 14–17 year-old students. We refer to the US/International Grades scale (see, e.g., `https://www.asmilan.org/admissions/grade-equivalents`).

[4]Map of the OR educational initiatives collected (February 2021): `https://shorturl.at/deyIZ`. This list is certainly not exhaustive. There may be plenty of initiatives we are not aware of and that we did not find through our research. If the reader is aware of other initiatives, please let us know.

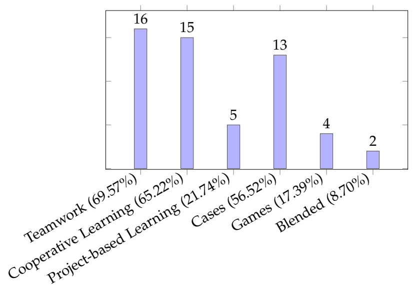
Then, we classified them as *promotion activities* done by societies and communities, *national and international projects*, *competitions*, *training courses* for teachers, *workshops* for students, and *didactic units* or lectures. We compared the initiatives according to their focus and objectives (modelling and algorithmics – see Figure A.4a), topics (mostly linear and integer programming, graph theory, and combinatorial optimization – see Figure A.4b), teaching methods (mostly based on constructionism, active learning, and collaboration – see Figure A.4c), instruments and software, and feedback.



(a) Objectives and focus



(b) OR topics



(c) Teaching methods

FIGURE A.4: Some comparisons among the OR educational initiatives collected in [Raffaele and Gobbi, 2021].

Also, all the main guidelines on mathematics education, stated by international organizations such as PISA, UNESCO, and European Union, share the objective to promote the learning of how to apply mathematics to model, analyse, and solve real-world problems. This is exactly what the study of OR would entail, enhancing problem-solving and modelling skills, and the implementation of algorithms.

For all the details, I refer the reader to [Raffaele and Gobbi, 2021].

A.2.2 The ROAR project

By making treasure of all the information collected in [Raffaele and Gobbi, 2021], we designed *Ricerca Operativa Applicazioni Reali* (ROAR; in English, *Real Applications of Operations Research*), a three-year project for higher secondary schools ([Colajanni et al., 2022]). One of the key features of ROAR is the study and analysis of *authentic problems* (i.e., problems coming from a particular field and recognized by workers in that field as possible situations they might face in their daily work [Niss, 1992]). These problems are inspired by students' everyday life or by industrial realities. Also, students are supposed to work in teams, to foster their collaboration.

ROAR is composed of three teaching units. The first one, addressed to Grade 10, provides an introduction to OR and to mathematical models and techniques, solvers, and other digital technologies useful to tackle optimization problems. The second unit, for Grade 11, concerns common graph-theory problems and algorithms. The third unit, for Grade 12, is on the implementation of OR methods and algorithms in a programming language students are already familiar with. It also involves the learning of an algebraic modelling programming language.

The design of the first teaching unit is completely described in [Colajanni et al., 2022], in terms of objectives, prerequisites, and instructors' roles. Also, we provided full details about the first implementation of the unit, occurred from March to May 2021 in a Grade-10 class at the scientific high school IIS Antonietti in Iseo (Brescia, Italy). Indeed, we illustrated the positioning of the unit in the mathematics program of the class, the teaching methods adopted, the digital technologies used, the organization of the lectures, and the feedback received from students and teachers.

Moreover, all the lectures were video-recorded, including all group-work sessions. This allowed us to deeply analyse the work of some groups as case studies, by also evaluating their protocols (i.e., images, PDFs, screenshots, and Excel sheets they produced). By referring to some tasks assigned throughout the whole teaching unit, we investigated some research questions linking OR and mathematics education. For instance, we investigated whether it is appropriate to include OR in regular mathematics lectures, or how collaborative group work and the use of digital technologies can foster the development of the modelling and problem-solving competences. Through qualitative and quantitative analysis, we showed how the answers are positive, and how such activities could impact students' understanding and appreciation of OR. For more details, I refer the reader to [Taranto et al., 2022].

Beyond the manuscripts, all the material produced, in terms of the slides and the texts of the problems used in the experimentations, is available on a public repository, both in Italian and in English[5]. Indeed, we wish that it will be used by other researchers or higher secondary school teachers, to develop similar teaching units, other extra-curricular workshops or seminars. This is exactly what happened in November 2021, with another experimentation in a Grade-12 class in Piazza Armerina, Enna (Italy), and with a training course based in Catania addressed to some higher-secondary school teachers (Italy)[6]. Both these two initiatives were managed by Dr. Colajanni and Dr. Taranto.

A.3 On communicating operations research

Besides their specific objectives, the projects described in Section A.2 and Section A.1 can be seen as two different ways to fulfil the *third mission* of universities, that is, trying to generate knowledge outside academic environments to the benefit of the social, cultural, and economic development ([Compagnucci and Spigarelli, 2020]). In this sense, the two projects share the purpose to communicate OR to laypeople, in order to increase their awareness. Precisely with regard to the communication of this discipline, there are some issues that are worth considering and discussing.

A.3.1 The issue of visibility of operations research

As mentioned, OR finds application in many fields. In the last few years, it has also taken into account important aspects such as sustainability (see, e.g., [Dekker et al., 2012]) and shared mobility (see, e.g, [Mourad et al., 2019]). Moreover, since March 2020, several works have been dedicated to manage and solve several optimization problems arisen after the spread of the COVID-19 pandemic (see, e.g., [Shen, 2020] for a general survey, or [INFORMS, 2020] and [The OR Society, 2020] for the specific web-pages developed by the two international OR associations).

However, despite all the results OR could achieve since its birth in the 1940s, it is still almost unknown to the public. On the contrary, other buzzwords have been spreading on and on, such as "machine learning", "big data" or "data science". If we compare on Google Trends[7] the worldwide interest in these search terms over the months of 2021, we obtain basically the same results I reported in [Raffaele, 2021] over three months of 2020. In Figure A.5, the numbers represent search interest relative to the highest point on the chart: a value of 100 is the peak popularity

[5]`https://github.com/aliceraffaele/ROAR`
[6]Map of ROAR initiatives in Italy (December 2021): `https://shorturl.at/bjnqP`.
[7]`https://trends.google.com/`

for the input terms; a value of 50 means that the term is half as popular; a score of 0 means that there are not enough data. Even if the comparison is in relative terms, we can observe that OR searches are constant and low throughout time.

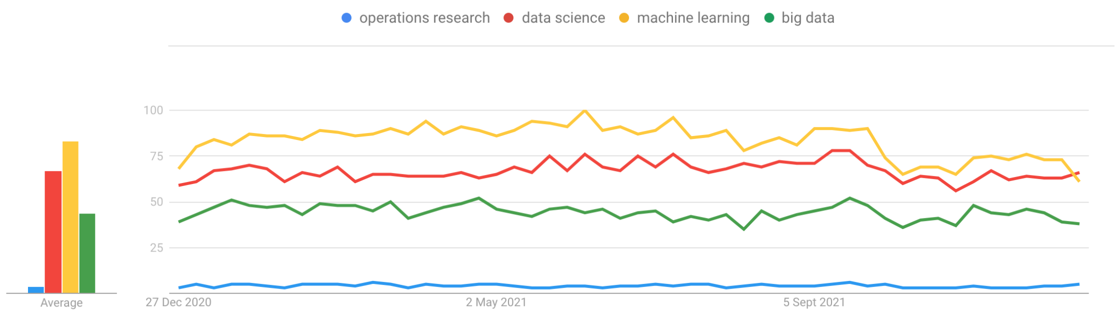


FIGURE A.5: Interest over time of the search terms "operations research", "data science", "machine learning", and "big data", in the period [December 27, 2020 – December 23, 2021].

OR is still remaining invisible, even being quite used. For instance, people are shopping online more and more, receiving their goods at home; when they book a plane ticket, they are offered the possibility to choose their preferred seating, by paying some fees accordingly. Still, they are usually unaware that there may probably be OR behind both situations. Often, what is on people's mouths is a simplification, an intuition behind the true concepts which exploit information and digital technologies. The news talks almost every day about some "new artificial intelligence" believed to do something magical. Algorithms have become so woven with our lives that people usually do not even ask how these work: they are satisfied with what algorithms do and what they can be applied for.

When one says "operations research", very few understand. This lack of visibility has long been a source of concern to OR professionals. [Pidd, 2001] described the two main paradoxes of this discipline: "*OR is neither young nor old and, despite its widespread use, the visibility of OR in the public eye is very limited*". [Power et al., 2018] reported that the concepts of OR and management science may have been replaced by "business analytics", used as a synonym. On "The Boston Globe", [Postrel, 2004] stated that OR "*is probably the most important field nobody's ever heard of*". Through the years, data availability and quality have been increasing. As [Bixby, 2020] said, "*At the highest levels of companies, there is an awareness of the importance of being able to understand data and use it to make the best possible business decisions (which is precisely what mathematical optimization does)*". And still, in December 2021, things seem not to have changed, according to [Rothberg, 2021], who wrote on Forbes that "*employers will be competing to hire statisticians and data scientists, but they'll also need to fill a* *role many have never heard of: operations research analyst"*.

What could be the main reasons for this issue? In [Raffaele, 2021], I tried to answer this question. [Brixius, 2015] wrote that OR is an indispensable tool of industry, but has never really connected with the technology community in the way it deserves, and that may be its fault. The interfaces we adopt to present OR may be too intricate; software developed only for specific and peculiar problems, not easy to use and general purpose. Recently, [Rothberg, 2020] agreed: OR applications are usually exploited to address highly-complex, large-scale business problems that are not as tangible to most people as other technologies such as machine learning. People may be scared away with too many technical details. [Lübbecke, 2015] suggested that we are *"too complicated"*. He reported the story of Randal Olson, an AI researcher who, starting from a procedure developed to compute the optimal search strategy for *Where is Waldo?*, implemented in 2015 his own genetic algorithm, in order to obtain an optimal road trip to visit each U.S. state and D.C. ([Olson, 2015]). Exactly as the original problem studied by [Dantzig et al., 1954]. But Olson's work was promoted in an article of [The Washington Post, 2015] that made appear an instance with 49 cities as very large to solve, while not mentioning a single word about the deep study of the Travelling Salesman Problem done in the last decades (e.g., see [Applegate et al., 2006]). Apparently, OR is not taken into account, but we know that it can have a relevant role in tackling several problems arising from industry or everyday life. We know that OR can empower companies to exploit their data to make better decisions (as in Section A.1). Also, it can provide young students with problem-solving and modelling skills, as well as encourage them to continue their studies or pursue a career in a STEM discipline (as in Section A.2).

A.3.2 Different contexts, different purposes, and different actions

In order to discuss what we can do to make OR more "visible", hereafter we evaluate some interpretations of this term by defining three possible contexts.

Industry Enhancing OR visibility to companies can mean promoting the application of OR techniques to real-world situations that need to be optimized. Indeed, OR can offer as powerful tools as machine learning and artificial intelligence, and these do not have to be alternative but rather synergical. In this case, maybe it is not so relevant that people understand what OR really is; it would be more effective to make companies understand what OR can actually do, opening CEOs' eyes to further applications. Using the OR label would not be required. Indeed, *"a rose by* *any other name would smell as sweet*" ([Shakespeare and Gibbons, 2002]). To this purpose, the absorption by other labels, as defined by [Pidd, 2001], could be acceptable, even if the main risk would be not to fully credit OR of its merits. OR practitioners employed in the industry would be the ideal communicators to expose to CEOs the potential achievements of OR. They would get their expertise widely taken up by managers. Here, one possible skill could be to be able to exploit OR interplay with other disciplines, adapting to the latest trend influencing the industry, in order to sell OR contents, implicitly or explicitly.

In the case study described in Section A.1, when discussing with the employees of Province of Brescia, we almost never used the OR label. Indeed, the most important aspect to them was not the learning of the OR methods and the techniques eventually applied. Their primary goal was the evaluation of their service. Thus, they cared more about the *interpretation of the results* obtained by relying on mixed-integer linear programming and Gurobi. In particular, they were more interested in the computation of the possible savings (in terms of kilometers travelled by couriers), and in getting practical suggestions to improve their implementation, no matter the OR or the statistical methods exploited. The most delicate parts in the interaction with the company were the understanding of the problem at the beginning and the discussion about the results in the end. Indeed, we had to solve any misunderstandings in terms of lexicon and terminology used, to correctly identify all the decision variables and to ensure to have included all the constraints. As a simple example, we noticed that the employees of the company used the word "*variable*" to refer to what actually was a parameter, a value that could change in different instances but that was anyway known. Regarding the presentation of the results, they appreciated the dynamic approach proposed, but they also right away stated that it was not very doable in reality, for the reasons explained in Subsection A.1.3. Thus, we found a few minor aspects to optimize. In these exchanges and dialogues, OR was exploited but nevertheless concealed.

Education The OR label should be explicitly adopted when introducing the discipline to students. In this case, fostering OR visibility means to uncover a specific branch of applied mathematics in order to increase students' motivation towards mathematics and all other STEM disciplines. Examples of activities to present OR are workshops, seminars, or competitions. When we present a topic to some students, we could provide them some preliminary details to grow their interest, not only theory and formulas, but maybe inspiring quotes or scenes from movies and books related to the topic we are going to present. Actually, not only students but also teachers should be considered as possible recipients, through, for instance, ad hoc training courses. Indeed, they may become the future communicators of OR, reaching a wider audience throughout the years, by presenting OR to their classrooms and, why not, to their colleagues too. Furthermore, beyond OR topics, historic OR figures can be presented such as George Dantzig, Delbert Ray Fulkerson, Ailsa H. Land and Alison Grant Harcourt, to encourage both girls and boys.

When carrying out the experimentation of the educational initiative reported in Section A.2, we always use the OR label. Also, we always start from examples and problems very close to students' everyday life[8]. For instance, to present the knapsack problem, we can consider a streaming service such as Netflix, a few TV series, and some spare time available. Then, we ask the students to decide which episodes to watch in order to minimize the time not used. Also, to introduce them to graph theory for the first time, we can rely on social media networks such as Facebook, Instagram, and TikTok in order to represent and play with their actual accounts and relationships. Only after that, we give them some basic notions about undirected and directed graphs, and we ask them to indicate other situations where graphs can be adopted. This is what we usually do when we present a new problem or method, i.e., we show students disparate applications and encourage them to find many others as well. In this way, we try at the same time to increase their awareness about OR and its connections to reality, and to disprove the idea that students may have about mathematics, which definitely does not only involve abstract and repeated calculations.

Government institutions Finally, as the third and last context, we consider government institutions as target audiences. Unfortunately, not always national or local OR organizations are well-known as international societies such as The OR Society or INFORMS. They should aim to be more visible, i.e., in terms of recognition by governments. In fact, being more recognized may lead to obtaining support and funding for research activities, as well as to be taken into account when governments make decisions. These can be improved by OR. Indeed, it can be applied to problems such as urban and air transportation, natural resources management, homeland security and safety risks, military systems, deployment of police, fire and emergency units, but also voting systems, and sports (for further details, see [Pollock, 1994]). Thus, also in this context the OR label should be adopted explicitly, without being absorbed by any other discipline. Ideal communicators may be, for instance, presidents and counsellors of associations and societies.

[8]`https://github.com/aliceraffaele/ROAR/`.

A.4 Conclusions

What we could do, in our own small way, is try to increase and improve our communication. When we write something and especially mathematics, we should have a specific reader in mind ([Steenrod et al., 1973]); the same when we communicate. To divulge OR, we should first understand our audience, and then choose our words properly, without sounding complex or out of reach, and also avoiding misunderstandings (e.g., the term "optimization" may have several meanings, based on the specific field of mathematics, to common sense, or other connotations).

We could and should try to simplify our arguments. [Goulet and Lamontagne, 2018] suggested adapting the language by summarizing the main concepts to transmit in a limited amount of words and making the text more understandable. But what is the right level of simplicity? [Scharrer et al., 2017] observed that laypeople can be hit by the *easiness effect* when reading easy texts. The easiness effect is the simplification of complex information characteristic of authentic popularized articles addressed to the general public. Despite being not trained or qualified in a particular subject, people can find persuasive the information provided. They can over confidently rely on their own judgement, maybe underestimating the real complexity of the topic. Thus, some cautiousness becomes necessary. Determining how much to simplify may depend both on the message we want to transmit and on the qualification and expertise of our target audience.

We should aim to make our topics more flexible. Among possible models of communication of science and technology, the contextual one acknowledges that people process information according to social and psychological schemas shaped by their previous experiences, cultural context, and personal circumstances ([Lewenstein, 2003]). Thus, their background, knowledge, and objectives are relevant. Also, nowadays we cannot forget the online platforms they visit. Perhaps social networks are the tool to bet on, to find a breakthrough in the visibility problem of OR. Easy to use and interact with, they can reach thousands of people all over the world in the time of a click. Companies and also policymakers and politicians exploit them to reach their customers and partners or their public. We could try to exploit more Twitter, where the OR community has a very active niche.

The side projects described in Section A.1 and Section A.2 are just two examples, two very small pieces of a puzzle our community of OR researchers could try to solve together, to show a clearer and larger picture of OR to other researchers and laypeople. According to the age and the identity of our listeners, we need to change our words and the tone of our voices, to deliver our main message. We are playing with their attention level, that is why we should move our eyes from the topics to these recipients. What would they like to hear? After all, operations research is polyhedric: there must be some facets (or some vertices) our target audience is curious to discover. Let's list them.

References

[Abel and Bicker, 1982] Abel, U. and Bicker, R. (1982). Determination of All Minimal Cut-Sets between a Vertex Pair in an Undirected Graph. *IEEE Transactions on Reliability*, R-31(2):167–171.

[Abu-khzam et al., 2005] Abu-khzam, F. N., Baldwin, N. E., Langston, M. A., and Samatova, N. F. (2005). On the relative efficiency of maximal clique enumeration algorithms, with application to high-throughput. In *Computational Biology, Proceedings, International Conference on Research Trends in Science and Technology*.

[Alstrup et al., 1997] Alstrup, S., Holm, J., de Lichtenberg, K., and Thorup, M. (1997). Minimizing diameters of dynamic trees. In Degano, P., Gorrieri, R., and Marchetti-Spaccamela, A., editors, *Automata, Languages and Programming*, pages 270–280, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Alstrup et al., 2005] Alstrup, S., Holm, J., Lichtenberg, K. D., and Thorup, M. (2005). Maintaining Information in Fully Dynamic Trees with Top Trees. *ACM Trans. Algorithms*, 1(2):243–264.

[Apaolaza et al., 2017] Apaolaza, I., San José-Eneriz, E., Tobalina, L., Miranda, E., Garate, L., Agirre, X., Prósper, F., and Planes, F. J. (2017). An in-silico approach to predict and exploit synthetic lethality in cancer metabolism. *Nature communications*, 8(1):1–9.

[Applegate et al., 2006] Applegate, D. L., Bixby, R. E., Chvatál, V., and Cook, W. J. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

[Aringhieri et al., 2003] Aringhieri, R., Hansen, P., and Malucelli, F. (2003). Chemical trees enumeration algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(1):67–83.

[Arunkumar and Lee, 1979] Arunkumar, S. and Lee, S. H. (1979). Enumeration of All Minimal Cut-Sets for a Node Pair in a Graph. *IEEE Transactions on Reliability*, R-28(1):51–55.

[Avis and Fukuda, 1992] Avis, D. and Fukuda, K. (1992). A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computational Geometry*, 8:295–313.

[Avis and Fukuda, 1996] Avis, D. and Fukuda, K. (1996). Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46. First International Colloquium on Graphs and Optimization.

[Avis and Jordan, 2018] Avis, D. and Jordan, C. (2018). mplrs: A scalable parallel vertex/facet enumeration code. *Mathematical Programming Computation*, 10(2):267–302.

[Avis and Jordan, 2021] Avis, D. and Jordan, C. (2021). mts: a light framework for parallelizing tree search codes. *Optimization Methods and Software*, 36(2-3):279–300.

[Avis and Roumanis, 2013] Avis, D. and Roumanis, G. (2013). A portable parallel implementation of the lrs vertex enumeration code. In *International Conference on Combinatorial Optimization and Applications*, pages 414–429. Springer.

[Bagan et al., 2007] Bagan, G., Durand, A., and Grandjean, E. (2007). On Acyclic Conjunctive Queries and Constant Delay Enumeration. In Duparc, J. and Henzinger, T. A., editors, *Computer Science Logic*, pages 208–222, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Baldwin et al., 2004] Baldwin, N. E., Collins, R. L., Langston, M. A., Symons, C. T., Leuze, M. R., and Voy, B. H. (2004). High performance computational tools for motif discovery. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, page 192. IEEE.

[Bellmore and Jensen, 1970] Bellmore, M. and Jensen, P. A. (1970). An Implicit Enumeration Scheme for Proper Cut Generation. *Technometrics*, 12(4):775–788.

[Berge, 1989] Berge, C. (1989). *Hypergraphs: Combinatorics of Finite Sets.*, volume 45 of *North-Holland mathematical library*. North-Holland.

[Bioch and Ibaraki, 1995] Bioch, J. and Ibaraki, T. (1995). Complexity of Identification and Dualization of Positive Boolean Functions. *Information and Computation*, 123(1):50–63.

[Birmelé et al., 2012] Birmelé, E., Ferreira, R., Grossi, R., Marino, A., Pisanti, N., Rizzi, R., and Sacomoto, G. (2012). Optimal listing of cycles and st-paths in undirected graphs. In Khanna, S., editor, *SODA 2012: the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1884–1896, New Orleans, LA, USA.

[Bixby, 2020] Bixby, R. (2020). *Mathematical Optimization: Past, Present and Future (Part 3)*. https://www.gurobi.com/resource/mathematical-optimization-past-present-and-future-part-3/. Accessed: 2021-12-29.

[Bläsius et al., 2022] Bläsius, T., Friedrich, T., Lischeid, J., Meeks, K., and Schirneck, M. (2022). Efficiently enumerating hitting sets of hypergraphs arising in data profiling. *Journal of Computer and System Sciences*, 124:192–213.

[Bondy and Murty, 1976] Bondy, J. A. and Murty, U. (1976). *Graph Theory with Applications*. Macmillan.

[Boros et al., 2004] Boros, E., Elbassioni, K., Gurvich, V., and Khachiyan, L. (2004). An Efficient Implementation of a Joint Generation Algorithm. In Ribeiro, C. and Martins, S., editors, *Experimental and Efficient Algorithms*, pages 114–128, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Boros et al., 2002] Boros, E., Elbassioni, K., Gurvich, V., Khachiyan, L., and Makino, K. (2002). Dual-Bounded Generating Problems: All Minimal Integer Solutions for a Monotone System of Linear Inequalities. *SIAM Journal on Computing*, 31:1624–1643.

[Boros et al., 2007] Boros, E., Elbassioni, K., Gurvich, V., and Makino, K. (2007). Generating vertices of polyhedra and related monotone generation problems. Technical report, DIMACS Technical Report 2007-03.

[Boros et al., 2009] Boros, E., Elbassioni, K., Gurvich, V., and Makino, K. (2009). Generating vertices of polyhedra and related problems of monotone generation. *Proceedings of the Centre de Recherches Mathématiques at the Université de Montréal, special issue on Polyhedral Computation (CRM Proceedings and Lecture Notes)*, 49:15–43.

[Boros and Makino, 2009] Boros, E. and Makino, K. (2009). A Fast and Simple Parallel Algorithm for the Monotone Duality Problem. In Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., and Thomas, W., editors, *Automata, Languages and Programming*, pages 183–194, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Boucher, 1997] Boucher, V. (1997). *Interlibrary loan practices handbook*. American Library Association.

[Brixius, 2015] Brixius, N. (2015). *Hole Hawg: OR's PR Problem*. https://nathanbrixius.wordpress.com/2015/03/19/hole-hawg-ors-pr-problem/. Accessed: 2021-12-29.

[Bron and Kerbosch, 1973] Bron, C. and Kerbosch, J. (1973). Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577.

[Brüngger et al., 1999] Brüngger, A., Marzetta, A., Fukuda, K., and Nievergelt, J. (1999). The parallel search bench zram and its applications. *Annals of Operations Research*, 90:45–63.

[Bussieck and Lübbecke, 1998] Bussieck, M. R. and Lübbecke, M. E. (1998). The vertex set of a 01-polytope is strongly p-enumerable. *Computational Geometry*, 11(2):103–109.

[Calvino, 1988] Calvino, I. (1988). *Six memos for the next millennium*. Harvard University Press.

[Capelli and Strozecki, 2017] Capelli, F. and Strozecki, Y. (2017). On The Complexity of Enumeration.

[Carmeli and Kröll, 2019] Carmeli, N. and Kröll, M. (2019). On the enumeration complexity of unions of conjunctive queries. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '19, page 134–148, New York, NY, USA. Association for Computing Machinery.

[Cheliyan and Bhattacharyya, 2018] Cheliyan, A. and Bhattacharyya, S. (2018). Fuzzy fault tree analysis of oil and gas leakage in subsea production systems. *Journal of Ocean Engineering and Science*, 3(1):38–48.

[Colajanni et al., 2022] Colajanni, G., Gobbi, A., Picchi, M., Raffaele, A., and Taranto, E. (2022). An OR-based Teaching Unit for Grade 10: The ROAR Experience, Part I. *INFORMS Transactions on Education*. To appear.

[Colbourn, 1987] Colbourn, C. J. (1987). *The Combinatorics of Network Reliability*. Oxford University Press, Inc., USA.

[Compagnucci and Spigarelli, 2020] Compagnucci, L. and Spigarelli, F. (2020). The Third Mission of the university: A systematic literature review on potentials and constraints. *Technological Forecasting and Social Change*, 161:120284.

[Conte, 2018] Conte, A. (2018). Enumeration algorithms for real-world networks: efficiency and beyond. *PhD Thesis*.

[Conte et al., 2020] Conte, A., Crescenzi, P., Marino, A., and Punzi, G. (2020). Enumeration of sd separators in dags with application to reliability analysis in temporal graphs. In *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[Conte et al., 2018] Conte, A., Grossi, R., Marino, A., Rizzi, R., Uno, T., and Versari, L. (2018). Tight Lower Bounds for the Number of Inclusion-Minimal st-Cuts. In *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, pages 100–110.

[Conte et al., 2016] Conte, A., Grossi, R., Marino, A., and Versari, L. (2016). Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques. In Ioannis Chatzigiannakis Michael Mitzenmacher, Y. R. and Sangiorgi, D., editors, *ICALP 2016: the 43rd International Colloquium on Automata, Languages, and Programming*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 148:1—-148:15, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Conte et al., 2021] Conte, A., Kanté, M. M., Uno, T., and Wasa, K. (2021). Maximal strongly connected cliques in directed graphs: Algorithms and bounds. *Discrete Applied Mathematics*, 303:237–252. Combined Special Issue: 1) 17th Cologne–Twente Workshop on Graphs and Combinatorial Optimization (CTW 2019); Guest edited by Johann Hurink, Bodo Manthey 2) WEPA 2018 (Second Workshop on Enumeration Problems and Applications); Guest edited by Takeaki Uno, Andrea Marino.

[Conte and Uno, 2019] Conte, A. and Uno, T. (2019). New polynomial delay bounds for maximal subgraph enumeration by proximity search. In Charikar, M. and Cohen, E., editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1179–1190. ACM.

[Cordone et al., 2005] Cordone, R., Ferrarini, L., and Piroddi, L. (2005). Enumeration algorithms for minimal siphons in petri nets based on place constraints. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 35(6):844–854.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

[Danielson, 1968] Danielson, G. (1968). On finding the simple paths and circuits in a graph. *IEEE Transactions on Circuit Theory*, 15(3):294–295.

[Dantzig et al., 1954] Dantzig, G. B., Fulkerson, D. R., and Johnson, S. M. (1954). *Solution of a large-scale Traveling-Salesman Problem*. *Operations Research*, 2:393–410.

[Debieux et al., 2017] Debieux, V., Pignolet, Y., and Sivanthi, T. (2017). Faster Exact Reliability Computation. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 121–124.

[Dekker et al., 2012] Dekker, R., Bloemhof, J., and Mallidis, I. (2012). Operations research for green logistics – an overview of aspects, issues, contributions and challenges. *European Journal of Operational Research*, 219(3):671–679. Feature Clusters.

[Deo, 2017] Deo, N. (2017). *Graph Theory with Applications to Engineering and Computer Science*. Dover Publications.

[Desaulniers et al., 2002] Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M. M., and Soumis, F. (2002). VRP with Pickup and Delivery. *The vehicle routing problem*, 9:225–242.

[Diestel, 2017] Diestel, R. (2017). *Graph Theory*. Springer Publishing Company, Incorporated, 5th edition.

[Duarte et al., 2021] Duarte, G. L., Eto, H., Hanaka, T., Kobayashi, Y., Kobayashi, Y., Lokshtanov, D., Pedrosa, L. L. C., Schouery, R. C. S., and Souza, U. S. (2021). Computing the Largest Bond and the Maximum Connected Cut of a Graph. *Algorithmica*, 83(5):1421–1458.

[Edmonds and Fulkerson, 1970] Edmonds, J. and Fulkerson, D. (1970). Bottleneck extrema. *Journal of Combinatorial Theory*, 8(3):299–306.

[Eiter and Gottlob, 1991] Eiter, T. and Gottlob, G. (1991). Identifying the minimal transversals of a hypergraph and related problems. *CD-TR 91/16*.

[Eiter and Gottlob, 1995] Eiter, T. and Gottlob, G. (1995). Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304.

[Eiter and Gottlob, 2000] Eiter, T. and Gottlob, G. (2000). Identifying The Minimal Transversals Of A Hypergraph And Related Problems. *SIAM Journal on Computing*, 31.

[Eiter et al., 2003] Eiter, T., Gottlob, G., and Makino, K. (2003). New Results on Monotone Dualization and Generating Hypergraph Transversals. *SIAM Journal on Computing*, 32(2):514–537.

[Eiter et al., 2008] Eiter, T., Makino, K., and Gottlob, G. (2008). Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049. In Memory of Leonid Khachiyan (1952 - 2005).

[Elbassioni, 2008] Elbassioni, K. (2008). On the complexity of monotone dualization and generating minimal hypergraph transversals. *Discrete Applied Mathematics*, 156(11):2109–2123.

[Eppstein, 1998] Eppstein, D. (1998). Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673.

[Eppstein et al., 2010] Eppstein, D., Löffler, M., and Strash, D. (2010). Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In Cheong, O., Chwa, K.-Y., and Park, K., editors, *Algorithms and Computation*, pages 403–414, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Eppstein and Strash, 2011] Eppstein, D. and Strash, D. (2011). Listing All Maximal Cliques in Large Sparse Real-World Graphs. In Pardalos, P. M. and Rebennack, S., editors, *Experimental Algorithms*, pages 364–375, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Eriksson et al., 2006] Eriksson, A. P., Barr, O., and Astrom, K. (2006). Image segmentation using minimal graph cuts. In *SSBA Symposium on Image Analysis*.

[Ferreira et al., 2011] Ferreira, R., Grossi, R., and Rizzi, R. (2011). Output-sensitive listing of bounded-size trees in undirected graphs. In Demetrescu, C. and Halldórsson, M. M., editors, *ESA 2011: the 19th Annual European Symposium on Algorithms*, volume 6942 of *Lecture Notes in Computer Science*, pages 275–286. Springer Berlin Heidelberg.

[Ferreira et al., 2014] Ferreira, R., Grossi, R., Rizzi, R., Sacomoto, G., and Marie-France, S. (2014). Amortized $\widetilde{O}(|V|)$-Delay Algorithm for Listing Chordless Cycles in Undirected Graphs. In Schulz, A. S. and Wagner, D., editors, *ESA 2014:*

the 22th Annual European Symposium on Algorithms, volume 8737 of *Lecture Notes in Computer Science*, pages 418–429, Wroclaw, Poland. Springer Berlin Heidelberg.

[Floyd, 1967] Floyd, R. W. (1967). Nondeterministic algorithms. *J. ACM*, 14(4):636–644.

[Ford and Fulkerson, 2015] Ford, L. R. and Fulkerson, D. R. (2015). *Flows in networks*. Princeton University Press.

[Frederickson, 1997] Frederickson, G. N. (1997). Ambivalent Data Structures for Dynamic 2-Edge-Connectivity and k Smallest Spanning Trees. *SIAM Journal on Computing*, 26(2):484–538.

[Fredman and Khachiyan, 1996] Fredman, M. L. and Khachiyan, L. (1996). On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *Journal of Algorithms*, 21(3):618–628.

[Fukuda and Prodon, 1995] Fukuda, K. and Prodon, A. (1995). Double description method revisited. In *Franco-Japanese and Franco-Chinese Conference on Combinatorics and Computer Science*, pages 91–111. Springer.

[Gabow and Myers, 1978] Gabow, H. N. and Myers, E. W. (1978). Finding all spanning trees of directed and undirected graphs. *SIAM Journal on Computing*, 7(3):280–287.

[Gaur and Krishnamurti, 2004] Gaur, D. and Krishnamurti, R. (2004). Average Case Self-Duality of Monotone Boolean Functions. In Tawfik, A. and Goodwin, S., editors, *Advances in Artificial Intelligence*, pages 322–338, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Gaur et al., 2021] Gaur, V., Yadav, O. P., Soni, G., and Rathore, A. P. S. (2021). A literature review on network reliability analysis and its engineering applications. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 235(2):167–181.

[Gerstl et al., 2016] Gerstl, M. P., Klamt, S., Jungreuthmayer, C., and Zanghellini, J. (2016). Exact quantification of cellular robustness in genome-scale metabolic networks. *Bioinformatics*, 32(5):730–737.

[Gianinazzi et al., 2021] Gianinazzi, L., Besta, M., Schaffner, Y., and Hoefler, T. (2021). Parallel algorithms for finding large cliques in sparse graphs. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, page 243–253, New York, NY, USA. Association for Computing Machinery.

[Ginzburg, 2017] Ginzburg, N. (2017). *The Little Virtues: Essays*. Simon and Schuster.

[Goulet and Lamontagne, 2018] Goulet, C. and Lamontagne, M. (2018). To Reach a Wider Audience, Simplify Your Science. *Seismological Research Letters*, 89(2A):677–677.

[Grossi, 2016] Grossi, R. (2016). Enumeration of Paths, Cycles, and Spanning Trees. In Kao, M.-Y., editor, *Encyclopedia of Algorithms*, pages 640–645. Springer New York, New York, NY.

[Gunopulos et al., 1997] Gunopulos, D., Mannila, H., Khardon, R., and Toivonen, H. (1997). Data Mining, Hypergraph Transversals, and Machine Learning. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '97, pages 209–216, New York, NY, USA. Association for Computing Machinery.

[Gurobi Optimization, nd] Gurobi Optimization (n.d.). Gurobi Optimizer. https://www.gurobi.com. Accessed: 2021-12-29.

[Hagen, 2008] Hagen, M. (2008). *Algorithmic and Computational Complexity Issues of MONET*. Cuvillier Verlag.

[Hagen et al., 2009] Hagen, M., Horatschek, P., and Mundhenk, M. (2009). Experimental Comparison of the Two Fredman-Khachiyan-Algorithms. In *Proceedings of the Meeting on Algorithm Engineering and Experiments*, pages 154–161, USA. Society for Industrial and Applied Mathematics.

[Harary, 1969] Harary, F. (1969). *Graph Theory*. Addison-Wesley, Reading, MA.

[Henzinger and Fredman, 1998] Henzinger, M. R. and Fredman, M. L. (1998). Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22(3):351–362.

[Henzinger and King, 1999] Henzinger, M. R. and King, V. (1999). Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM (JACM)*, 46(4):502–516.

[Holm et al., 2001] Holm, J., de Lichtenberg, K., and Thorup, M. (2001). Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity. *J. ACM*, 48(4):723–760.

[Hopcroft and Tarjan, 1973] Hopcroft, J. E. and Tarjan, R. E. (1973). Algorithm 447: Efficient Algorithms for Graph Manipulation. *Commun. ACM*, 16(6):372–378.

[Hädicke and Klamt, 2010] Hädicke, O. and Klamt, S. (2010). Computing complex metabolic intervention strategies using constrained minimal cut sets. *Metabolic engineering*, 13:204–13.

[IBM, nd] IBM (n.d.). ILOG CPLEX Optimization Studio. https://www.ibm.com/products/ilog-cplex-optimization-studio?lnk=STW_US_STESCH&lnk2=trial_ILOGOptStudio&pexp=def&psrc=none&mhsrc=ibmsearch_a&mhq=cplex. Accessed: 2021-12-29.

[INFORMS, 2020] INFORMS (2020). *Information on COVID-19 and Pandemics*. https://www.informs.org/Impact/O.R.-Analytics-for-Policymakers/COVID-19. Accessed: 2021-12-29.

[Jensen and Bellmore, 1969] Jensen, P. A. and Bellmore, M. (1969). An algorithm to determine the reliability of a complex system. *IEEE Trans. Rehahty R-I 4*, pages 169–174.

[Johnson, 1975] Johnson, D. B. (1975). Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84.

[Johnson et al., 1988] Johnson, D. S., Yannakakis, M., and Papadimitriou, C. H. (1988). On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123.

[Jordan et al., 2017] Jordan, C., Joswig, M., and Kastner, L. (2017). Parallel enumeration of triangulations. *arXiv preprint arXiv:1709.04746*.

[Jung et al., 2020] Jung, S., Yoo, J., and Lee, Y.-J. (2020). A software fault tree analysis technique for formal requirement specifications of nuclear reactor protection systems. *Reliability Engineering & System Safety*, 203:107064.

[Kao, 2016] Kao, M.-Y., editor (2016). *Encyclopedia of Algorithms*. Springer, New York, 2nd edition.

[Kavvadias and Stavropoulos, 2003] Kavvadias, D. J. and Stavropoulos, E. C. (2003). Monotone Boolean dualization is in co-NP$[log^2 n]$. *Information Processing Letters*, 85(1):1–6.

[Khachiyan et al., 2009] Khachiyan, L., Boros, E., Borys, K., Gurvich, V., and Elbassioni, K. (2009). Generating all vertices of a polyhedron is hard. In *Twentieth Anniversary Volume:*, pages 1–17. Springer.

[Khachiyan et al., 2006] Khachiyan, L., Boros, E., Elbassioni, K., and Gurvich, V. (2006). An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals and its application in joint generation. *Discrete Applied Mathematics*, 154(16):2350–2372.

[Khachiyan et al., 2005] Khachiyan, L., Boros, E., Elbassioni, K., Gurvich, V., and Makino, K. (2005). On the Complexity of Some Enumeration Problems for Matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984.

[Kiyomi, 2016] Kiyomi, M. (2016). Reverse Search; Enumeration Algorithms. In Kao, M.-Y., editor, *Encyclopedia of Algorithms*, pages 1840–1842. Springer New York, New York, NY.

[Klamt and Gilles, 2004] Klamt, S. and Gilles, E. D. (2004). Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20 2:226–34.

[Kurita et al., 2021a] Kurita, K., Wasa, K., Arimura, H., and Uno, T. (2021a). Efficient enumeration of dominating sets for sparse graphs. *Discrete Applied Mathematics*, 303:283–295. Combined Special Issue: 1) 17th Cologne–Twente Workshop on Graphs and Combinatorial Optimization (CTW 2019); Guest edited by Johann Hurink, Bodo Manthey 2) WEPA 2018 (Second Workshop on Enumeration Problems and Applications); Guest edited by Takeaki Uno, Andrea Marino.

[Kurita et al., 2021b] Kurita, K., Wasa, K., Uno, T., and Arimura, H. (2021b). A constant amortized time enumeration algorithm for independent sets in graphs with bounded clique number. *Theoretical Computer Science*, 874:32–41.

[Lehman, 1964] Lehman, A. (1964). A Solution of the Shannon Switching Game. *Journal of the Society for Industrial and Applied Mathematics*, 12(4):687–725.

[Lehman, 1979] Lehman, A. (1979). On the width—length inequality. *Mathematical Programming*, 17:403–417.

[Lewenstein, 2003] Lewenstein, B. V. (2003). *Models of public communication of science and technology*.

[Lin et al., 2003] Lin, H.-Y., Kuo, S.-Y., and Yeh, F.-M. (2003). Minimal cutset enumeration and network reliability evaluation by recursive merge and BDD. In *Proceedings of the Eighth IEEE Symposium on Computers and Communications. ISCC 2003*, pages 1341–1346 vol.2.

[Liu and Edelberg, 1968] Liu, C. L. and Edelberg, M. (1968). *Introduction to Combinatorial Mathematics*. Computer science series. McGraw-Hill.

[Liu et al., 2015] Liu, F., Vilaça, P., Rocha, I., and Rocha, M. (2015). Development and application of efficient pathway enumeration algorithms for metabolic engineering applications. *Computer Methods and Programs in Biomedicine*, 118(2):134–146.

[Lovász, 1992] Lovász, L. (1992). *Combinatorial optimization: some problems and trends*. DIMACS, Center for Discrete Mathematics and Theoretical Computer Science.

[Lübbecke, 2015] Lübbecke, M. (2015). *Are we too complicated?* https://mluebbecke.wordpress.com/2015/03/23/are-we-too-complicated/. Accessed: 2021-12-29.

[Marino, 2015] Marino, A. (2015). *Enumeration Algorithms*, pages 13–35. Atlantis Press, Paris.

[Martelli, 1976] Martelli, A. (1976). A Gaussian Elimination Algorithm for the Enumeration of Cut Sets in a Graph. *Journal of the ACM*, 23(1):58–73.

[Marzetta, 1998] Marzetta, A. (1998). *ZRAM: A library of parallel search algorithms and its use in enumeration and combinatorial optimization*. Citeseer.

[Miltersen et al., 1994] Miltersen, P. B., Subramanian, S., Vitter, J. S., and Tamassia, R. (1994). Complexity models for incremental computation. *Theoretical Computer Science*, 130(1):203 – 236.

[Modani and Dey, 2009] Modani, N. and Dey, K. (2009). Large maximal cliques enumeration in large sparse graphs. In *COMAD*. Citeseer.

[Montané and Galvão, 2002] Montané, F. A. T. and Galvão, R. D. (2002). Vehicle routing problems with simultaneous pick-up and delivery service. *Opsearch*, 39(1):19–33.

[Motzkin et al., 1953] Motzkin, T. S., Raiffa, H., Thompson, G. L., and Thrall, R. M. (1953). The double description method. *Contributions to the Theory of Games*, 2(28):51–73.

[Mourad et al., 2019] Mourad, A., Puchinger, J., and Chu, C. (2019). A survey of models and algorithms for optimizing shared mobility. *Transportation Research Part B: Methodological*, 123:323–346.

[Nguyen and Lin, 2021] Nguyen, T.-P. and Lin, Y.-K. (2021). Reliability assessment of a stochastic air transport network with late arrivals. *Computers & Industrial Engineering*, 151:106956.

[Niss, 1992] Niss, M. (1992). *Applications and modelling in school Mathematics-directions for future development*, volume 3. Development in School Mathematics Education Around the World.

[Niu et al., 2017] Niu, Y.-F., Gao, Z.-Y., and Lam, W. H. (2017). Evaluating the reliability of a stochastic distribution network in terms of minimal cuts. *Transportation Research Part E: Logistics and Transportation Review*, 100:75–97.

[Olson, 2015] Olson, R. (2015). *Computing the optimal road trip across the U.S.* http://www.randalolson.com/2015/03/08/computing-the-optimal-road-trip-across-the-u-s/. Accessed: 2021-12-29.

[Pan and Santos, 2008] Pan, L. and Santos, E. E. (2008). An anytime-anywhere approach for maximal clique enumeration in social network analysis. In *2008 IEEE International Conference on Systems, Man and Cybernetics*, pages 3529–3535.

[Papadimitriou and Yannakakis, 1984] Papadimitriou, C. and Yannakakis, M. (1984). The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259.

[Peng et al., 2013] Peng, B., Zhang, L., and Zhang, D. (2013). A survey of graph theoretical approaches to image segmentation. *Pattern Recognition*, 46(3):1020–1038.

[Pidd, 2001] Pidd, M. (2001). *The future of OR*. *Journal of the Operational Research Society*, 52:1181–1190.

[Pollock, 1994] Pollock, S. M., editor (1994). *Operations Research and the Public Sector, Volume 6 – 1st Edition*. Handbooks in Operations Research and Management Science. Elsevier Science.

[Postrel, 2004] Postrel, V. (2004). *Operation everything – It stocks your grocery store, schedules your favorite team's games, and helps plan your vacation. A primer on the most influential academic discipline you've never heard of*. http://archive.boston.com/news/globe/ideas/articles/2004/06/27/operation_everything?pg=full. Accessed: 2021-12-29.

[Power et al., 2018] Power, D. J., Heavin, C., McDermott, J., and Daly, M. (2018). *Defining business analytics: an empirical approach*. *Journal of Business Analytics*, 1(1):40–53.

[Raffaele, 2021] Raffaele, A. (2021). Becoming Visible: Why We Should be Better Communicators Now. *Operations Research Forum*, 2(1):7.

[Raffaele and Gobbi, 2021] Raffaele, A. and Gobbi, A. (2021). Teaching Operations Research Before University: A Focus on Grades 9–12. *Operations Research Forum*, 2(1):13.

[Raffaele et al., 2022] Raffaele, A., Gussago, M., Zavatteri, M., Bazzoli, F., and Rizzi, R. (2022). Analysis, evaluation, and improvement of the routing of an interlibrary loan service: the industrial case study of Province of Brescia. *Working paper*.

[Raffaele and Rizzi, 2021] Raffaele, A. and Rizzi, R. (2021). A new decomposition for the Monotone Boolean Duality problem. *July 2021*. Submitted.

[Raffaele et al., 2021] Raffaele, A., Rizzi, R., and Uno, T. (2021). Listing the bonds of a graph in $\widetilde{O}(n)$-delay. *June 2021*. Submitted.

[Rains and Sloane, 1999] Rains, E. M. and Sloane, N. J. A. (1999). On Cayley's Enumeration of Alkanes (or 4-Valent Trees). *Journal of Integer Sequences*, 2:11.

[Rausand, 2014] Rausand, M. (2014). *Reliability of Safety-Critical Systems: Theory and Applications*. Wiley.

[Ravikrishnan et al., 2018] Ravikrishnan, A., Nasre, M., and Raman, K. (2018). Enumerating all possible biosynthetic pathways in metabolic networks. *Scientific Reports*, 8(1):9932.

[Read and Tarjan, 1975] Read, R. C. and Tarjan, R. E. (1975). Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252.

[Rothberg, 2020] Rothberg, E. (2020). *Imagine a World without Mathematical Optimization*. https://www.gurobi.com/resource/imagine-a-world-without-mathematical-optimization/. Accessed: 2021-12-29.

[Rothberg, 2021] Rothberg, E. (2021). *Operations Research Analyst: The Fastest-Growing Job You've Never Heard Of*. https://www.forbes.com/sites/forbestechcouncil/2021/12/20/operations-research-analyst-the-fastest-growing-job-youve-never-heard-of/. Accessed: 2021-12-23.

[Salhi et al., 2014] Salhi, S., Imran, A., and Wassan, N. A. (2014). The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. *Computers and Operations Research*, 52(Part B):315–325.

[Schaefer, 1978] Schaefer, T. J. (1978). The complexity of satisfiability problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978)*, pages 216–226. ACM, New York.

[Scharrer et al., 2017] Scharrer, L., Rupieper, Y., Stadtler, M., and Bromme, R. (2017). When science becomes too easy: Science popularization inclines laypeople to underrate their dependence on experts. *Public Understanding of Science*, 26(8):1003–1018.

[Sedaghat et al., 2018] Sedaghat, N., Stephen, T., and Chindelevitch, L. (2018). Speeding up Dualization in the Fredman-Khachiyan Algorithm B. In D'Angelo, G., editor, *17th International Symposium on Experimental Algorithms (SEA 2018)*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:13, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Seymour, 1976] Seymour, P. (1976). The Forbidden Minors of Binary Clutters. *Journal of the London Mathematical Society*, s2-12(3):356–360.

[Shakespeare and Gibbons, 2002] Shakespeare, W. and Gibbons, B. (2002). *Romeo and Juliet*. Arden edition of the works of William Shakespeare. Arden Shakespeare.

[Shen, 2020] Shen, S. (2020). *From Data to Actions, From Observations to Solutions. A Summary of Operations Research and Industrial Engineering Tools for Fighting COVID-19*. `http://www-personal.umich.edu/~siqian/docs/or-ie-fighting-covid19_v1.pdf`. Accessed: 2021-12-29.

[Shier, 1991] Shier, D. R. (1991). *Network Reliability and Algebraic Structures*. Clarendon Press, USA.

[Shioura et al., 1997] Shioura, A., Tamura, A., and Uno, T. (1997). An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM Journal on Computing*, 26(3):678–692.

[Shmoys, 1995] Shmoys, D. B. (1995). Computing near-optimal solutions to combinatorial optimization problems. *Combinatorial Optimization*, 20:355–397.

[Sleator and Tarjan, 1983] Sleator, D. D. and Tarjan, R. E. (1983). A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.*, 26:362–391.

[Steenrod et al., 1973] Steenrod, N. E., Halmos, P. R., Schiffer, M. M., and Dieudonne, J. A. (1973). *How to Write Mathematics*. American Mathematical Society.

[Tamaki, 2000] Tamaki, H. (2000). Space-efficient enumeration of minimal transversals of a hypergraph. In *IPSJ-AL*, volume 75, pages 29–36.

[Taranto et al., 2022] Taranto, E., Colajanni, G., Gobbi, A., Picchi, M., and Raffaele, A. (2022). Fostering students' development of modelling and problem-solving skills through Operations Research and digital technologies in a collaborative environment. *November 2021*. Submitted.

[Tarjan, 1971] Tarjan, R. (1971). Depth-first search and linear graph algorithms. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 114–121.

[Tarjan, 1985] Tarjan, R. E. (1985). Amortized computational complexity. *SIAM Journal on Algebraic Discrete Methods*, 6(2):306–318.

[The OR Society, 2020] The OR Society (2020). *Coronavirus (COVID-19)*. https://www.theorsociety.com/get-involved/coronavirus-covid-19/. Accessed: 2021-12-29.

[The Washington Post, 2015] The Washington Post (2015). *A data genius computes the ultimate American road trip*. https://www.washingtonpost.com/news/wonk/wp/2015/03/10/a-data-genius-computes-the-ultimate-american-road-trip/. Accessed: 2021-12-29.

[Thorup, 2000] Thorup, M. (2000). Near-Optimal Fully-Dynamic Graph Connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, page 343–350, New York, NY, USA. Association for Computing Machinery.

[Tomita et al., 2019] Tomita, E., Mitsutake, A., and Nozaki, T. (2019). Enumeration of maximum cliques and its application to coding theory. In Boros, E., Kimelfeld, B., Pichler, R., and Schweikardt, N., editors, *Enumeration in Data Management (Dagstuhl Seminar 19211)*, volume 9, pages 89–109, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Tsukiyama et al., 1980] Tsukiyama, S., Shirakawa, I., Ozaki, H., and Ariyoshi, H. (1980). An Algorithm to Enumerate All Cutsets of a Graph in Linear Time per Cutset. *J. ACM*, 27:619–632.

[Tutte, 1966] Tutte, W. T. (1966). *Connectivity in graphs*. Mathematical expositions. University of Toronto Press.

[Uno, 2016] Uno, T. (2016). Amortized Analysis on Enumeration Algorithms. In Kao, M.-Y., editor, *Encyclopedia of Algorithms*, pages 72–76. Springer New York, New York, NY.

[Van Lierde and Chow, 2019] Van Lierde, H. and Chow, T. W. (2019). Query-oriented text summarization based on hypergraph transversals. *Information Processing & Management*, 56(4):1317–1338.

[Van Slyke and Frank, 1971] Van Slyke, R. M. and Frank, H. (1971). Network reliability analysis: Part I. *Networks*, 1(3):279–290.

[Veyrat-Charvillon et al., 2013] Veyrat-Charvillon, N., Gérard, B., Renauld, M., and Standaert, F.-X. (2013). An optimal key enumeration algorithm and its application to side-channel attacks. In Knudsen, L. R. and Wu, H., editors, *Selected Areas in Cryptography*, pages 390–406, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Wang et al., 2021] Wang, Z., Hu, W., Chen, G., Yuan, C., Gu, R., and Huang, Y. (2021). Towards efficient distributed subgraph enumeration via backtracking-based framework. *IEEE Transactions on Parallel and Distributed Systems*, 32(12):2953–2969.

[Wasa, 2019] Wasa, K. (2019). Enumeration of Enumeration Algorithms and Its Complexity. `https://kunihirowasa.github.io/enum`. Accessed: 2021-12-29.

[Weibel, 2010] Weibel, C. (2010). Implementation and parallelization of a reverse-search algorithm for minkowski sums. In *2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 34–42. SIAM.

[Wilson, 1972] Wilson, R. (1972). Introduction to graph theory, oliver and boyd.

[Wulff-Nilsen, 2013] Wulff-Nilsen, C. (2013). Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, page 1757–1769, USA. Society for Industrial and Applied Mathematics.

[Yan et al., 1994] Yan, L., Taha, H. A., and Landers, T. L. (1994). A recursive approach for enumerating minimal cutsets in a network. *IEEE Transactions on Reliability*, 43(3):383–388.

[Ziegler, 2012] Ziegler, G. M. (2012). *Lectures on polytopes*, volume 152. Springer Science & Business Media.