OMNITIG LISTING AND CONTIG ASSEMBLY FOR GENOMIC DE BRUIJN GRAPHS

Author: Elia Carlo Zirondelli
Advisor: Prof. Romeo Rizzi

Doctoral Thesis
Ph.D. course in Mathematics
University of Trento, Department of Mathematics
University of Verona, Department of Computer Science
December 2021

Contents

Introduction

The present dissertation offers technical advances in the field of genome assembly, related to the problem of efficiently finding all safe solutions admitted by the problem modelizations, and a novel framework, for obtaining safe and complete algorithms, also allowing for easy characterization of both old and new problems.

The thesis comprises the following two articles:

- Massimo Cairo, Romeo Rizzi, Alexandru I. Tomescu, Elia C. Zirondelli: Genome Assembly, from Practice to Theory: Safe, Complete and Linear-Time (full version at https://arxiv.org/abs/2002.10498). In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of LIPIcs, pages 43:1–43:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

- Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian S. Schmidt, Alexandru I. Tomescu, Elia C. Zirondelli: The Hydrostructure: a Universal Framework for Safe and Complete Algorithms for Genome Assembly. CoRR abs/2011.12635 (2020) (preprint available at https://arxiv.org/abs/2011.12635).

The contents of the first article appear in Chapter 2. The $O(n)$ bound of Theorem 2.25 (from $O(m)$ initially) is due to Sebastian Schmidt.

The contents of the latter article appear in Chapter 3. I had major technical contributions both on the discovery and initial study of the Hydrostructure, that reached its final form with Definition 3.2 and in Section 3.3. The writing of the paper mentioned above is mostly due to my co-author Sebastian Schmidt. To conclude, we chose to omit from this thesis sections where I did not have any relevant technical contributions (check the paper mentioned above to have the full version of the results). However, for the sake of completeness, I included *Implementation* subsections at the end of each main sections, where my co-author Sebastian Schmidt had major contributions.

The first chapter of the thesis, Chapter 1, comprises the introductions of the corresponding two articles to which I contributed in a minor way. In general, my contributions in the article were related to the mathematical and algorithmic results; however, for the sake of completeness, I also include, in Chapter 1, the description behind the bioinformatics motivation (Section 1.0.1)

and the subsection *Limitations of the existing theory of safe and complete algorithms* (and Figure 1.2), parts to which I did not contribute in a major way.

Genome assembly asks to reconstruct an unknown string from many shorter substrings of it. Its hardness stems both from practical issues (size and errors of real data), and from the fact that problem formulations inherently admit multiple solutions. Given these, at their core, most state-of-the-art assemblers are based on finding non-branching paths (*unitigs*) in an assembly graph. While such paths constitute only partial assemblies, they are likely to be correct. More precisely, if one defines a genome assembly solution as a *closed arc-covering walk* of the graph, then unitigs appear in all solutions, being thus *safe* partial solutions. Until recently, it was open what are *all* the safe walks of an assembly graph. Tomescu and Medvedev (RECOMB 2016) characterized all such safe walks (*omnitigs*), thus giving the first safe and *complete* genome assembly algorithm. Even though omnitig finding was later improved to quadratic time, it remained open whether the crucial linear-time feature of finding unitigs can be attained with omnitigs.

We answer this question affirmatively: in Chapter 2, it is described a surprising $O(m)$-time algorithm to *identify* all maximal omnitigs of a graph with n nodes and m arcs, notwithstanding the existence of families of graphs with $\Theta(mn)$ total maximal omnitig size. These results have been presented at the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021). The main result is based on the discovery of a family of walks (*macrotigs*) with the property that all the non-trivial omnitigs are univocal extensions of subwalks of a macrotig. This has two consequences: (1) A *linear-time output-sensitive* algorithm enumerating all maximal omnitigs. (2) A *compact* $O(m)$ *representation* of all maximal omnitigs, which allows, e.g., for $O(m)$-time computation of various statistics on them. Our results close a long-standing theoretical question inspired by practical genome assemblers, originating with the use of unitigs in 1995. We envision our results to be at the core of a reverse transfer from theory to practical and *complete* genome assembly programs, as has been the case for other key Bioinformatics problems.

However, all results, from the very first safe and complete genome assembly algorithm [66] to our most recent result described in Chapter 2, typically used very specific approaches, which did not generalize, and as a consequence could not handle practical issues. As such, one of the problems that remained open was whether one could be *complete* also for models of genome assembly of more practical applicability. Moreover, despite previous results presenting *optimal* algorithms, they were based on avoiding *forbidden structures*, and hence it was open whether there would exist any simple characterization to complete the understanding the problem structure.

We answered these questions in Chapter 3, where we present a *universal framework* for obtaining safe and complete algorithms unifying the previous results, while also allowing for easy generalizations to other assembly problems incorporating many practical aspects. These results

have been obtained in cooperation with part of the Graph Algorithms team of the Algorithmic Bioinformatics group at the Department of Computer Science, University of Helsinki, under the supervision of professor Alexandru Tomescu.

This framework is based on an entirely new perspective for studying safety, and on a novel graph structure (the *hydrostructure* of a walk) highlighting the reachability properties of the graph from the perspective of the walk. The hydrostructure, indeed, allows for simple characterizations of the existing and of new models for safe walks. Moreover, the hydrostructure serves as a simple YES-certificate for *all* the studied models. Almost all of our characterizations are directly adaptable to *optimal* verification algorithms, and *simple* enumeration algorithms. Most of these enumeration algorithms are also improved to optimality using an incremental computation procedure (see section *Incremental Computation of the Hydrostructure* in [13]) and an existing optimal algorithm for the basic model.

On the theoretical side, we consider the hydrostructure as a generalization of the standard notion of a *cut*, giving a more flexible technique for studying safety of many other types of covering walks of a graph. On the practical side, we believe that the hydrostructure could also lead to improvements of *practical* genome assembly from the point of view of *completeness*.

Chapter 1

Basic de nitions, motivations and state of the art

General background of genome assembly. Genome assembly is one of the flagship problems in Bioinformatics, along with other problems originating in—or highly motivated by—this field, such as edit distance computation, reconstructing and comparing phylogenetic trees, text indexing and compression. In genome assembly, we are given a collection of strings (or *reads*) and we need to reconstruct the unknown string (the genome) from which they originate. This is motivated by sequencing technologies that are able to read either "short" strings (100-250 length, Illumina technology), or "long" strings (10.000-50.000 length, Pacific Biosciences or Oxford Nanopore technologies) in huge amounts from the genomic sequence(s) in a sample. For example, the SARS-CoV-2 genome was obtained in [70] from short reads using the MEGAHIT assembler [47].

Other leading Bioinformatics problems have seen significant theoretical progress in major Computer Science venues, culminating (just to name a few) with both positive results, see e.g. [22, 69] for phylogeny problems, [8, 41] for text indexing, [26, 9, 42] for text compression, and negative results, see e.g. [4, 1, 5, 25] for string matching problems. However, the genome assembly problem is generally lacking major theoretical advances.

One reason for this stems from practice: the huge amount of data (e.g. the 3.1 Billion characters long human genome is read 50 times over) which impedes slower than linear-time algorithms, errors of the sequencing technologies (up to 15% for long reads), and various biases when reading certain genomic regions [57]. Another reason stems from theory: historically, finding an optimal genome assembly solution is considered NP-hard under several formulations [61, 40, 39, 53, 56, 36, 58], but, more fundamentally, even if one outputs a 3.1 Billion characters long string, this is likely incorrect, since problem formulations inherently admit a large number of solutions of such length [44].

Given all these setbacks, most state of the art assemblers, e.g., MEGAHIT [47] (for short reads), or wtdbg2 [64] (for long reads), generally employ a very simple and *linear-time* strategy,

dating back to 1995 [39]. They start by building an assembly graph encoding the overlaps of the reads, such as a *de Bruijn graph* [62] or an *overlap graph* [55] (graphs are directed in this thesis). After some simplifications to this graph to remove practical artifacts such as errors, at their core they find strings labeling paths whose internal nodes have in-degree and out-degree equal to 1 (called *unitigs*), approach dating back to 1995 [39]. That is, they do not output *entire* genome assemblies, but only shorter strings that are likely to be present in the sequenced genome, since unitigs do not branch at internal nodes.

Safe and complete algorithms. With the aim of enhancing the widely-used practical approach of assembling just unitigs—as those walks considered to be present in any possible assembly solution—a result in a major Bioinformatics venue [67] asked *what is the \limit" of the correctly reconstructible information from an assembly graph*. Moreover, is all such reconstructible information still obtainable in *linear time*, as in the case of the popular unitigs? Variants of this question also appeared in [32, 10, 56, 65, 45, 11], while other works already considered simple linear-time generalizations of unitigs [63, 54, 37, 44], without knowing if the "assembly limit" is reached.

To make this question precise, [67] introduced the following *safe and complete* framework. Given a notion of a solution to a problem (e.g. a type of walk in a graph), a partial solution (e.g. some shorter walk in the graph) is called *safe* if it appears (e.g. is a subwalk) in all solutions. An algorithm reporting only safe partial solutions is called a *safe algorithm*. A safe algorithm reporting *all* safe partial solutions is called *safe and complete*. A safe and complete algorithm outputs all and only what is likely part of the unknown object to be reconstructed, *synthesizing all solutions from the point of view of correctness.* Safety generalizes the existing notion of *persistency*: a *single* node or arc of the graph was called *persistent* if it appears in all solutions [33, 20, 15], for example persistent arcs for maximum bipartite matchings [20]. It also has roots in other Bioinformatics works [68, 16, 27, 72] considering the aligned symbols appearing in all optimal (and sub-optimal) alignments of two strings.

There are many theoretical formulations of genome assembly as an optimization problem, e.g. a shortest common superstring of all the reads [61, 40, 39], or some type of shortest walk covering all nodes or arcs of the assembly graph [63, 53, 54, 38, 36, 58, 56]. However, it is widely acknowledged [56, 58, 52, 57, 50, 44] that, apart from some being NP-hard, these formulations are lacking in several aspects, for example they collapse repeated regions of a genome. At present, given the complexity of the problem, there is no definitive notion of a "good" genome assembly solution. Therefore, [67] considered as genome assembly solution *any* closed arc-covering walk of a graph, where *arc-covering* means that it passes through each arc *at least* once. The main benefit of considering *any* arc-covering walk is that safe walks for them are safe also for any possible restriction of such covering walks (e.g. by some additional optimality criterion: for example, closed arc-covering walks are a common relaxation of the fundamental

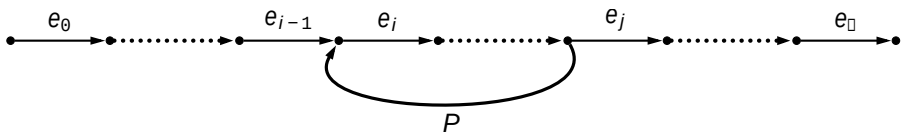

Figure 1.1: Walk $e_0 \cdots e_\ell$ is *not* an omnitig because there is a forbidden path P.

notions of closed Eulerian walk (we now pass through each arc *at least once*, instead of *exactly once* as in Eulerian walks), and of closed Chinese postman walk (i.e. a closed arc-covering walk *of minimum length*) [31], which were mentioned in [56] as unsatisfactory models of genome assembly. Put otherwise, safe walks for *all* arc-covering walks are more likely to be correct than safe walks for some particular type of arc-covering walks.

Prior results on safety in closed arc-covering walks. It is immediate to see that unitigs are safe walks for closed arc-covering walks. A first safe generalization of unitigs consisted of those paths whose internal nodes have only out-degree equal to 1 (with no restriction on their in-degree) [63]. Further, these safe paths have been generalized in [54, 37, 44] to those partitionable into a prefix whose nodes have in-degree equal to 1, and a suffix whose nodes have out-degree equal to 1. *All* safe walks for closed arc-covering walks were characterized by [67, 66] as being exactly those that are *omnitigs*, see Definition 1.1, Figure 1.1, and Theorem 1.7. This leads to the first *safe and complete* genome assembly algorithm (obtained thus 20 years after unitigs were first considered), outputting all *maximal* omnitigs in polynomial time (maximal omnitigs are those which are not sub-walks of other omnitigs). See Section 1.0.2, to have detailed definitions of structures presented next.

De nition 1.1 (Omnitig). *Let $W = e_0 \cdots e_\ell$ be a walk. We say that a non-empty path P is a j-i* forbidden path *for W, for some $1 \leq i \leq j \leq \ell$, if the first arc of P has the same tail as e_j and is different from e_j, and the last arc of P has the same head as e_{i-1} and is different from e_{i-1}. We say that W is an* omnitig *if for no $1 \leq i \leq j \leq \ell$ there exists a j-i forbidden path for W.*

Furthermore, through experiments on "perfect" human read datasets, [67] also showed that strings labeling omnitigs are about 60% longer on average than unitigs, and contain about 60% more biological content on average. Thus, once other issues of real data (e.g. errors) are added to the problem formulation, omnitigs (and the safe walks for such extended models) have the potential to significantly improve the quality of genome assembly results. Nevertheless, for this to be possible, one first needs the best possible results for omnitigs (given e.g. the sheer size of the read datasets), and a full comprehension of them, otherwise, such extensions are hard to solve efficiently.

Cairo et al. [14] recently proved that the length of all maximal omnitigs of any graph with n nodes and m arcs is $O(nm)$, and proposed an $O(nm)$-time algorithm enumerating all maximal

omnitigs. This was also proven to be optimal, in the sense that they constructed families of graphs where the total length of all maximal omnitigs is $\Theta(nm)$. However, it was left open if it is necessary to pay $O(nm)$ even when the total length of the output is smaller. Moreover, that algorithm cannot break this barrier, because e.g. $O(m)$-time traversals have to be done for $O(n)$ cases.

Significance of our results. The results we show in Chapter 2 shows that *all* the strings that can be correctly assembled from a graph can be obtained in output-sensitive linear time, a time feasible for being implemented in practical genome assemblers. It closes the issue of finding safe walks for a fundamental model of genome assembly (*any* closed arc-covering walk), a long-standing theoretical question and originating with the use of unitigs in 1995 [39].

This theoretical question is crucial also from the practical point of view: assembly graphs have the number of nodes and arcs in the order of millions, and yet the total length of the maximal omnitigs is almost linear in the size of the graph. For example, the compressed (see Definition 2.6) de Bruijn graph of human chromosome 10 (length 135 million) has 467 thousand arcs [14, Table 1], and the length of all maximal omnitigs (i.e. their total number of arcs, not their total string length) is 893 thousand. Moreover, even though this chromosome is only about 4% of the full human genome, the authors of [14] obtained that the running time of the *quadratic* algorithm on the compressed de Bruijn graph of the genome was about 30 minutes.

We envision a reverse transfer from theory to practical and *complete* genome assembly programs, as in other Bioinformatics problems. For example, trivially, safe walks for all closed arc-covering walks are also safe for more specific types of arc-covering walks. Moreover, while a genome solution defined as a single closed arc-covering walk does not incorporate several practical issues of real data, in Chapter 3 we show that omnitigs are the basis of more advanced models handling many practical aspects. For example, to allow more types of genomes to be assembled, one can define an assembly solution as a *set* of closed walks that together cover all arcs [2], which is the case in *metagenomic* sequencing of bacteria. For linear chromosomes (as in eukaryotes such as human), or when modeling missing sequencing coverage, one can analogously consider one, or many, such *open* walks [66, 67]. Safe walks for all these models are subsets of omnitigs [2, 13]. Moreover, when modeling sequencing errors, or mutations present e.g. only in the mother copy of a chromosome (and not in the father's copy), one can require some arcs not to be covered by a solution walk, or even to be "invisible" from the point of view safety. Finding safe walks for such models is also based on first finding omnitigs-like walks [13].

Notice that such separation between theoretical formulations and their practical embodiments is common for many classical problems in Bioinformatics. For example, computing edit distance is often replaced with computing edit distance under affine gap costs [23], or enhanced with various heuristics as in the well-known BLAST aligner [3]. Also text indexes such as the FM-index [26] are extended in popular read mapping tools (e.g. [48, 46]) with many heuristics

handling errors and mutations in the reads.

Finally, our results show that safe partial solutions enjoy interesting combinatorial properties, further promoting the persistency and safety frameworks. For real-world problems admitting multiple solutions, safe and complete algorithms are more pragmatic than the classical approach of outputting an arbitrary optimal solution. They are also more efficient than enumerating all, or only the first k-best, solutions [24], because they already *synthesize all that can be correctly reconstructed from the input data.*

State-of-the-art in genome assembly. Most problems in Bioinformatics are based at their core on some theoretical computational problem. After initial progress based on heuristics, several such Bioinformatics problems witnessed a drastic improvement in their practical solutions as a consequence of a breakthrough in their theoretical foundations. A major example is how the FM-index [26] revolutionized the problem of *read mapping*, being central in tools such as [49, 48, 46]. Other such theoretical breakthroughs include computing quartet distance [22, 69] motivated by *phylogenetics*, or fine-grained complexity lower bounds for *edit distance* computation [4] motivated by *biological sequence alignment*. However, despite this successful exchange of problems and results between Bioinformatics and Theoretical Computer Science, another flagship Bioinformatics problem, *genome assembly*, is generally lacking similar developments.

As we already anticipated, given a collection of *reads* (short strings sequenced from an unknown source genome), the main task is to reconstruct the source genome from which the reads were sequenced. This is one of the oldest problems in Bioinformatics [61], whose formulations range from a shortest common superstring of the reads [61, 40, 39], to various models of node- or arc-covering walks in different *assembly graphs* (encoding the overlaps between the reads, such as *de Bruijn graphs* [62], or *overlap graphs* [55]) [63, 53, 54, 38, 36, 58, 56]. In general, most such models of genome assembly are NP-hard. However, a more fundamental theoretical limitation in genome assembly is that such "global" problem formulations inherently admit a large number of solutions [44], given the large size and complexity of the input data. In practice, genome assemblers output only shorter strings that are likely to be *correct* (i.e. are substrings of the source genome) [57, 50, 52]. Such a strategy commonly uses the assembly graph to find only the paths (*unitigs*) whose internal nodes have *unit* in- and out-degree. Since unitigs do not branch, their labels are *correct* and can also be computed in linear time. The use of unitigs dates back to 1995 [39] and is at the core of most state-of-the-art genome assemblers, for both *long reads* (such as wtdbg2 [64]), and *short reads* (such as MEGAHIT [47]). Even though *long reads* are theoretically preferable, due to various practical limitations *short reads* are still used in many biomedical applications, such as the assembly of the SARS-CoV-2 genome [70].

Surpassing the theoretical limitations using safe and complete algorithms. Despite being at the core of the state-of-the-art in both theory and practice, there is no reason why only 'unitigs' be the basis of correct partial answers to the genome assembly problem. In fact,

various results [32, 10, 56, 65, 45, 11] presented the *open question* about the "assembly limit" (if any), or formally, *what all can be correctly assembled from the input reads*, by considering both graph theoretic and non-graph theoretic formulations. Unitigs were first generalized by [63] by considered the paths having internal nodes with *unit* out-degree (with no restriction on in-degree). These were later generalized [54, 37, 44] evolving the idea of *correctness* to the paths of the form $P = P_1eP_2$, such that e is an arc, the nodes of path P_1 have unit in-degree, and the nodes of path P_2 have unit out-degree (intuitively, P_1 is the only way to reach e, and P_2 is the only way e reaches other nodes). The question about the "assembly limit" was finally resolved in 2016 (around 20 years after unitigs were first introduced) in a major Bioinformatics venue [67] by introducing *safe and complete* algorithms for the problem. Notions similar to *safety* were studied earlier in Bioinformatics [68, 16, 57], and in other fields, including *persistence* [33, 20, 15], *d-transversals* [21], *d-blockers* [71], and *vital nodes/arc* [7].

In general, given an assembly graph, the most basic notion of a solution (or a source genome) is that of a walk covering all nodes or all arcs at least once, thereby explaining their existence in the assembly graph [67, 56, 58, 52, 57, 50, 44]. The safe walks for this notion of solution include unitigs and their generalizations described above. Tomescu and Medvedev [67] characterized the safe walks w.r.t. closed arc-covering walks as *omnitigs*. On simulated error-free reads where the source genome is indeed a closed arc-covering walk, omnitigs were found to be on average 60% longer than unitigs, and to contain 60% more biological information without employing any heuristics. Moreover, [67] presented an $O(m^2n)$-time algorithm[1] finding *all* safe walks for such genome assembly solutions, in a graph with m arcs and n nodes. Later, Cairo et al. [14] improved this bound to $O(mn)$, which they also proved to be optimal using worst-case graphs having $\Theta(mn)$-sized solutions. In Chapter 2, we present a *linear-time* output-sensitive algorithm for computing all maximal omnitigs, using a compact representation of the safe walks, called *macrotigs* (see Definition 2.21).

Limitations of the existing theory of safe and complete algorithms. To better understand the motivation behind the work presented in Chapter 3, we quickly recall a known notation of *certi cates*. In general, given a decision problem, a certificate is a string (or a structure) that certifies the answer, *yes* or *no*, to the problem. For example, given a walk W in a graph, to prove that W is *not* an omnitig (Definition 1.1), one has to highlight its forbidden path in the graph; the existence of such a forbidden path acts as a NO-certificate to the decision problem.

Despite presenting optimal algorithms, a theoretical limitation of the previous results from [67, 14, 2] is that safety is characterized in terms of *forbidden structures* (i.e. NO-certificates) for the safety of a walk. These turned out to be an unnatural view on more advanced models of genome assembly, where only a subset of the omnitigs is safe (see next subsections). As such,

[1] Trivial analysis using new results about omnitigs proves $O(m^2n)$ time for [67], though not explicitly stated.

these characterizations were incomplete and unnatural in the absence of easily verifiable YES-certificates. To illustrate this, consider the classical notion of a *strong bridge* (see Figure 1.2): an arc $(x;y)$ is a strong bridge in a graph G if and only if there exist nodes u and v in G such that all the paths from u to v in G contain the arc $(x;y)$. Its NO-certificate is a path from x to y avoiding $(x;y)$, which can thus be seen as a forbidden path. The corresponding YES-certificate is a *cut* between a set of nodes S (containing at least the nodes reachable from x without using $(x;y)$), and the remaining nodes T (containing at least the nodes reaching y without using $(x;y)$), such that the only arc crossing the cut is $(x;y)$. Such a certificate captures much more information about the structure of the graph from the viewpoint of the arc $(x;y)$. We generalize this idea from single arcs to walks to get a new perspective on safety problems in genome assembly. For that, we use a similar graph structure (now recognizing *safe* walks) which is essentially a generalization of a cut, and hence a YES-certificate, as follows.

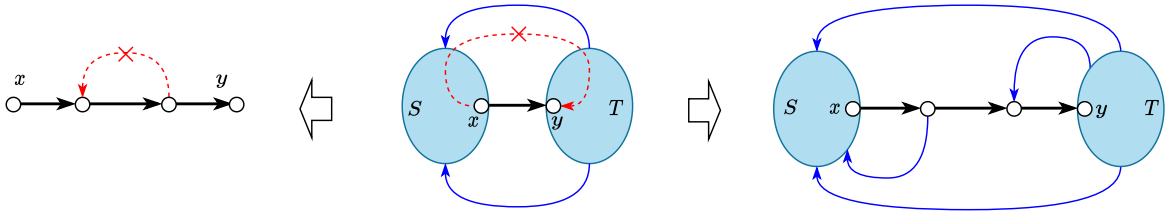

Figure 1.2: The perspective of a safe walk (bold black), generalizing a strong bridge. Center: The strong bridge $(x;y)$ has a NO-certi cate in form of a forbidden path (red), and a YES-certi cate in form of a cut between S and T where $(x;y)$ is the only arc leaving S. Left: The NO-certi cate of the safety of the walk from x to y is a similar forbidden path (red). Right: The YES-certi cate of the safety of the walk from x to y is a graph partition where T is reachable from S only by using the walk from x to y contiguously.

Our perspective to study safety distinguishes between the reasons for the safety of a walk. In general, the primary and simplistic reason is the *covering* constraint making every arc individually safe (and hence the left and right extensions of the arc), from the more profound reason arising from the *bridge-like* nature of some walks, which generalizes the property of a strong bridge. Such *bridge-like* walks are required to be traversed contiguously for reachability between some nodes. Thus, analogous to NO-certificates of a strong bridge (see Figure 1.2), a bridge-like walk (say from x to y) requires the absence of a forbidden path which allows reaching y from x without traversing the whole walk contiguously, as described in previous results [67, 14, 2]. Similarly, analogous to the YES-certificate of a strong bridge (see Figure 1.2), a bridge-like walk has a directed *cut-like* structure between the set of nodes S reachable from x without traversing the whole walk, and the set of nodes T reaching y without traversing the whole walk. Crossing the cut from y to x uses the remainder of the graph, whereas crossing the cut from x to y (or S to T) requires traversing the whole walk contiguously. This partitions the whole graph from the perspective of the walk, reducing the requirement for the contiguous traversal of the walk, to the simple requirement to reach from x to y, allowing simpler charac-

terizations of more advanced models. Moreover, we can now use the same YES-certificate for many different models, as opposed to finding a NO-certificate separately for each model. Thus, our new perspective of a safe walk as a generalization of a strong bridge results in a universal approach for the complete characterization of safe walks.

Formulation of practically relevant genome assembly models. Modeling a genome assembly solution as a single arc-covering walk is extremely limiting in practice, due to the presence of multiple (not necessarily circular) genomes in the sample, sequencing errors or unsequenced genomic regions. See Section 1.0.1 for further motivation of our definitions below.

Most of these practical issues can be handled by considering more flexible theoretical formulations of the problem. Instead of always considering the solution to be a *single* closed arc-covering walk of the assembly graph, we can change the model so that the solution is an *arc-covering* collection of k 2 closed walks (i.e. every arc appears in some walk of the collection). Further, as one can see in [13], when addressing *linear* genomes, or unsequenced genomic regions, we further change the model so that the solution is one *open* arc-covering walk from a given node s to a given node t (*s-t walk*), or an arc-covering collection of exactly k 2 open s-t walks. Moreover, if there is no constraint k on the number of walks in the collection, then we will say that $k = 1$.

Definition 1.2 (k-circular safe walk, k-st safe walk). *Let $G = V [E$ be a graph, let $s; t 2 V$ and let k 1. A walk W is called* k-circular safe *(or* k-st safe*) if W is a subwalk of at least one walk of any arc-covering collection of exactly k circular walks, see Section 1.0.2, (or exactly k walks from s to t).*

Remark 1.3. *A graph admits an arc-covering collection of k 1 circular walks if and only if it is a disjoint union of at most k strongly connected graphs. As such, in the circular models we can assume the graph to be strongly connected. In the linear models, we first solve the strongly connected case, and then solve the cases $k = 1; 1$ for non-strongly connected graphs.*

Further, the notion of genome assembly solution can be naturally extended to handle sequencing errors. For example, the models can be extended so that the collection of walks is required to cover only a *subset F* of the arcs (*F-covering*). Another possible extension is to mark the erroneous arcs in E r F as *invisible*, in the sense that they are invisible when we define safety (*F-visible*). These not only allow handling errors, but also allow handling even more general notions of genome assembly solutions through simple reductions (see Remark 1.6).

Definition 1.4 (F-subsequence). *Let $G = V [E$ be a graph, let F E and let W be a walk. We call* F-subsequence *of a walk W, the ordered sequence of arcs of W obtained by removing every arc e such that e 2 F.*

Definition 1.5 (Subset covering / visible). *Let $G = V [E$ be a graph, let $s; t 2 V$, F E, and let k 1. A walk W is called:*

- F-covering k-circular safe *(or F-covering k-st safe) if W is a subwalk of at least one walk of every F-covering collection of exactly k circular walks (or exactly k walks from s to t).*
- F-visible k-circular safe (or F-visible k-st safe) *if the F-subsequence of W occurs contiguously in the F-subsequence of at least one walk of every arc-covering collection of exactly k circular walks (or exactly k walks from s to t).*

Remark 1.6. *The subset covering model also allows us to solve a generalization of the linear models, where the walks in the collection start in any node of a given set S, and end in any node of a given set T. For that, we set $F = E$, add a new global source s connected by new arcs not in F to all nodes in S, and an analogous global sink t connected from every node in T. Moreover, we can also combine the subset covering and subset visibility models for some $F; F' \subseteq E$, to get F-covering F'-visible safe walks, for both circular and linear models, and obtain analogous results (see also Figure 3.1). This also allows us to solve the same models in a node-centric formulation, where only (a subset of) the nodes are required to be covered and/or visible. This can be achieved using a simple transformation of the graph expanding each node to an arc and choosing only such node-arcs as the subset to be covered.*

1.0.1 Bioinformatics Motivation

Assuming we are sequencing a single circular genome (as when sequencing a single bacterium), the most basic notion of a solution (or a source genome) is that of a walk in the assembly graph covering all nodes or all arcs at least once, thereby explaining their existence in the assembly graph [67]. Even if the earlier works (on "global" genome assembly formulations) include various *shortest* versions of such walks (e.g. Eulerian, Chinese postman, or even Hamiltonian), it is widely acknowledged [56, 58, 52, 57, 50, 44] that a shortest walk misses repeats in the source genome. Moreover, safe walks for *all* arc-covering walks are also trivially safe for more specialised types of walks. Both of these facts are related to the lack of constraints of the solution walks, except that to require that they are indeed arc-covering.

However, such theoretical formulation of genome assembly using a single closed arc-covering walk uses the following assumptions. (i) All the reads are sequenced from a *single circular* genome, such that these reads have (ii) *no errors*, and (iii) *no missing coverage* (e.g. every position of the genome is covered by some read). However, these are very strong and impractical assumptions that are violated by real input data sets, as we explain next:

- Assumption (i) is not practical for several reasons. When sequencing a bacterium [18], one indeed obtains reads from a *single circular* genome. However, when sequencing all bacteria in an environmental sample (as in *metagenomics* [51]), the reads originate from *multiple circular* genomes. In case of a virus [29], the reads originate from a *single linear* genome. Finally, when sequencing eukaryotes such as human [12], the reads originate from *multiple linear* chromosomes.

- Assumption (ii) is not practical because the sequencing process introduces errors in the reads, with error rates ranging from 1% for short reads, up to 15% for long reads. Such read errors produce certain known structures in the assembly graph [57], such as *tips* (short induced paths ending in a sink), and *bubbles* (two induced paths of the same length, and with the same endpoints, see also their generalisation to *superbubbles* [34, 28, 60]), which are usually handled in an initial *error-correction* stage using heuristics (e.g. tips are removed, and bubbles are "popped" by removing one of the parallel paths). Such bubbles can also arise from a correct read position in diploid genomes (such as human), but on which the mother's copy of the chromosome differs from the father's. Popping bubbles is favorable in this setting in order to obtain longer assemblies (since unitigs would otherwise be broken up by the endpoints of the bubble). Moreover, in a de Bruijn graph, arcs that appear very few times (having low *abundance*) in the input reads are heuristically removed, as they are assumed to be errors. But in practice, the assembly graph's topology might give evidence for the correctness of these arcs, especially in scenarios where low abundance is common [47], so a more accurate removal strategy is likely to result in a better assembly. This applies even to state-of-the-art long read assemblers like wtdbg2 [64] which simply removes low abundance arcs.
- Assumption (iii) is not true due to the practical limitations of current sequencing technologies. Thus, since not all parts of the genome can always be read, even if the sample contains a *single circular* genome, the reads appear as if they were sequenced from *multiple linear* genomic sequences.

The genome assembly models from Section 2.1 handle all such issues flexibly and in a theoretically solid way by considering *collections* of closed or open walks. In the subset covering models, the arcs not required to be covered can be those in tips, bubbles and those with low abundance in the reads. However, merely making them avoidable (and not removing them) can break the safety around such regions. Hence, another possible extension is to mark such parts of the graph as invisible. For example, marking bubbles as invisible prevents disrupting the safety of their flanking regions. See Figure 1.3 for an example of these models.

1.0.2 De nitions and notations

In this section we elaborate definitions and notations we used throughout the thesis; despite the fact that some definitions were already introduced, we are going to restate them when needed, to better clarify new concepts. The section is divided into paragraphs distinguishing and highlighting the differences between definitions and notations (e.g., two different definitions of walk in a graph) adopted in the two main chapters of the thesis. Indeed, in Chapter 3, these differences caused minor changes to the terminology we used as well; this was done to help us in handling some particular cases, which we are going to explain in detail.

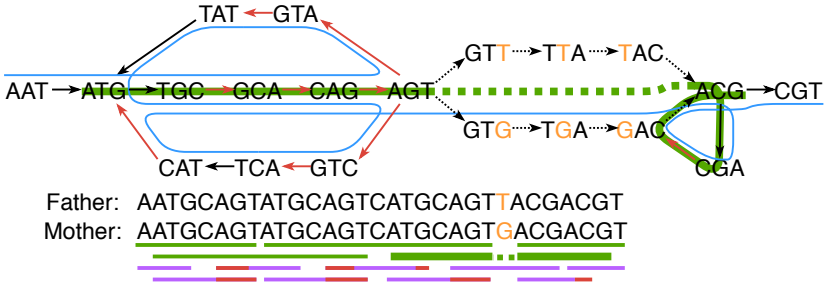

Figure 1.3: Assume a \full" read coverage scenario in which reads of length at least 4 have been sequenced starting from each position a diploid genome (i.e. from both the mother and the father copy of a chromosome). An *arc-centric* de Bruijn graph of order 3 has all substrings of length 3 of the reads as nodes, and all substrings of length 4 as arcs (from their length-3 pre x to their length-3 su x). The thin blue line depicts the mother genome as a walk in this graph. Orange marks a position in which the two haplotypes di er; this induces a bubble in the de Bruijn graph (dotted arcs). We assume that the red arcs have low abundance in the read dataset for some natural reasons, despite being correct. But they would still be removed by current heuristic error-correction strategies, even though our models show evidence for their correctness. We set the dotted arcs to invisible, and model the genome as an s-t walk not required to cover the red and dotted arcs, where s = AAT and t = CGT. Below the mother's haplotype, we mark in green its maximal 1-st safe substrings. The thick safe walk of the mother's haplotype is also depicted in the graph (in green as well). We show in magenta the unitigs of the graph aligned to the mother haplotype. Additionally, the red parts of the unitigs would be missing if the red arcs were removed.

Chapter 2 de nitions and notations. A *graph* is a pair $G = (V; E)$, where V is a finite set of *nodes* $v \in V$, E is a finite multi-set of ordered pairs of nodes called *arcs*, $e = (u; v) \in E$, where $u; v \in V$; an arc $e = (u; v)$ is *incident* to both the nodes u and v it connects. Parallel arcs and self-loops are allowed. The *reverse graph* G^R of G is obtained by reversing the direction of every arc. For $e \in E$, we denote $G - e = V \cup (E - \{e\})$; analogously, for $v \in V$, we denote $G - v = (V - \{v\}) \cup E$. We assume, if not stated otherwise, a graph $G = (V; E)$ to be strongly connected, with $|V| = n$ and $|E| = m \geq n$. The *in-degree* $d^-(v)$ of a node v denotes the number of arcs incident to v with their head node, symmetrically, the *out-degree* $d^+(v)$ of a node v denotes the number of arcs incident to v with their tail node. The *degree* of a node v is $d(v) = d^-(v) + d^+(v)$. In the rest of the thesis, we assume a fixed strongly connected graph $G = V \cup E$ which is not a cycle [2], with $|V| = n$ and $|E| = m > n$.

A *walk* in G is a sequence $W = (v_0; e_1; v_1; e_2; \ldots; v_{\ell-1}; e_\ell; v_\ell)$, $\ell \geq 0$, where $v_0; v_1; \ldots; v_\ell \in V$, and each e_i is an arc from v_{i-1} to v_i. Sometimes the nodes $v_0; \ldots; v_\ell$ of a walk W may be omitted to write W more compactly as $e_1 \ldots e_\ell$, if $\ell \geq 1$. If an arc e appears in W, we write $e \in W$.

Functions $t(\,)$ and $h(\,)$ denote, respectively, the *tail* node and the *head* node of an arc or a walk.

We say that $W = (v_0; e_1; \ldots; e_\ell; v_\ell)$ goes from $t(W) = v_0$ to $h(W) = v_\ell$, has *length* ℓ, contains $v_1; \ldots; v_{\ell-1}$ as *internal nodes*, *starts with* e_1, *ends with* e_ℓ, and contains $e_2; \ldots; e_{\ell-1}$ as

[2] Safe walks in a cycle are not properly de ned as they can repeat inde nitely.

internal arcs. A walk W is called *empty* if it has length zero, and *non-empty* otherwise. There exists exactly one empty walk $\varepsilon_v = (v)$ for every node $v \in V$, and $t(\varepsilon_v) = h(\varepsilon_v) = v$. A walk W is called *closed* if it is non-empty and $t(W) = h(W)$, otherwise it is *open*. The concatenation of walks W and W' (with $h(W) = t(W')$) is denoted WW'. A walk $W = (v_0, e_1, v_1, \ldots, e_\ell, v_\ell)$ is called a *path* when the nodes $v_0, v_1, \ldots, v_\ell$ are all distinct, with the exception that $v_\ell = v_0$ is allowed (in which case we have either a closed or an empty path). Subwalks of open walks are defined in the standard manner. For a closed walk $W = e_0 \ldots e_{\ell-1}$, we say that a walk $W' = e'_0 \ldots e'_j$, $j \leq \ell - 1$, is a subwalk of W if there exists $i \in \{0, \ldots, \ell - 1\}$ such that for every $k \in \{0, \ldots, j\}$ it holds that $e'_k = e_{(i+k) \bmod \ell}$.

A closed *arc-covering* walk (i.e. passing through every arc at least once) exists if and only if the graph is strongly connected. We are interested in the (safe) walks that are subwalks of all closed arc-covering walks, characterized in [67].

Theorem 1.7 ([67]). *Let G be a strongly connected graph different from a closed path. Then a walk W is a subwalk of all closed arc-covering walks of G if and only if W is an* omnitig.

Terminology. To give a deeper insight on the structure and properties of safe walks, we classify the nodes and arcs of a strongly connected graph as follows (see Figure 2.1 for a visual reference): (i) A node v is a *join node* if $d^-(v) > 1$, and a *join-free node* otherwise. An arc f is called a *join arc* if $h(f)$ is a join node, and a *join-free arc* otherwise. (ii) A node v is a *split node* if $d^+(v) > 1$, and a *split-free node* otherwise. An arc g is called a *split arc* if $t(g)$ is a split node, and a *split-free arc* otherwise. (iii) A node or arc is called *bivalent* if it is both join and split, and it is called *biunivocal* if it is both split-free and join-free. A walk W is *split-free* (resp., *join-free*) if all its arcs are split-free (resp., join-free). Given a walk W, its *univocal extension* $U(W)$ is defined as $W^- WW^+$, where W^- is the longest join-free path to $t(W)$ and W^+ is the longest split-free path from $h(W)$ (observe that they are uniquely defined).

Notice that W is an omnitig in G if and only if W^R is an omnitig in G^R. Moreover, any subwalk of an omnitig is an omnitig. For every arc e, its univocal extension $U(e)$ is an omnitig. A walk W satisfying a property P is *right-maximal* (resp., *left-maximal*) if there is no walk We (resp., eW) satisfying P. A walk satisfying P is *maximal* if it is left- and right-maximal w.r.t. P. Notice that if G is a closed path, then every *walk* of G is an omnitig. As such, it is relevant to find the maximal omnitigs of G only when G is different from a closed path. Thus, in the rest of the thesis a strongly connected graph G is considered to be different from a closed path, even when we do not mention it explicitly.

Chapter 3 definitions and notations. Refer to the previous paragraphs for every definition that is not restated here. A *strongly connected component* of a graph (SCC) is a maximal subgraph that is *strongly connected*, i.e., for any two nodes x and y in the SCC, there exists a path from x to y. Similarly, a *weakly connected component* (WCC) is a maximal subgraph that

is weakly connected, i.e., any two nodes x and y in the WCC are connected by an undirected path. A node is called a *source* if it has no incoming arcs, and a node is called a *sink* if it has no outgoing arcs.

A w_1-w_ℓ *walk* (or simply *walk*) in G is a non-empty alternating sequence of nodes and arcs $W = (w_1, \ldots, w_\ell)$, where for all $1 \quad i < \ell$: $\text{HEAD}(w_i) = w_{i+1}$ if w_{i+1} is a node, and $\text{TAIL}(w_{i+1}) = w_i$ otherwise. However, if not otherwise indicated, in a w_1-w_ℓ walk of G, w_1 and w_ℓ are *arcs*. We call $\text{START}(W) = w_1$ the *start* of W, and $\text{END}(W) = w_\ell$ the *end* of W. W is a *path* if it repeats no node or arc, except that $\text{START}(W)$ may equal $\text{END}(W)$.

A walk W is called *closed* if $\text{START}(W) = \text{END}(W)$ and $\ell > 1$, otherwise it is *open*. We may sometimes use the term *circular walks* referring to a particular kind of closed walks, that is a closed walk in which its starting/ending node or arc has lost of significance and the walk itself can be considered to be repeated indefinitely. The notation WW' denotes the concatenation of walks $W = (w_1, \ldots, w_\ell)$ and $W' = (w'_1, \ldots, w'_{\ell'})$, if $(w_1, \ldots, w_\ell, w'_1, \ldots, w'_{\ell'})$ is a walk. When writing a walk as a concatenation, then lower-case letters denote single nodes or arcs, e.g., aZb denotes the walk $(a, z_1, \ldots, z_\ell, b)$ (where a and b are arcs and $Z = (z_1, \ldots, z_\ell)$). Subwalks of walks are defined in the standard manner, where subwalks of closed walks do not repeat the start/end if they run over the end. A walk W is *bridge-like* if there exist $x, y \in G$ such that each x-y walk contains W as subwalk, and otherwise it is called *avertible*. Observe that a bridge-like walk is an open path.

A *walk-cover* of a graph is a set of walks that together cover all arcs of the graph, and its size is the number of walks. The minimum walk-cover (and hence its size) can be computed in $O(mn)$ time using minimum flows, by reducing the problem to maximum flows (see e.g. [6]) and applying Orlin's and King's $O(mn)$ time algorithms [59, 43].

Terminology. As we did for the previous paragraphs, some terminology is restated here with the notation adopted in Chapter 3, to help the reader easily follow the relative chapter. A walk $W = (w_1, \ldots, w_\ell)$ is *univocal* if no node in W is a split node, except possibly w_ℓ (if it is a node) and it is *R-univocal* if no node in W is a join node, except possibly w_1 (if it is a node) , and it is *biunivocal* if it is both univocal and R-univocal. Its *univocal extension* $U(W)$ is $W^l W W^r$ where $W^l \text{START}(W)$ is the longest R-univocal walk to $\text{START}(W)$ and $\text{START}(W)W^r$ is the longest univocal walk from $\text{END}(W)$.

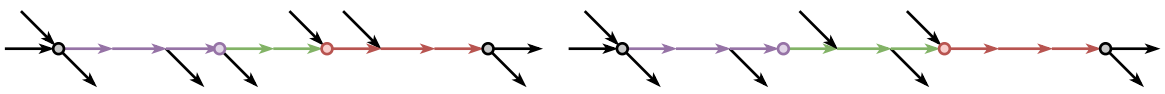

Figure 1.4: A trivial walk on the left and a non-trivial walk on the right. The heart is colored green, the left wing violet and the right wing red. The left wing is from arc to node and the right wing from node to arc.

For a walk $W = w_1, \ldots, w_\ell$ let w_i be its first join arc or $w_i = w_1$ if W has no join arc, and

let w_j be its last split arc or $w_j = w_{\cdot}$ if W has no split arc. If $i \quad j$, then the *trivial heart* (or simply *heart*) $\mathsf{Heart}(W)$ of W is its w_j-w_i subwalk, and otherwise the *non-trivial heart* (or simply *heart*) $\mathsf{Heart}(W)$ is its w_i-w_j subwalk. A walk with a trivial heart is a *trivial walk*, and a walk with a non-trivial heart is a *non-trivial walk*. The *left wing* and *right wing* of W are W^l and W^r in the decomposition $W^l\mathsf{Heart}(W)W^r$. See Figure 1.4 for a visualisation of these definitions.

Further, we are going to extensively use some particular results of Chapter 2 ((a) and (b)), briefly summarized in the following theorem. The first result was also used in [2] (and possibly other previous works) even though not stated explicitly.

Theorem 1.8. *For a strongly connected graph G with n nodes and m arcs, the following hold:*

(a) (Two-Pointer Algorithm [2]) *Given a walk W and a procedure A to verify the safety of its subwalks, all the maximal safe subwalks of W can be reported in $O(|W| f(m;n))$ time, where each invocation of A requires $f(m;n)$ time.*

(b) (Omnitig Bounds [14]) *There are at most m maximal 1-circular safe walks [14] where each has length of at most $O(n)$ [14] and of which $O(n)$ are non-trivial (Chapter 2), and the total length of all maximal 1-circular safe walks in G is $O(mn)$ [14] (if G is not a cycle).*

(c) (Fault tolerant SCCs [30]) *We can preprocess G in $O(m)$ time, to report whether x and y are in the same SCC in $G \setminus z$ in $O(1)$-time, for any $x;y;z \in G$.*

Chapter 2

Genome assembly, from practice to theory: safe, complete and linear time

2.1 Introduction

Our main result is an $O(m)$-size representation of all maximal omnitigs, based on a careful structural decomposition of the omnitigs of a graph. This is surprising, given that there are families of graphs with $\Theta(nm)$ total length of maximal omnitigs [14]. Notice that the total length of the maximal omnitigs is at least m, since every arc is an omnitig by definition.

Theorem 2.1. *Given a strongly connected graph G with n nodes and m arcs, there exists an $O(m)$-size representation of all maximal omnitigs, consisting of a set M of walks (*maximal macrotigs*, Definition 2.21) of total length $O(n)$ and a set F of arcs, such that every maximal omnitig is the univocal extension of either a subwalk of a walk in M, or of an arc in F.*

Moreover, M, F, and the endpoints of macrotig subwalks univocally extending to maximal omnitigs can be computed in time $O(m)$.

Since the univocal extension $U(W)$ of a walk W can be trivially computed in time linear in the length of $U(W)$, we immediately get the linear-time output sensitive algorithm:

Corollary 2.1.1. *Given a strongly connected graph G, it is possible to enumerate all maximal omnitigs of G in time linear in their total length.*

We obtain Theorem 2.1 using two interesting ingredients. The first is a novel graph structure (that we called *macronodes*, Definition 2.10), obtained after a *compression* operation (described in Section 2.2.2) of 'easy' nodes and arcs. The second is a connection to a recent result by Georgiadis et al. [30] showing that it is possible to answer in $O(1)$-time strong connectivity queries under a *single* arc removal, after linear-time preprocessing (notice that a forbidden path is defined w.r.t. *two* arcs to avoid).

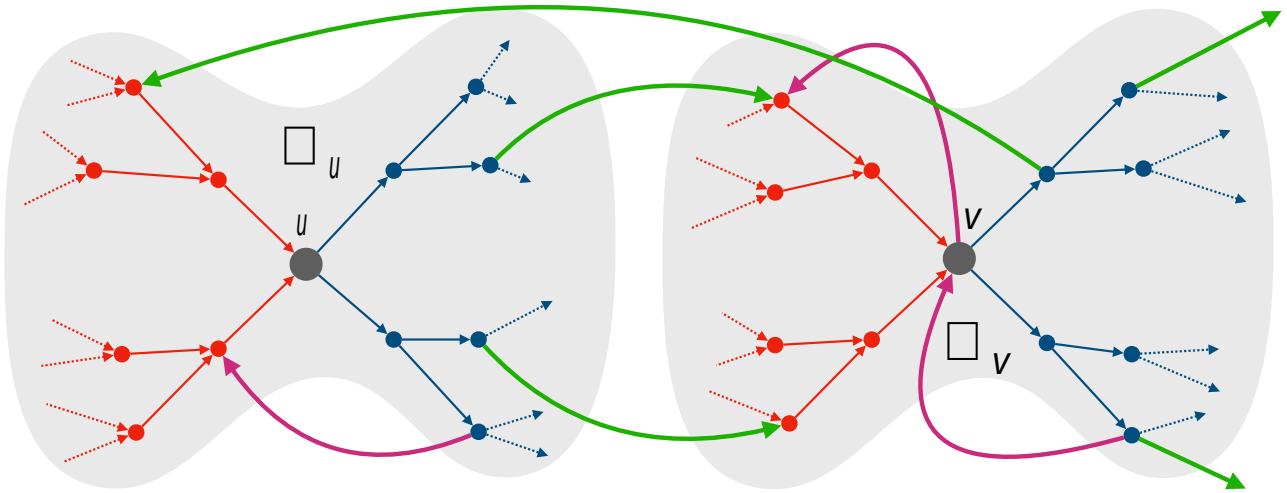

Figure 2.1: Figure 2.1: Given a bivalent node v, the macronode M_v is the subgraph of G induced by the nodes reaching v with a split-free path (in red), and the nodes reachable from v with a join-free path (in blue). These two types of nodes induce the two trees of the macronode. By de nition, every arc with endpoints in di erent macronodes is bivalent (in green, denoted *cross-bivalent arcs*). The remaining bivalent arcs have endpoints in the same macronode (in purple, denoted *self-bivalent arcs*).
.

Theorem 2.1 has additional practical implications. First, omnitigs are also representable in the same (linear) size as the commonly used unitigs. Second, maximal macrotigs enable various $O(m)$-time operations on maximal omnitigs (without listing them explicitly), for example, by pre-computing the univocal extensions from any node, which are needed in Theorem 2.1. For example, given that the number of maximal omnitigs is $O(m)$ [14], this implies the following result:

Corollary 2.1.2. *Given a strongly connected graph G with m arcs, it is possible to compute the lengths of all maximal omnitigs in total time $O(m)$.*

Corollary 2.1.2 leads to a linear-time computation of various statistics about maximal omnitigs, such as minimum, maximum, and average length (useful e.g. in [19]). One can also use this to filter out subfamilies of them (e.g. those of length smaller and/or larger than a given value) before enumerating them explicitly.

Structure. The main structural insight of this paper is that omnitigs enjoy surprisingly limited freedom, in the sense that any omnitig can be seen as a concatenation of walks in a very specific set. In order to give the simplest exposition, we first simplify the graph by contracting biunivocal nodes and arcs. The nodes of the resulting graph can now be partitioned into *macronodes* (see Figure 2.1 and Definition 2.10), where each macronode M_v is uniquely identified by a bivalent node v (its *center*). We can now split the problem by first finding omnitigs inside each macronode, and then characterizing the ways in which omnitigs from different macronodes can combine.

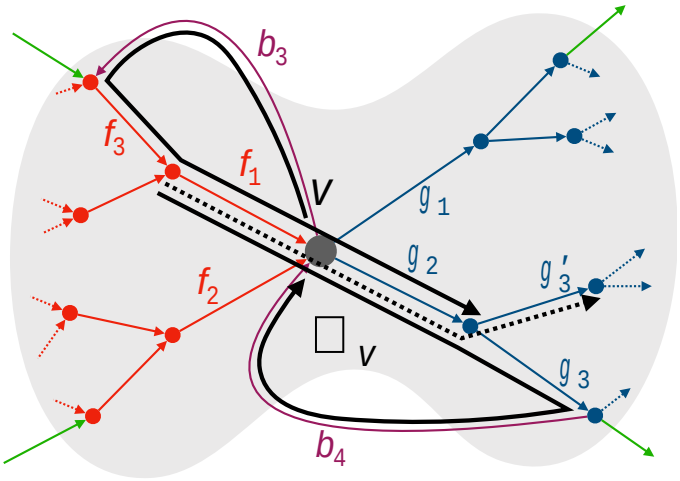

Figure 2.2: The only omnitig traversing the bivalent node v is f_1g_2; e.g., by the X-intersection Property neither f_2g_2 is an omnitig ($b_3f_3f_1$ is a forbidden path) nor f_1g_1 is an omnitig ($g_2g_3b_4$ is a forbidden path). Extending the central-micro omnitig (De nition 2.12) f_1g_2 to the right we notice that $f_1g_2g_3$ is an omnitig and by the Y-intersection Property $f_1g_2g'_3$ is not an omnitig (g_3b_4 is a forbidden path). Hence, the only maximal right-micro omnitig is $f_1g_2g_3b_4$, and the only maximal left-micro omnitig is $b_3f_3f_1g_2$. Merging the two on f_1g_2, we obtain the maximal microtig $b_3f_3f_1g_2g_3b_4$.

We discover a key combinatorial property of how omnitigs can be extended: there are at most two ways that any omnitig can traverse a macronode center (see also Figure 2.2):

Theorem 2.15 (X-intersection Property). *Let v be a bivalent node. Let $f_1 \neq f_2$ be join arcs with $h(f_1) = h(f_2) = v$; let $g_1 \neq g_2$ be split arcs with $t(g_1) = t(g_2) = v$.*

i) If f_1g_1 and f_2g_2 are omnitigs, then $d^+(v) = d^-(v) = 2$.

ii) If f_1g_1 is an omnitig, then there are no omnitigs f_1g' with $g' \neq g_1$, nor $f'g_1$ with $f' \neq f_1$.

In order to prove the X-intersection Property, we prove an even more fundamental property: once an omnitig traverses a macronode center, for any node it meets after the center node, there is at most one way of continuing from that node (Y-intersection Property), see Figure 2.2. The basic intuition is that if there is more than one possibilities, then strong connectivity creates forbidden paths.

Given an omnitig fg traversing the bivalent node v, we define the *maximal right-micro omnitig* as the longest extension fgW in the macronode M_v (see Figure 2.2 and Definition 2.12). The *maximal left-micro omnitig* is the symmetrical omnitig Wfg. By Theorem 2.15, there are at most two maximal right-micro omnitigs and two maximal left-micro omnitigs. The merging of a maximal left- and right-micro omnitig on fg is called a *maximal microtig* (see Figure 2.2 and Definition 2.12; notice that a microtig is not necessarily an omnitig). These *at most two* maximal microtigs represent "forced tracks" to be followed by omnitigs crossing v.

We now describe how omnitigs can advance from one macronode to another. We prove that any arc having endpoints in different macronodes is a bivalent arc, and moreover, for every

maximal microtig ending with a bivalent arc b, there is at most one maximal microtig starting with b. Maximal microtigs can be seen as omnitig tracks, i.e., walks in which omnitigs can be found in kind of a sequence. As such, when a so called omnitig track exits a macronode, there is at most one way of connecting it with an omnitig track from another macronode. It is natural to merge all omnitig tracks (i.e., maximal microtigs) on all bivalent arcs between *di erent* macronodes, and thus obtain *maximal macrotigs* (Definition 2.21 and Figure 2.7). The total size of all maximal macrotigs is $O(n)$ (Theorem 2.25), and they are a representation of all maximal omnitigs, except for those that are univocal extensions of the arcs of F, see below and Theorem 2.26.

Algorithms. Our algorithms first build the set M of maximal macrotigs, and then identify maximal omnitigs inside them. The set F of arcs univocally extending to the remaining maximal omnitigs will be the set of bivalent arcs not appearing in M (Theorem 2.26).

Crucial to the algorithms is an *extension* primitive deciding what new arc (if any) to choose when extending an omnitig (recall that the X- and Y-intersection Properties limits the *number* of such arcs to one). Suppose we have an omnitig fW, with f a join arc, and we need to decide if it can be extended with an arc g out-going from $h(W)$. Naturally, this extension can be found by checking that there is no forbidden path from $t(g) = h(W)$. However, this forbidden path can potentially end in any node of W. Up to this point, [66, 67, 14] need to do an entire $O(m)$ graph traversal to check if any node of W is reachable by a forbidden path.

We prove here a new key property:

Theorem 2.29 (Extension Property). *Let fW be an omnitig in G, where f is a join arc. Then fWg is an omnitig if and only if g is the only arc with $t(g) = h(W)$ such that there exists a path from $h(g)$ to $h(f)$ in $G \setminus f$.*

Thus, for each arc g with $t(g) = h(W)$, we can do a *single* reachability query under one arc removal: "does $h(g)$ reach $h(f)$ in $G \setminus f$?" Since the target of the reachability query is also the head of the arc excluded f, then we can apply an immediate consequence of [30]:

Theorem 2.2 ([30]). *Let G be a strongly connected graph with n nodes and m arcs. After $O(m)$-time preprocessing, one can build an $O(n)$-space data structure that, given a node w and an arc f, tests in $O(1)$ worst-case time if there is a path from w to $h(f)$ in $G \setminus f$.*

Using the Extension Property and Theorem 2.2, we can thus pay $O(1)$ time to check each out-outgoing arc g, before discovering the one (if any) with which to extend fW. In Section 2.2 we describe how to transform the graph to have constant degree, so that we pay $O(1)$ per node. This transformation also requires slight changes to the maximal omnitig enumeration algorithm to maintain the linear-time output sensitive complexity (see Section 2.4.1). We use the Extension Property when building the left- and right-maximal micro omnitigs, and when identifying maximal omnitigs inside macrotigs, as follows.

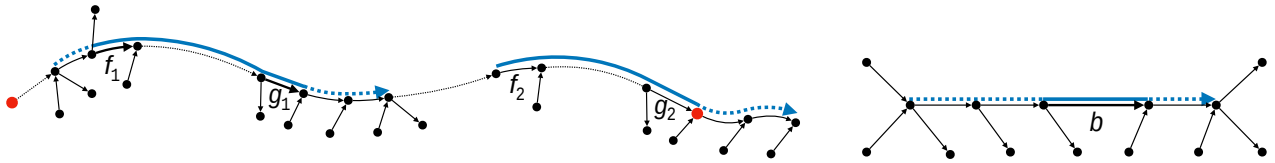

Figure 2.3: Any maximal omnitig is identified (in solid blue) either by a macrotig interval (from a join arc f to a split arc g; left), or by a bivalent arc b not appearing in any macrotig (right). The full maximal omnitig is obtained by univocal extension (dotted blue), extension which may go outside of the maximal macrotig, the extremities of which are represented by red dots.

Once we have the set M of maximal macrotigs, we scan each macrotig with two pointers, a left one always on a join arc f, and a right one always on a split arc g (see Figure 2.3 and Algorithm 5). Both pointers move from left to right in such a way that the subwalk between them is always an omnitig. The subwalk is grown to the right by moving the right pointer as long as it remains an omnitig (checked with the Extension Property).

When growing to the right is no longer possible, the omnitig is shrunk from the left by moving the left pointer. This technique runs in time linear to the total length of the maximal macrotigs, namely $O(n)$.

Comparison with previous techniques. The algorithm of [67] exhaustively extends an omnitig with every arc outgoing from its head, as long as the resulting walk remained an omnitig, and did not use any insights on the structure of omnitigs. The $O(nm)$-time algorithm of [14] was obtained using two structural results: there can be only one left-maximal omnitig ending with a split arc (which we do not use here, since we prove deeper insights on the structure of omnitigs, e.g. the X- and Y-intersection Properties) and the existence of an acyclic order between split arcs connected by "simple" omnitigs (which we use as Lemma 2.23 in Section 2.3.2). In [14], these allow computation to be memoized when recursively computing the left-maximal omnitig ending with a given split arc. The two-pointer technique was used also in [2] for a related problem, to test the safety of intervals of an *entire solution*. Our surprising discovery of macrotigs allow for a "small search space" of total size to $O(n)$, and eliminate the need of recursion, while the Extension Property enables the use of [30], thus the pay of $O(1)$ per omnitig extension, instead of $O(m)$ as in [66, 67, 14].

2.2 Constant degree and compression

In this section, we describe three transformations of a given graph G to guarantee the assumption of compression (contraction of biunivocal nodes and arcs) and constant degree on every node. It is easy to see that such transformations and their inverses can be performed in linear time.

2.2.1 Constant degree

The first transformation allows us to reduce to the case in which the graph has constant out-degree (see Figure 2.4 for an example).

Transformation 1. *Given G, for every node v with $d^+(v) > 2$, let $e_1; e_2; \ldots; e_k$ be the arcs outgoing from v. Replace v with the path $(v_1; e'_1; v_2; e'_2; \ldots; e'_{k-2}; v_{k-1})$, where $v_1; \ldots; v_{k-1}$ are new nodes, and $e'_1; \ldots; e'_{k-2}$ are new arcs. Each arc e_i with $t(e_i) = v$ in G now has $t(e_i) = t(e'_i) = v_i$, except for e_k which has $t(e_k) = v_{k-1}$. Each arc e of G, with $h(e) = v$, now has $h(e) = v_1$.*

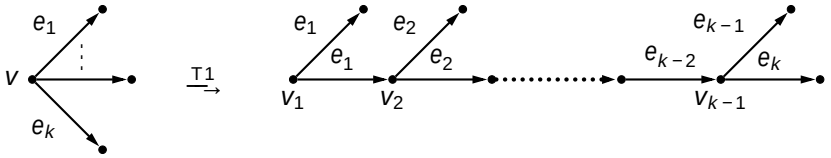

Figure 2.4: An example of Transformation 1 (T1) applied to the node v, where $e_1; \ldots; e_{k\ 2}$ $^+(v)$ are the arcs with tail equal to v.

By also applying the symmetric transformation, the problem on the original graph G is thus reduced to a graph G' with constant out- and in-degree. Notice that the number of arcs of G' is still $O(m)$, where m is the number of arcs of the original graph. As such, we can obtain the macrotigs of G' in $O(m)$ time. The trivial strategy to obtain all maximal omnitigs of G is to enumerate all maximal omnitigs of G', and from these contract all the new arcs introduced by the transformation (while also removing duplicate maximal omnitigs, if necessary). However, this may invalidate the linear-time complexity of the enumeration step, since the length of the maximal omnitigs of G may be super-linear in total maximal omnitig length of G, see Figure 2.5. In Section 2.4.1 we explain how we can easily modify the maximal omnitig enumeration step to maintain the $O(m)$ output-sensitive complexity.

To prove the correctness of Transformation 1, we proceed as follows. Let $c_e(G)$ be the graph obtained from G by contracting an arc e (*contracting* e means that we remove e and identify its endpoints). For every walk W of G, we denote by $c_e(W)$ the walk of $c_e(G)$, obtained from W by removing every occurrence of e (here we regard walks as sequences of arcs). In the following, we regard c_e as a surjective function from the family of walks of G to the family of walks of $c_e(G)$.

Observation 2.3. *When e is a split-free or join-free arc, then c_e is a bijection when restricted to the closed (arc-covering) walks, or to the non-empty open walks of G whose first and last arc are different from e.*

Lemma 2.4. *Let e be a join-free arc of G. A walk W' of $c_e(G)$ is an omnitig of $c_e(G)$ if and only if there exists an omnitig W of G such that $W' = c_e(W)$.*

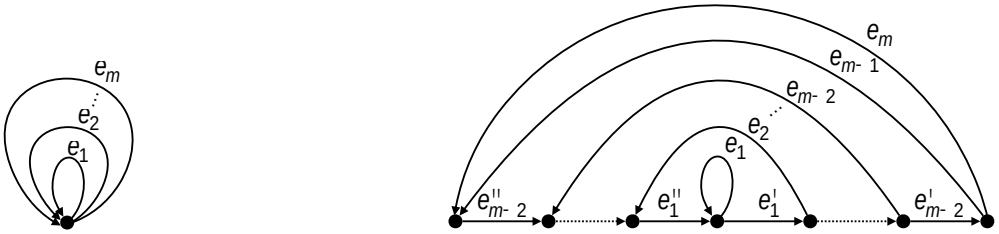

Figure 2.5: Left: A graph G made up of a single node and m 3 self-loops $e_1;:::;e_m$. Its m maximal omnitigs are $e_1;:::;e_m$. Right: The graph G^0 obtained from G by applying Transformation 1 and its symmetric transformation; the nodes of G^0 have in-degree and out-degree at most 2. Notice that the number of arcs of G^0 is $O(m)$. The m maximal omnitigs of G^0 are of the form $U(e_i) = e''_1 \quad e''_{i\ 1}e_ie'_{i\ 1} \quad e'_1$ (for $i\ 2\ f1;:::;mg$). Notice that their total length is (m^2), thus one cannot enumerate all maximal omnitigs of G^0 and convert these to maximal omnitigs of G. However, one can stop all univocal extensions of the arcs e_i when reaching arcs introduced by the transformations in G^0, see Section 2.4.1.

Proof. Consider the shortest walk $\underline{W}$ of G such that $W^0 = c_e(\underline{W})$. Notice that the first and last arc of $\underline{W}$ are different than e. Moreover, W^0 is an omnitig of $c_e(G)$ iff $\underline{W}$ is an omnitig of G. Indeed, for every circular arc-covering walk C of G it holds that C avoids $\underline{W}$ iff $c_e(C)$ avoids W^0. □

Corollary 2.4.1. *Let e be a join-free arc of G. A walk W^0 of $c_e(G)$ is a maximal omnitig of $c_e(G)$ if and only if there exists a maximal omnitig W of G such that $W^0 = c_e(W)$.*

Proof. Let W be a maximal omnitig of G. Then $c_e(W)$ is an omnitig of $c_e(G)$ by Lemma 2.4. Moreover, if W^0 was an omnitig of $c_e(G)$ strictly containing $c_e(W)$, then there would exist an omnitig $\overline{W}$ of G such that $W^0 = c_e(\overline{W})$, by Lemma 2.4. Clearly, $\overline{W}$ would contain W and contradict its maximality. Therefore, $c_e(W)$ is a maximal omnitig of $c_e(G)$.

For the converse, let W^0 be a maximal omnitig of $c_e(G)$. Let $\underline{W}$ be the shortest and unique minimal walk of G such that $W^0 = c_e(\underline{W})$. By Lemma 2.4, $\underline{W}$ is an omnitig of G. Let $\overline{W}$ be any maximal omnitig of G containing $\underline{W}$. We claim that $c_e(\overline{W}) = W^0 = c_e(\underline{W})$, which concludes the proof. If not, then $c_e(\overline{W})$ would strictly contain W^0 and contradict its maximality since also $c_e(\overline{W})$ would be an omnitig of $c_e(G)$ by Lemma 2.4. □

Finally, we can prove that the graph, obtained with Transformation 1, preserves the maximal omnitigs of the original graph, in the sense that no maximal omnitig is lost in the process.

Lemma 2.5. *Let G be a graph and let G^0 be the graph obtained by applying Transformation 1 to G. Then a walk W of G is a maximal omnitig of G if and only if there exists a maximal omnitig W^0 of G^0 such that W is the string obtained from W^0 by suppressing all the arcs introduced with the transformation.*

Proof. Notice that G is obtained by applying c_e to each arc e introduced by Transformation 1, that is, to each arc of G^0 that is not an arc of G. Notice that W is the string obtained from

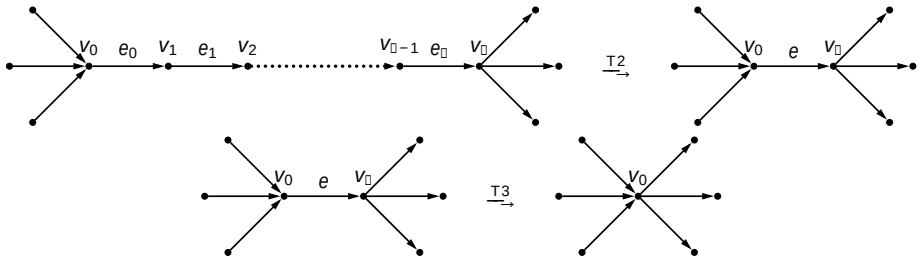

Figure 2.6: An example of Transformation 2 ($T2$) applied to the path $P = (v_0; e_0; \ldots; e_{\ell-1}; v_\ell)$, where $v_1; \ldots; v_{\ell-1}$ are biunivocal nodes and e is the new arc from v_0 to v_ℓ. The Transformation 3 ($T3$) compresses biunivocal arcs.

W' by suppressing all the arcs introduced with the transformation if and only if W is obtained from W' by contracting each arc e introduced by Transformation 1. Apply Corollary 2.4.1. □

2.2.2 Compression

We start by recalling the definition of compressed graph.

Definition 2.6 (Compressed graph). *A graph G is* compressed *if it contains no biunivocal nodes and no biunivocal arcs.*

To obtain a compressed graph, we introduce two transformations. The first one removes biunivocal nodes, by replacing those paths whose internal nodes are biunivocal with a single arc from the tail of the path to its head (see Figure 2.6 for an example).

Transformation 2. *Given G, for every longest path $P = (v_0; e_0; \ldots; e_{\ell-1}; v_\ell)$, $\ell \geq 2$, such that $v_1; \ldots; v_{\ell-1}$ are biunivocal nodes, we remove $v_1; \ldots; v_{\ell-1}$ and their incident arcs from G, and we add a new arc from v_0 to v_ℓ.*

This transformation is widely used in the genome assembly field, and it clearly preserves the maximal omnitigs of G: if $P = (v_0; e_0; \ldots; e_{\ell-1}; v_\ell); \ell \geq 2$ is a path where $v_1; \ldots; v_{\ell-1}$ are biunivocal nodes, in any closed arc-covering walk of G, whenever e_0 appears it is always followed by $e_1; \ldots; e_\ell$.

The last transformation contracts the biunivocal arcs of the graph (see Figure 2.6 for an example).

Transformation 3. *Given G, we contract every biunivocal arc e, namely we set $t(e') = t(e)$ for every out-going arc from $h(e)$ and remove the node $h(e)$.*

Also this transformation preserves the maximal omnitigs of G because every maximal omnitig which contains an endpoint of e, also contains e. Notice that after Transformations 2 and 3, the maximum in-degree and the maximum out-degree are the same as in the original graph.

2.3 Macronodes and macrotigs

In this section, unless otherwise stated, we assume that the input graph is *compressed*, in the sense that it has no biunivocal nodes and arcs. In some algorithms we will also require that the graph has constant in- and out-degree. In a compressed graph all arcs are split, join or bivalent; moreover, the following observation holds.

Observation 2.7. *Let G be a compressed graph. Let f and g be a join and a split arc, respectively, in G. The following holds:*

(i) *if fWg is a walk, then W has a node which is a bivalent node;*

(ii) *if gWf is a walk, then gWf contains a bivalent arc.*

The following lemmata brings new structural knowledge about omnitigs, which is going to be useful in proving some more advanced results.

Lemma 2.8. *Every maximal omnitig of a compressed graph contains both a join arc and a split arc. Moreover, it has a bivalent arc or an internal bivalent node.*

Proof. Consider an omnitig W composed only of split-free arcs. Notice first that W is a path. Consider any arc e, with $h(e) = t(W)$ and observe that eW is an omnitig, since the only outgoing arcs of internal nodes of eW are arcs of eW; thus there is no forbidden path between any two internal nodes of eW. Therefore, W is not a maximal omnitig. Symmetrically, no maximal omnitig is composed only of join-free arcs. This already implies the first claim in the statement: any maximal omnitig W contains at least one join arc f and at least one split arc g. If $f = g$ then W contains the bivalent arc f. Otherwise, either W contains a subwalk of the form $fW'g$ or it contains a subwalk of the form $gW'f$, where W' might be an empty walk. In the first case W has an internal node which is bivalent, by Observation 2.7(i). In the second case W contains a bivalent arc, by Observation 2.7(ii). □

Lemma 2.9. *Let e be a join or a split arc. No omnitig can traverse e twice.*

Proof. By symmetry, we only consider the case of two sibling split arcs g and g'. Since prefixes and suffixes of omnitigs are omnitigs, then a minimal violating omnitig would be of the form gZg, with $g \notin Z$. Since G is strongly connected, then there exists a simple cycle C of G with $g' \in C$ and with g' as its first arc. Notice that $g \notin C$, since C is simple. Consider then the first node u shared by both C and Z, and let e be the arc of C with $h(e) = u$. Clearly, $e \in Z$; in addition, $e \neq g$, since C is a path. Let C_u represent the prefix of C ending in u. Therefore, C_u is a forbidden path for the omnitig gZg, since it starts from $t(g) = t(g')$, with $g' \neq g$, and it ends in u with $e \in Z$. □

2.3.1 Macronodes

It is natural to identify a partition of the nodes of a compressed graph, where each class of such a partition (i.e. a *macronode*) contains precisely one bivalent node. We identify each class with the unique bivalent node they contain. Every other node that belongs to the same class are those that either reach the bivalent node with a join-free path or those that are reached by the bivalent node with a split-free path (recall Figure 2.1).

Definition 2.10 (Macronode). *Let v be a bivalent node of G. Consider the following sets:*

- $R^+(v) := \{u \in V(G) : \exists$ a join-free path from v to $u\}$;
- $R^-(v) := \{u \in V(G) : \exists$ a split-free path from u to $v\}$;

The subgraph M_v induced by $R^+(v) \cup R^-(v)$ is called the macronode centered in v.

Hence, the following result.

Lemma 2.11. *In a compressed graph G, the following properties hold:*

i) The set $\{V(M_v) : v$ is a bivalent node of $G\}$ is a partition of $V(G)$.

ii) In a macronode M_v, $R^+(v)$ and $R^-(v)$ induce two trees with common root v, but oriented in opposite directions. Except for the common root, the two trees are node-disjoint, all nodes in $R^-(v)$ being join nodes and all nodes in $R^+(v)$ being split nodes.

iii) The only arcs with endpoints in two different macronodes are bivalent arcs.

Proof. For i), by definition every node belongs to at least one macronode. Let u and v be distinct bivalent nodes and suppose for a contradiction that there exists $x \in V(M_u) \setminus V(M_v)$. W.l.o.g., assume x is a join node (the case where x is a split node is symmetric). By definition, $x \in R^-(u) \setminus R^-(v)$ holds. Let P_u and P_v be split-free paths from x to u and to v, respectively. Notice that x can not be a bivalent node, since otherwise from x no split-free path can start. Since the out-degree of x is one, P_u and P_v share a prefix of length at least one, but since u and v are distinct bivalent nodes, P_u and P_v differ by at least one arc. Let e be the first arc such that $e \in P_u$, but $e \notin P_v$, and let e' be its sibling arc, with $e' \notin P_u$, but $e' \in P_v$. Notice that $t(e) = w$ is a join node, since it belongs to split-free paths, but it also has out-degree two, since $w = t(e) = t(e')$; hence w is an internal bivalent node of split-free paths, a contradiction. Properties ii) and iii) trivially follow from the definition of macronode. □

To analyze how omnitigs can traverse a macronode and the degrees of freedom they have in choosing their directions within the macronode, we introduce the following definitions. In general, *central-micro omnitigs* are the smallest omnitigs that cross the center of a macronode. *Left- and right-micro omnitigs* start from a central-micro omnitig and proceed to the periphery of a macronode. Finally, we can combine left- and right-micro omnitigs into microtigs (which are not necessarily omnitigs themselves); recall Figure 2.2 for a visual reference. Formally:

Definition 2.12 (Micro omnitigs, microtigs). *Let f be a join arc and g be a split arc, such that fg is an omnitig.*

- *The omnitig fg is called a* central-micro omnitig.
- *An omnitig fgW (Wfg, resp.) that does not contain a bivalent arc as an internal arc is called a* right-micro omnitig *(respectively,* left-micro omnitig*).*
- *A walk $W = W_1fgW_2$, where W_1fg and fgW_2 are, respectively, a left-micro omnitig, and a right-micro omnitig, is called a* microtig.

We are now able to prove an helpful lemma to better understand the structure of omnitigs.

Lemma 2.13. *Let u be a bivalent node. No omnitig contains u twice as an internal node.*

Proof. Assume for a contradiction that there exists an omnitig W that contains u twice as internal node. Since u is an internal node of W, we can distinguish the case in which an omnitig contains twice a central-micro omnitig that traverses u, and the case in which an omnitig contains both the central-micro omnitigs that traverse u. In the first case, let fg be the central-micro omnitig of an omnitig W that traverses u. Notice that f is a join arc contained twice in W, contradicting Lemma 2.9. In the latter case, let f_1g_1 and f_2g_2 the two central-micro omnitigs that traverse u, with $f_1 \neq f_2$ and $g_1 \neq g_2$. Consider W to be a minimal violating omnitig of the form $f_1g_1\bar{W}f_2g_2$. Notice that $u \notin \bar{W}$, by minimality; hence $g_1\bar{W}f_2$ is a forbidden path, contradicting W being an omnitig. □

Given a join arc f, we first find central-micro omnitigs (of the type fg) with the generic function RightExtension($G; f; W$) from Algorithm 1, where W is a join-free path (possibly empty). This extension uses the following weak version of the Extension Property (since W is join-free). To build up the intuition, we also give a self-contained proof of this weaker result.

Lemma 2.14 (Weak form of the Extension Property (Theorem 2.29)). *Let fW be an omnitig in G, where f is a join arc and W is a join-free path. Then fWg is an omnitig if and only if g is the only arc with $t(g) = h(W)$ such that there exists a path from $h(g)$ to $h(f)$ in $G \setminus f$.*

Proof. To prove the existence of an arc g, which satisfies the condition, consider any closed path Pf' in G, where f' is an arbitrary sibling join arc of f. Notice that W is a prefix of Pf', since fW is an omnitig, since otherwise one can easily find a forbidden path for the omnitig fW as a subpath of Pf', from the head of the very first arc of Pf' that is not in W to $h(f')$. Therefore, let g be the the first arc of Pf' after the prefix W, in such a way that the suffix of Pf' starting from $h(g)$ is a path to $h(f)$ in $G \setminus f$.

For the direct implication, assume that there is a path P in $G \setminus f$ from $h(g')$, where g' sibling of g and $g' \neq g$, to $h(f)$. Then, this forbidden path P contradicts the fact that fWg is an omnitig.

For the reverse implication, assume that fWg is not an omnitig. Then take any forbidden path P for fWg. Since fW is an omnitig, P must start with some g' sibling arc of $g; g' \neq g$. Since W is join-free, then P must end in $h(f)$ with the last arc different from f. Therefore, P is a path from $h(g')$ to $h(f)$ in $G \setminus f$. □

Not only Lemma 2.14 gives us an efficient extension mechanism, but it also immediately implies the Y-intersection Property (for clarity of reusability, we state both its symmetric variants).

Corollary 2.14.1 (Y-intersection Property). *Let fWg be an omnitig, where f is a join arc, and g is a split arc.*

i) If W is a (possibly empty) join-free path, then for any g' a sibling split arc of g, the walk fWg' is not an omnitig.

ii) If W is a (possibly empty) split-free path, then for any f' a sibling join arc of f, the walk $f'Wg$ is not an omnitig.

We now use the Y-intersection Property to prove the X-intersection Property.

Theorem 2.15 (X-intersection Property). *Let v be a bivalent node. Let $f_1 \neq f_2$ be join arcs with $h(f_1) = h(f_2) = v$; let $g_1 \neq g_2$ be split arcs with $t(g_1) = t(g_2) = v$.*

i) If f_1g_1 and f_2g_2 are omnitigs, then $d^+(v) = d^-(v) = 2$.

ii) If f_1g_1 is an omnitig, then there are no omnitigs f_1g' with $g' \neq g_1$, nor $f'g_1$ with $f' \neq f_1$.

Proof. For point *i*), assume there exists an arc g_3, distinct from g_1 and g_2, such that $t(g_3) = v$. Consider any shortest closed path g_3P (with P possibly empty), which exists by the strong connectivity of G. Let f be the last arc of P. If $f \neq f_1$ then g_3P is a forbidden path for the omnitig f_1g_1, since $g_3 \neq g_1$. Otherwise, if $f = f_1$ then g_3P is a forbidden path for the omnitig f_2g_2, since $g_3 \neq g_1$. In both cases we reached a contradiction, therefore g_1 and g_2 are the only arcs in G with $t(g_1) = t(g_2) = v$. To prove that f_1 and f_2 are the only arcs in G with $h(f_1) = h(f_2) = v$ one can proceed by symmetry.

Point *ii*) follows from Corollary 2.14.1 (by taking the W path of its statement to be empty). □

Given an omnitig fg, we obtain the maximal right-micro omnitig with function MaximalRightMicroOmnitig$(G; f; g)$ from Algorithm 1. This works by extending fg, as much as possible, with the function RightExtension$(G; f; W)$ (where initially $W = g$). This extension stops when reaching the periphery of the macronode (i.e. a bivalent arc).

Algorithm 1: Functions RightExtension and MaximalRightMicroOmnitig.

```
1  Function RightExtension(G; f; W)
       Input  : The compressed graph G, fW omnitig with W join-free.
       Output : The unique arc e such that fWe is an omnitig, if it exists. Otherwise,
                nil.
2      S ← { e ∈ E(G) | t(e) = h(W) and there is a path from h(e) to h(f) in G \ {f} }
3      if there is exactly one arc e ∈ S then return e
4      return nil
5  Function MaximalRightMicroOmnitig(G; f; g)
       Input  : The compressed graph G, fg omnitig with f join arc and g split arc.
       Output : The path W such that fgW is a maximal right-micro omnitig.
6      W ← empty path
7      while True do
8          if fgW ends with a bivalent arc then return W
9          e ← RightExtension(G; f; W)
10         if e = nil then return W
11         W ← We
```

Lemma 2.16. *The functions in Algorithm 1 are correct. Moreover, assuming that the graph has constant degree, we can preprocess it in time $O(m)$ time, so that* RightExtension$(G; f; W)$ *runs in constant time, and* MaximalRightMicroOmnitig$(G; f; g)$ *runs in time linear in its output size.*

Proof. For RightExtension$(G; f; W)$, recall Lemma 2.14 and Theorem 2.2 and that the input graph is a compressed graph, and as such every node has constant degree.

For MaximalRightMicroOmnitig$(G; f; g)$, notice that every iteration of the *while* loop increases the output by one arc and takes constant time, since RightExtension$(G; f; W)$ runs in $O(1)$ time. □

Algorithm 2 is the procedure to obtain all maximal microtigs of a compressed graph. It first finds all central-micro omnitigs fg (with RightExtension$(G; f; ;)$), and it extends each to the right (i.e. forward in G) and to the left (i.e. forward in G^R) with MaximalRightMicroOmnitig.

In the following lemmata, we show some structural properties of central-micro omnitigs that are needed to prove the correctness of Algorithm 2, in Theorem 2.19.

Lemma 2.17. *Let fg be a central-micro omnitig. The following hold:*

i) There exists at most one maximal right-micro omnitig fgW, and at most one maximal left-micro omnitig Wfg.

ii) There exists a unique maximal microtig containing fg.

Algorithm 2: Function AllMaximalMicrotigs

```
1  Function AllMaximalMicrotigs(G)
       Input  : The compressed graph G.
       Output : All the maximal microtigs in G.
2      S ← ∅;
3      foreach bivalent node u in G do
4          foreach join arc f with h(f) = u do
5              foreach split arc g with t(g) = u do
6                  if g = RightExtension(G, f, ∅) then
                       . fg is a central-micro omnitig
                       . applied symmetrically for left- and right-micro omnitigs
7                      W1 ← MaximalRightMicroOmnitig(G^R, g, f)
8                      W2 ← MaximalRightMicroOmnitig(G, f, g)
9                      add W1^R fg W2 to S
10     return S
```

Proof. We prove only the first of the two symmetric statements in i). If g is a bivalent arc, the claim trivially holds by definition of maximal right-micro omnitig. Otherwise, a minimal counterexample consists of two right-micro omnitigs $fgPg_1$ and $fgPg_2$ (with P a join-free path possibly empty), with g_1 and g_2 distinct sibling split arcs. Since gP is a join-free path, the fact that both $fgPg_1$ and $fgPg_2$ are omnitigs contradicts the Y-intersection Property (Corollary 2.14.1).

For ii), given fg, by i) there exists at most one maximal left-micro omnitig W_1fg and at most one maximal right-micro omnitig fgW_2, as such there is at most one maximal microtig W_1fgW_2. □

Lemma 2.18. *Let e be an arc. The following hold:*

i) if e is not a bivalent arc, then there exists at most one maximal microtig containing e.

ii) if e is a bivalent arc, there exist at most two maximal microtigs containing e, of which at most one is of the form eW_1, and at most one is of the form W_2e.

Proof. By symmetry, in i) we only prove the case in which e is a split-free arc. Notice that by Lemma 2.11, $h(e)$ belongs to a uniquely determined macronode M_u of G; let P be the split-free path in G, from $h(e)$ to u. Let f be the last arc of eP ($f = e$ if P is empty). By the X-intersection Property (Theorem 2.15), there exists at most one split arc g with $t(g) = u = h(f)$ such that fg is an omnitig; if it exists, fg is a central-micro omnitig, hence by Lemma 2.17, there is at most one maximal left-micro omnitig Wfg. Finally, if such a maximal left-micro omnitig exists, ePg is a subwalk of Wfg, by the Y-intersection Property (Corollary 2.14.1). Otherwise, a minimal

counterexample consists of paths f_1Rg (subpath of ePg) and f_2Rg (subpath of Wfg), where $f_1 \neq f_2$ and R is a split-free path, since it is subpath of the split-free path eP; since both f_1Rg and f_2Rg are omnitigs, this contradicts the Y-intersection Property.

For ii), we again prove only one of the symmetric cases. The proof is identical to the above, since by Lemma 2.11, $h(e)$ belongs to a unique macronode M_{v_1} of G. As such, e belongs to at most one maximal microtig eW_1 in M_{v_1}. Symmetrically, $t(e)$ belongs to a uniquely determined macronode M_{v_2} of G. Thus, e belongs to at most one maximal microtig W_2e within M_{v_2}. □

We conclude the section stating an important result on maximal microtigs.

Theorem 2.19 (Maximal microtigs). *The maximal microtigs of any strongly connected graph G with n nodes, m arcs, and arbitrary degree have total length $O(n)$, and can be computed in time $O(m)$.*

Proof. First we prove the $O(n)$ bound on the total length. As we explain in Section 2.2 we can transform G into a compressed graph G' such that G' has $n' \leq n$ nodes and $m' \leq m$ arcs.

Since G' has at most n' macronodes (recall that macronodes partition the node set, Lemma 2.11), and every macronode has at most two maximal microtigs, then number of maximal microtigs is at most $2n'$. The total length of all maximal microtigs is bounded as follows. Every internal arc of a maximal microtig is not a bivalent arc, by definition. Since every non-bivalent arc appears in at most one maximal microtig (Lemma 2.18), and there are at most n' non-bivalent arcs in any graph with n' nodes, then the number of internal arcs in all maximal microtigs is at most n'. Summing up for each maximal microtigs its two non-internal arcs (i.e., its first and last arc), we obtain that the total length of all maximal microtigs is at most $2n' + n' = 3n'$, thus $O(n)$.

As mentioned, in Section 2.2 we show how to transform G into a compressed graph G' with $O(m)$ arcs, $O(m)$ nodes, and constant degree. On this graph we can apply Algorithm 2. Since every node of the graph has constant degree, the *if* check in Line 6 runs a number of times linear in the size $O(m)$ of the graph. Checking the condition in Line 6 takes constant time, by Lemma 2.16; in addition, the condition is true for every central-micro omnitig fg of the graph. The *then* block computes a maximal microtig and takes linear time in its size, Lemma 2.16. By Lemma 2.18 we find every microtig in linear total time. □

2.3.2 Macrotigs

In this section we analyze how omnitigs go from one macronode to another. Macronodes are connected with each other by bivalent arcs (Lemma 2.11), but merging microtigs on all possible bivalent arcs may create too complicated structures. However, this can be avoided by a simple classification of bivalent arcs: those that connect a macronode with itself (*self-bivalent*) and those that connect two different macronodes (*cross-bivalent*), recall Figure 2.1.

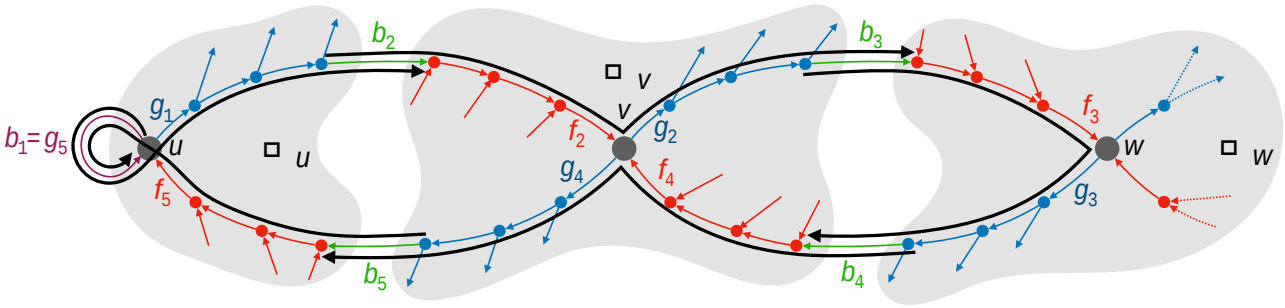

Figure 2.7: Three macronodes M_u; M_v; M_w (as gray areas) with arcs color-coded as in Figure 2.1. Black walks mark their ve maximal microtigs: $b_1g_1 : : : b_2$, $b_i : : : f_ig_i : : : b_{i+1}$ (i 2 f2; 3; 4g), $b_5 : : : f_5g_5$ ($g_5 = b_1$). The maximal macrotig M is obtained by overlapping them on the cross-bivalent arcs $b_2; b_3; b_4; b_5$, i.e. $M = b_1 : : : b_2 : : : b_3 : : : b_4 : : : b_5 : : : b_1$. Notice that no arc is contained twice in M, with the exception of the self-bivalent arc b_1, appearing as the rst and last arc of M.

De nition 2.20 (Self-bivalent and cross-bivalent arcs). *A bivalent arc b is called a* self-bivalent arc *if $U(b)$ goes from a bivalent node to itself. Otherwise it is called a* cross-bivalent arc.

A macrotig is now obtained by merging those microtigs from different macronodes which overlap only on a cross-bivalent arc, see also Figure 2.7.

De nition 2.21 (Macrotig). *Let W be any walk. W is called a* macrotig *if*

1. *W is an microtig, or*
2. *By writing $W = W_0b_1W_1b_2 : : : b_k \ _1W_k \ _1b_kW_k$, where $b_1; : : : ; b_k$ are all the internal bivalent arcs of W, the following conditions hold:*
   (a) *the arcs $b_1; : : : ; b_k$ are all cross-bivalent arcs, and*
   (b) *W_0b_1; $b_1W_1b_2$; : : : ; $b_k \ _1W_k \ _1b_k$; b_kW_k are all microtigs.*

Notice that the above definition does not explicitly forbid two different macrotigs of the form W_0bW_1 and W_0bW_2. However, Lemma 2.18 shows that there cannot be two different microtigs bW_1 and bW_2. Thus, our structural results (including Lemmas 2.22 and 2.24 below) show that we can construct all *maximal* macrotigs by repeatedly joining microtigs overlapping on cross-bivalent arcs, as long as possible, and obtain Theorem 2.25.

Lemma 2.22. *For any macrotig W there exists a unique maximal macrotig containing W.*

Proof. W.l.o.g., a minimal counterexample consists of a non-right-maximal macrotig Wb, such that there exist two distinct microtigs bW_1 and bW_2 (notice that b is a cross-bivalent arc). By Lemma 2.18 applied to b, we obtain $bW_1 = bW_2$, a contradiction. □

However, the macrotig definition does not forbid a cross-bivalent arc to be used twice inside a macrotig. In Lemma 2.24 below, we prove that also this is not possible, using the following result.

Lemma 2.23 ([14]). *For any two distinct non-sibling split arcs $g; g'$, write $g \;\; g'$ if there exists an omnitig gPg' where P is split-free. Then, the relation $\;\;$ is acyclic.*

Lemma 2.24. *Let W be a macrotig and let e be an arc of W. If e is self-bivalent, then e appears at most twice in W (as first or as last arc of W). Otherwise, e appears only once.*

Proof. If e is self-bivalent, then Definition 2.21 implies that e is either the first arc of W, the last arc of W, or both. Thus, e appears at most twice.

Suppose now that e is not self-bivalent. We first consider the case when e is a split arc. We are going to prove that any two consecutive non-self-bivalent split arcs are in relation $\;\;$ from Lemma 2.23. Indeed, let g and g' be two consecutive (i.e. closest distinct) non-self-bivalent split arcs along W: that is, gPg' subwalk of W, with P a split-free path. Notice that g and g' are not sibling arcs, since otherwise, g is a self-bivalent arc, by Observation 2.7. If $t(g')$ is not a bivalent node, then P is empty. In this case, g is a join-free arc, so gg' is an omnitig; as such, $g \;\; g'$. Otherwise, if $t(g')$ is a bivalent node, then gPg' is a left-micro omnitig and so it is an omnitig; as such, again, $g \;\; g'$.

Suppose for a contradiction that e is traversed twice. Since there are no internal self-bivalent arcs (as argued at the beginning of the proof), this would result in a cycle in the relation $\;\;$, which contradicts Lemma 2.23.

When e is a non-self-bivalent join arc, we proceed symmetrically. First, notice that the relation defined in Lemma 2.23 is symmetric: if f and f' are two distinct non-sibling join arcs such that fPf', with P a join-free path, then $f \;\; f'$. The claim above can be symmetrically adapted to hold for any two closest distinct non-self-bivalent join arcs f and f' within a macrotig (i.e. corresponding to a subwalk of W of the form fPf', with P a join-free path). Moreover, f and f' are not siblings; since otherwise, f' is a self-bivalent arc, by Observation 2.7.

Hence, by the acyclicity property of the relation $\;\;$ on the reverse graph, the claim also holds for non-self-bivalent join arcs. □

Therefore, we can construct all *maximal* macrotigs by repeatedly joining microtigs overlapping on cross-bivalent arcs, as long as possible, as in Algorithm 3.

Similarly to what we obtained in Theorem 2.19, we conclude this section with an important result concerning macrotigs of a graph.

Theorem 2.25 (Maximal macrotigs). *The maximal macrotigs of any strongly connected graph G with n nodes, m arcs, and arbitrary degree have total length $O(n)$, and can be computed in time $O(m)$.*

Proof. By Theorem 2.19, G has $O(n)$ maximal microtigs, of total length $O(n)$. By Lemma 2.24, every maximal microtig is contained in a unique maximal macrotig (and it appears only once inside such a macrotig), and the length of each maximal macrotig is at most the sum of the

Algorithm 3: Function AllMaximalMacrotigs.

```
1 Function AllMaximalMacrotigs(G)
      Input  : The compressed graph G.
      Output : All the maximal macrotigs in G.
2     S ← AllMaximalMicrotigs(G)
3     while ∃ W1b ∈ S and bW2 ∈ S with b cross-bivalent arc and non-empty W1; W2 do
4         remove W1b and bW2 from S
5         add W1bW2 to S
6     return S
```

lengths of its maximal microtigs; thus, we have that the total length of all maximal macrotigs is at most $O(n)$.

Using Algorithm 2, we can get all the $O(n)$ maximal microtigs of G in time $O(m)$ (Theorem 2.19). Once we have them, we can easily implement Algorithm 3 in $O(m)$-time. The correctness of this algorithm is guaranteed by Lemma 2.24. □

2.4 Maximal omnitig representation and enumeration

In the algorithms of this section we assume that the graph has constant degree, and we explain in Section 2.4.1 how to handle the non-constant degree case.

We begin by proving the first part of Theorem 2.1. Theorem 2.25 guarantees that the total length of maximal macrotigs is $O(n)$. Thus, it remains to prove the following lemma, since since any macrotig is a subwalk of a maximal macrotig (Lemma 2.22).

Theorem 2.26 (Maximal omnitig representation)**.** *Let W be a maximal omnitig.*

i) If W contains an internal bivalent node, then W is of the form $U(fW'g)$, where f is the first join arc of W, $g \neq f$ is the last split arc of W, and W' is a possibly empty walk. Moreover, $fW'g$ is a macrotig.

ii) Otherwise, W is of the form $U(b)$, where b is a bivalent arc, and b does not belong to any macrotig.

Proof. To prove *i*), let u be an internal bivalent node of W, and let f_u and g_u be, respectively, the join arc and the split arc of W with $h(f_u) = u = t(g_u)$; both such f_u and g_u exist, since u is an internal node of W. Therefore, since W contains at least f_u and g_u, let f and g be, respectively the first join arc and the last split arc of W. Observe that f is either f_u or it appears before f_u in W; likewise, g is either g_u or it appears after g_u in W. Thus, f comes before g, and we can write $W = W\ \ fW'gW^+$, where W' is the subwalk of W, possibly empty, from $h(f)$ to $t(g)$. Therefore, by the maximality of W, we have $W = W\ \ fW'gW^+ = U(fW'g)$.

To prove that the subwalk $fW'g$ of W is a macrotig, we prove by induction that *any* walk of the form $fW'g$, where f is a join arc and g is a split arc, is a macrotig. The induction is on the length of W'.

Case 1: W' contains no internal bivalent arcs. Since $fW'g$ contains a bivalent node (Observation 2.7), it is of the form $fW'g = W'_1f'g'W'_2$, with $h(f') = t(g') = u$ bivalent node. Notice that $W'_1f'g'W'_2$ is an microtig and thus it is a macrotig, by definition.

Case 2: $fW'g$ contains an internal bivalent arc b, i.e. $fW'g = W'_1bW'_2$, with $W'_1; W'_2$ non empty. By induction, W'_1b and bW'_2 are macrotigs and both contain a bivalent node as internal node. Suppose b is a self-bivalent arc, then both W'_1b and bW'_2 would contain the same bivalent node as internal node, contradicting Lemma 2.13. Thus, b is a cross-bivalent arc and $W'_1bW'_2$ is also a macrotig, by definition.

For ii), notice that if W contains no internal bivalent node then it contains a unique bivalent arc b, by Lemma 2.8 and Observation 2.7. Thus, by the maximality of W, it holds that $W = U(b)$. It remains to prove that there is no macrotig containing b.

Suppose for a contradiction that there is a maximal left-micro omnitig M containing b. By definition, M is of the form $bW_Mf_Mg_M$. Notice that $h(W) = t(g_M)$, by univocal extension of b, and, as such, Wg_M is a walk. In addition, Wg_M is an omnitig, since M is an omnitig and the arcs of W before b are join-free, therefore Wg_M can have no forbidden path. This contradicts the fact that W is maximal.

Symmetrically, we have that there is no maximal right-micro omnitig containing b. Thus, by definition, b appears in no microtig, and thus in no macrotig. □

Remark 2.27. *The number of maximal omnitigs containing an internal bivalent node is $O(n)$. This follows by Theorem 2.26(i), by maximality, and by the fact that the total length of maximal macrotigs is $O(n)$ (Theorem 2.25).*

Next, we are going to prove the second, algorithmic, part of Theorem 2.1. To describe the algorithm that identifies all maximal omnitigs (Algorithm 5), we first introduce an auxiliary procedure (Algorithm 4), which is a corollary of the Extension Property (Theorem 2.29) and of Theorem 2.2, to find the unique possible extension of an omnitig. By Theorem 2.25, we can compute the maximal macrotigs of G in time $O(m)$. We can trivially obtain in $O(m)$ time the set F of arcs not appearing in the maximal macrotigs. It remains to show how to obtain the subwalks of the maximal macrotigs univocally extending to maximal omnitigs. We first prove an auxiliary lemma needed for the proof of the Extension Property.

Lemma 2.28. *Let fW be an omnitig, where f is a join arc. Let P be a path from $t(P) = h(W)$ to a node in W, such that the last arc of P is not an arc of fW. Then no internal node of P is a node of W.*

Algorithm 4: Function IsOmnitigRightExtension

```
1 Function IsOmnitigRightExtension(G; f; g)
      Input  : The compressed graph G. A join arc f and a split arc g such that there
               exists a walk fWg where fW is an omnitig.
      Output : Whether fWg is also an omnitig.
2     S ← {g' ∈ E(G) | t(g') = t(g) and there is a path from h(g') to h(f) in G − {g}}
3     return True if S = {g} and False otherwise
```

Proof. Let P_W be the longest suffix of P such that no internal node of P_W is a node of W. If $P_W = P$, the lemma trivially holds. Let now $W = (u_0, e_1, u_1, e_2, \dots, e_k, u_k)$. Let $u_i = t(P_W)$ and $u_j = h(P_W)$. If $i \geq j$, then P_W is a forbidden path for fW, a contradiction. Hence, assume $i < j < k$. Let $f'WQ$ be a closed path. Consider the walk $Z = P_W e_{j+1} \dots e_k Q$. Notice that $e_{i+1} \in Z$ and $f \in Z$. Thus Z can transformed in a forbidden path for fW, from u_i to $h(f)$. □

Theorem 2.29 (Extension Property). *Let fW be an omnitig in G, where f is a join arc. Then fWg is an omnitig if and only if g is the only arc with $t(g) = h(W)$ such that there exists a path from $h(g)$ to $h(f)$ in $G - f$.*

Proof. As seen in the proof Lemma 2.14, at least one g exists which satisfies the condition. Assume g is a split arc, otherwise the statement trivially holds.

First, assume that there is a g' sibling split arc of g and a path P from $h(g')$ to $h(f)$ in $G - f$. We prove that there exists a forbidden path for fWg. Let P_W be the prefix of P ending in the first occurrence of a node in W (i.e., no node of P_W belongs to W, except for $h(P_W)$). Notice that $g'P_W$ is a forbidden path for the omnitig fWg (it is possible, but not necessary, that $h(P_W) = h(f)$).

Second, take any forbidden path P for the omnitig fWg. We prove that there exists a g' sibling split arc of g and a path from $h(g)$ to $h(f)$ in $G - f$. Notice that $t(P) = h(W) = t(g)$, otherwise P would be a forbidden path for fW. As such, P starts with a split arc $g' \neq g$ and, by Lemma 2.28, P does not contain f. Thus, the suffix of P from $h(g')$ is a path in $G - f$ from $h(g')$ to $h(f)$. □

Corollary 2.29.1. *Algorithm 4 is correct. Moreover, if the graph has constant degree, we can preprocess it in time $O(m)$ time, so that Algorithm 4 runs in constant time.*

Maximal omnitigs are identified with a two-pointer scan of maximal macrotigs (Algorithm 5): a left pointer always on a join arc f and a right pointer always on a split arc g, recall Figure 2.3. For completeness, Algorithm 5 also outputs the maximal omnitigs.

Theorem 2.30 (Maximal omnitig enumeration). *Algorithm 5 is correct and, if the compressed graph has constant degree, it runs in time linear in the size of the graph and of its output.*

Algorithm 5: Computing all maximal omnitigs

Input : The compressed strongly connected graph G of constant degree.
Output : All maximal omnitigs of G.

. Assume that AllMaximalMacrotigs(G) returns all the maximal macrotigs in G.
. Recall that $U(W)$ is the univocal extension of the walk W.

1 $B \leftarrow \{b \mid b$ bivalent arc that does not occur in any $W \in$ AllMaximalMacrotigs(G)$\}$
2 **foreach** $b \in B$ **do output** $U(b)$
3 **foreach** $fXg \in$ AllMaximalMacrotigs(G) **do**
. With the notation $X[f::g]$, we refer to the subwalk of fXg starting with the occurrence of f in fX (unique by Lemma 2.24) and ending with the occurrence of g in Xg (unique by Lemma 2.24).
4 $f \leftarrow f$, $g \leftarrow$ nil, $g' \leftarrow$ first split arc in Xg
5 **while** $g' \neq$ nil **do**
6 **while** $g' \neq$ nil and IsOmnitigRightExtension($f; g'$) **do**
. Grow $X[f::g]$ to the right as long as possible
7 $g \leftarrow g'$
8 $g' \leftarrow$ next split arc in Xg after g
. $X[f::g]$ cannot be grown to the right anymore
9 **output** $U(X[f::g])$
10 **while** $g' \neq$ nil and not IsOmnitigRightExtension($f; g'$) **do**
. Shrink $X[f::g]$ from the left until it can be grown to the right again
11 $f \leftarrow$ next join arc in fX after f

Proof. By Theorem 2.26, any maximal omnitig W is either of the form $U(fW'g)$ (where $fW'g$ is a macrotig, and thus also a subwalk of a maximal macrotig, by Lemma 2.22), or of the form $W = U(b)$, where b is a bivalent arc not appearing in any macrotig. In the latter case, such omnitigs are outputted in Line 2. In the former case, it remains to prove that the external *while* cycle outputs all the maximal omnitigs of the form $U(fW'g)$ where $fW'g$ is contained in a maximal macrotig fXg. At the beginning of the first iteration, $W = U(X[f::g'])$ is left-maximal since $f = f$. The first internal *while* cycle ensures that $W = U(X[f::g])$ is also right-maximal, at which point it is printed in output. Then, the second internal *while* cycle ensures that $W = U(X[f::g'])$ is a left-maximal omnitig, and the external cycle repeats.

For the running time analysis, note that AllMaximalMacrotigs(G) runs in $O(m)$ time, by Theorem 2.25. Each iteration of the *foreach* cycle takes time linear in the total size of the maximal macrotig X and of its output (by Corollary 2.29.1), and that the total size of all maximal macrotigs is linear, by Theorem 2.25. □

2.4.1 Maximal omnitig enumeration for non-constant degree

Given the input strongly connected graph G with m arcs, and non-constant degree, denote by G' the graph with constant in-degree and out-degree obtained by applying Transformation 1 and its symmetric transformation. The trivial strategy to obtain the set of maximal omnitigs of G, given the set of maximal omnitigs G', is to:

1. Contract in the maximal omnitigs all the arcs which were introduced by Transformation 1.
2. Remove any duplicate omnitig which may occur due to this contraction (i.e., two different maximal omnitigs in G' which result in the same walk in the G, after the contraction).

In general, the above procedure may require more than linear time in the final output size, recall Figure 2.5.

We avoid this, as follows. Let M and M' denote the set of maximal macrotigs of G and G', respectively, and let F and F' denote the set of bivalent arcs not appearing in any macrotig, of G and G', respectively (recall Theorem 2.1).

First, since G' has $O(m)$ nodes and arcs, by Transformation 1, then also the maximal macrotigs M' have total length $O(m)$, and both M' and F' can be obtained in $O(m)$ time (Theorem 2.25). From M', one can obtain M in time $O(m)$, by contracting the arcs introduced by the transformation. However, while contracting such arcs, we must keep track of the pair of arcs $(f;g)$ corresponding to maximal omnitigs, as follows.

We modify Algorithm 5 to also report, for each macrotig X' of G' and for each maximal omnitig of the form $U(X'[f::g])$ (in the order they were generated by the algorithm), the indexes of the arcs f and g in X'. We now contract the arcs of X' by removing from X' every occurrence of the arcs introduced by the transformation, and updating the indexes of f and g so that they still point at the first and last arc of the walk obtained from $X'[f::g]$, after the contraction. Second, to avoid duplicates, we scan the pair of indexes of f and g along each macrotig, and remove any duplicated pair (if duplicates are present, they must occur consecutively, and thus they can be removed in linear time).

Second, the transformations do not introduce bivalent arcs, thus $F = F'$. This also implies that the arcs introduced by the transformation appear either inside macrotigs, or inside univocal extensions $U(\,)$. Having the set of maximal macrotigs M and the new arc pairs $(f;g)$ inside the maximal macrotigs in M, it now suffices to perform the univocal extensions $U(\,)$ inside the original graph G.

Chapter 3

The Hydrostructure: a Universal Framework for Safe and Complete Algorithms for Genome Assembly

3.1 Introduction

The Hydrostructure as a Universal Framework for safety. In this chapter, we show how to obtain safe and complete algorithms for a plethora of *natural* genome assembly problems, as stated in Definitions 1.2 and 1.5 and Remark 1.6, with an entirely *new perspective*, and a *universal framework* for safety, the *hydrostructure* of a walk (see Section 3.2 for a detailed definition).

The hydrostructure gives a more structured view of the graph from the perspective of the walk on which the hydrostructure is built, allowing for simple safety characterization for *all* models (see Figure 3.1 and [13]). We characterise, as it is the main focus of thesis, k-circular safe walks for any given k (where $1 < k < 1$), which was mentioned as an open problem in [2], and prove the equivalence of the k-circular safe problems for all k 2. The Hydrostructure can also be used to for *single and multiple linear* models (mentioned as open problems in [67]), as shown in [13], an overview is given in Figure 3.1. Moreover, the *exibility* of the hydrostructure allows us to generalize these to the more practically relevant *subset covering* and *visibility* models.

Even though the hydrostructure is developed mainly for such arc-covering models, it reveals a novel fundamental insight into the structure of a graph, which may be of independent interest. It captures the bridge-like characteristic of a walk for characterizing safety. This distinguishes the core of a safe walk from its trivially safe extensions on both sides, i.e., along the only way to reach it or its *left wing* (if exists), and along the only way forward from it or its *right wing* (if exists).

Its water-inspired terminology stems from the standard notions of “source” and “sink”

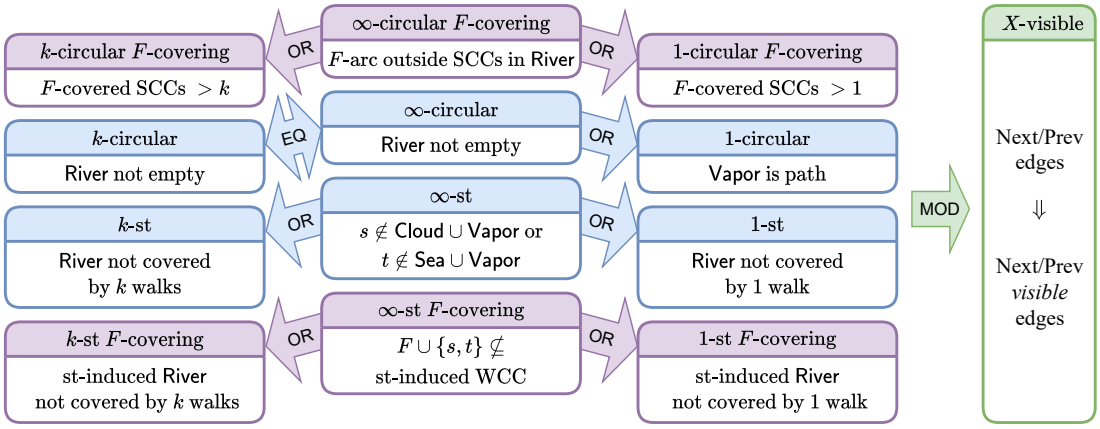

Figure 3.1: [Taken from [13]] An overview of the characterizations of safety for non-trivial walks in our various models. Relationships among the criteria for characterization of di erent models are depicted using, (i) EQ implying equivalent criteria, (ii) OR implying criteria is an OR of itself with that of the source, or (iii) MOD implies the criteria is modi ed from the source as described. For simplicity of presentation, the criteria for trivial walks and wings (for linear models) are not included, despite being simple criteria based on hydrostructure. Note that all characterizations imply that the Vapor is a path, and hence include being 1-circular safe.

nodes, but in the present case of a strongly connected graph now mimics the water cycle on Earth. The hydrostructure distinguishes the source part as Sea and the target part as Cloud. The internal part of the bridge-like walk forms the Vapor, which is the only way water moves from Sea to Cloud. On the other hand, water can easily move from Cloud to Vapor and from Vapor to Sea. The residual graph forms the River, which in general serves as an alternate route for the water from Cloud to Sea. For non-bridge-like walks the hydrostructure trivially reduces to Vapor being the entire graph, and the remaining components empty. For any walk, we show that its hydrostructure can be easily computed in *linear* time, by evaluating the restricted reachability of the end points of the walk.

Simple characterizations for existing and new problems. For single circular walks, the previous characterizations identified forbidden paths [67, 14] resulting only in a NO certificate. Our characterization simplifies it to merely Vapor not being the entire graph, which basically signifies that the walk is bridge-like, resulting in both YES and NO certificates which are easily verifiable. For *1* -circular safe walks, the previous characterization further required the presence of a *certi cate arc* [2] forbidding cycles of certain types, resulting in a NO certificate which is even harder to verify. Our characterization only adds an additional constraint of having a non-empty River, drastically simplifying the characterization. Note that it is the same for any given number *k* 2 of closed walks, proving their equivalence.

In the subset covering model, the previous characterizations are easily extendable using more concrete criteria involving the corresponding subset *F*. In the subset covering model, the hydrostructure itself is generalized for the corresponding subset *F*, so that the existing characterizations can be directly applied without any changes. Finally, for all these problems,

Safety Problems	Previous Results		New Results		
	Verify	Enumerate	Verify	Enum. Trivial	Enum. Improved
1-circular	$O(mn)$	$O(m^2n)$ [67]	$O(m)$	$O(m^2n)$	-
	$O(mn)$	$O(mn)$ [14]			
	$O(m)$	$O(m + o)$ Chapter 2			
k-circular	-	-	$O(m)$	$O(m^2n)$	$O(mn)$
1-circular	$O(mn)$	$O(m^2n)$ [2]	$O(m)$	$O(m^2n)$	$O(mn)$

Table 3.1: A comparison of the previous results for safety problems with the new results trivially obtained from the characterization, and the improved results. The size of the output is represented by o. The optimal algorithms are marked by . Optimality is obtained and proved thanks to the result in section *Incremental Computation of the Hydrostructure*, in [13]. All the above models can also be extended to *subset covering* (see Section 3.4) using the same bounds and to *subset visibility* (see Section 3.5) with an additive $O(mn)$ term.

the hydrostructure itself (or in some cases with additional walk cover[1]) serves as both the YES and NO certificate.

The simplicity of our characterization motivates us to introduce the problem of *verifying* whether a given walk is safe in a model. Despite this problem not being explicitly studied earlier, the previous characterizations [67, 14, 2] resulted in $O(mn)$ time verification algorithms, which again depend on the corresponding certificate for the model. In Chapter 2, we presented an algorithm which can be adapted to a linear-time verification algorithm for 1-circular safe walks, but it uses complex data structures.

Our characterizations (proved in [13], section *Incremental Computation of the Hydrostructure*) are directly adaptable to linear time *optimal* verification algorithms for almost all of our models, using simple techniques such as graph traversal. See Table 3.1 for a comparison.

Novel techniques to develop optimal enumeration algorithms. Our verification algorithms can also be adapted to simple $O(m^2n)$ time algorithms to enumerate all the maximal safe walks. Further, using the optimal 1-circular safe algorithm [14] and our new characterization, we not only improve upon [2] for 1-circular walks, but also make our k-circular safe algorithm *optimal*. In order to improve our linear algorithms, in [13], we presented a *novel technique* to compute the hydrostructure for all subwalks of a *special* walk using incremental computation. Since the linear characterizations are built on top of the 1-circular safe walks, we use the concise representation of 1-circular safe walks in $O(n)$ special walks, as seen in Chapter 2. At its core, the incremental computation uses the incremental reachability algorithm [35] requiring a total $O(m)$ for each special walk. Surprisingly every node and arc can enter and leave River exactly once during the incremental computation, allowing its maintenance in total $O(m)$ time.

Summarising, the hydrostructure is a *mathematical tool* which is *exible* for addressing a variety of models (handling different types of genomes, errors, complex or unsequenced regions),

[1]For simplicity we use the walk cover (a NO-certi cate) in our characterization, which have a simple equivalent YES-certi cate using *maximum arc antichain*

and is *adaptable* to develop simple yet efficient algorithms. Thus, we believe the various results shown in this chapter can form the theoretical basis of future *complete* genome assemblers.

3.2 Hydrostructure

The hydrostructure of a walk partitions the strongly connected graph from the perspective of the walk; for a *bridge-like* walk it identifies two parts of the graph separated by the walk, and gives a clear picture of the reachability among the remaining parts. This allows an easy characterization in problems (such as safety) that inherently rely on reachability. It is defined using the *restricted forward and backward reachability* for a walk, as follows.

Definition 3.1. *The* restricted forward and backward reachability *of a walk W is defined as*

- $R^+(W) = \{x \in G \mid \exists$ START(W)*-x walk W' s.t. W is not a subwalk of $W'\}$,*
- $R^-(W) = \{x \in G \mid \exists\, x$*-*END$(W)$ *walk W' s.t. W is not a subwalk of $W'\}$.*

By definition, all nodes and arcs of W (except for its last arc END(W), since by our convention the walks start and end in an *arc*), are always in $R^+(W)$ (and symmetrically for $R^-(W)$). Further, if a walk is not bridge-like, we additionally have END$(W) \in R^+(W)$ as we will shortly prove, and $R^+(W)$ extends to the entire graph G due to strong connectivity. Thus, inspired by the similarity between the water cycle on Earth and the reachability among the four parts of the Venn diagram of $R^+(W)$ and $R^-(W)$ (see Figure 3.2a), we define the hydrostructure of a walk as follows.

Definition 3.2 (Hydrostructure of a Walk)**.** *Let W be a walk with at least two arcs.*

- $\mathsf{Sea}(W) = R^+(W) \setminus R^-(W)$.
- $\mathsf{Cloud}(W) = R^-(W) \setminus R^+(W)$.
- $\mathsf{Vapor}(W) = R^+(W) \setminus R^-(W)$.
- $\mathsf{River}(W) = G \setminus (R^+(W) \cup R^-(W))$.

Notice that the hydrostructure is defined on walks consisting of at least two arcs: it is, indeed, meant to answer safety queries and, generally speaking, every arc is safe by definition (this is not true in some models, e.g., as in the F-visible model, but a single arc query in the F-model is easy to solve nonetheless). The reachability among the parts of the hydrostructure mimics the water cycle, which we describe as follows (formally proved later). For any node or arc in the Sea, the only way to reach the Cloud is by traversing the walk through the entire Vapor (similar to the evaporation process), as it is shared by both $R^+(W)$ and $R^-(W)$. Thus, for a bridge-like walk W, the hydrostructure exposes the Sea and the Cloud that are separated by W in the Vapor.

On the other hand, a node or an arc from the Cloud can directly reach the Vapor (by dissipation), being in $R^-(W)$, and from the Vapor it can directly reach the Sea (by condensation),

being in $R^+(W)$. Finally, the River acts as an alternative path from the nodes and arcs in the Cloud to the Sea (by rainfall), since the forward reachability from the Cloud and the backward reachability from the Sea are not explored in $R^+(W)$ and $R\ (W)$.

Properties. The strong connectivity of the graph results in the hydrostructure of an *avertible* walk to have the entire graph as the Vapor, whereas the Sea, Cloud, and River are empty. On the other hand, for a bridge-like walk, the Vapor is exactly the internal path of the walk, resulting in the following important property for any walk $W = aZb$.

Lemma 3.3. *For a walk $W = aZb$, $\mathsf{Vapor}(W)$ is the open path Z i W is bridge-like, otherwise $\mathsf{Vapor}(W)$ is G.*

Proof. We decompose W into aZb, where a and b are arcs. We first prove that if $\mathsf{Vapor}(aZb)$ is not exactly the open path Z, then $\mathsf{Vapor}(aZb) = G$ and aZb is avertible. Assume that Z is not an open path, or $\mathsf{Vapor}(aZb)$ contains a node or arc $x \in Z$. We distinguish three cases:

- If Z is not an open path, then it contains a cycle which can be removed to create a walk from a to b not containing aZb.
- If $x = a$ (or $x = b$) then by definition $R\ (aZb)$ (or $R^+(aZb)$) contains a walk from a to b that does not contain aZb.
- If $x \in \{a, b\}$, then by definition of $R^+(aZb)$ and $R\ (aZb)$, there are paths from a to x and from x to b that do not contain aZb. And since x is not in aZb, these paths can be joined together to create a walk from a to b that does not contain aZb.

In each case, there is a walk from a to b that does not contain aZb. And since b is the last element of this walk, by strong connectivity this walk can be extended within G to reach any node or arc in $G \setminus Z$ without containing aZb. By definition, the walk Z is in $\mathsf{Vapor}(aZb)$ as well. So $R^+(aZb) = G$, and by symmetry one can prove that $R\ (aZb) = G$, so $\mathsf{Vapor}(aZb) = G$. This also makes aZb avertible, since, for any pair $x, y \in G$, every occurrence of aZb in an x-y walk can be replaced with an a-b walk that does not contain aZb.

We now prove that if $\mathsf{Vapor}(aZb)$ is an open path equal to Z, then aZb is *bridge-like*. Since Z is an open path, $b \notin Z$ else it would repeat the node $\textsc{tail}(b)$. However, $b \in R\ (aZb)$ by definition, so we have $b \notin R^+(aZb)$, which means aZb is bridge-like because of the pair (a, b). □

Note that the hydrostructure partitions not just the nodes but also the arcs of the graph. The strong connectivity of the graph and the definitions above result in the following properties of the parts of the hydrostructure (see Figure 3.2b for the SCCs, and Figure 3.2a for reachability).

Lemma 3.4. *The hydrostructure on a bridge-like path aZb exhibits the following properties:*

a. (Safety) *Every walk from $x \in \mathsf{Sea}(aZb)$ to $y \in \mathsf{Cloud}(aZb)$ contains aZb as a subwalk.*

b. (Avoidance) *Every $x \in \mathsf{Cloud}(aZb)$ and $y \in \mathsf{Sea}(aZb)$ have an x-y walk with no subwalk aZb.*

c. (Separation) *a 2* Sea(*aZb*) *and b 2* Cloud(*aZb*).

d. (SCCs) *If* Sea(*aZb*) *is not just fag, then* Sea(*aZb*) *and the pre x of aZb ending in its last split-node form an SCC (denoted as the* sea-related SCC*). Similarly, if* Cloud(*aZb*) *is not just fbg, then* Cloud(*aZb*) *and the su x of aZb starting in its rst join-node form an SCC (denoted as the* cloud-related SCC*).*

e. (Reachability) *Let* (*x; y*) *2 G G be an ordered incidence pair (arc-node or node-arc) in G, where x and y are in di erent components of the hydrostructure of aZb. Then* (*x; y*) *can (only) be associated with the following pairs of components.*

- (Vapor*;* Cloud) *where y* = *b, or* (Sea*;* Vapor) *where x* = *a,*
- (Cloud*;* Vapor) *or* (Vapor*;* Sea)*, where both always occur for non-trivial aZb,*
- (Cloud*;* River) *or* (River*;* Sea)*, where both always occur for nonempty* River*,*
- (Cloud*;* Sea)*, where aZb is univocal or R-univocal.*

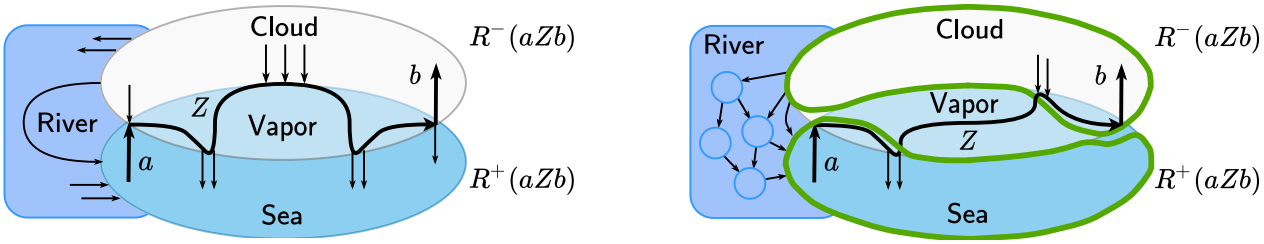

Figure 3.2: (a): The hydrostructure partitioning of a non-trivial heart *aZb* in the sets Sea, Cloud, Vapor, and River using $R^+(aZb)$ and $R\ (aZb)$. It also shows the membership of the arcs, and the reachability among the parts. (b): The hydrostructure for a trivial bridge-like walk *aZb*, with the sea-related and cloud-related SCC shown using *thick green* cycles. The River is a DAG of SCCs where the SCCs are blue cycles.

Proof. We prove the statements in reverse order for simplicity.

(e) We first prove all the incident pairs involving the River(*aZb*), and then the residual pairs involving the Vapor(*aZb*), and finally between Sea(*aZb*) and Cloud(*aZb*).

{ We show that *x; y 2* (River*;* Vapor) and *x; y 2* (River*;* Cloud) do not exist. Since *x 2* River(*aZb*), it holds that *x 2 R*$^+$(*aZb*) and therefore *x* ~~*6*~~= *a*. Thus, *y 2 R* (*aZb*) (and hence Vapor(*aZb*) or Cloud(*aZb*)), implies *x 2 R* (*aZb*) because of (*x; y*), as prepending *x* (~~*6*~~= *a*) does not make *aZb* a subwalk. This contradicts that *x 2* River(*aZb*) by definition. By symmetry *x; y 2* (Vapor*;* River) and *x; y 2* (Sea*;* River) do not exist as well.

{ For *x; y 2* (Vapor*;* Cloud) we necessarily have *y* = *b*, because *x 2 R*$^+$(*aZb*) would otherwise add *y 2 R*$^+$(*aZb*), removing it from Cloud(*aZb*). When *y* = *b* we get *aZb* as a subwalk, preventing $R^+(aZb)$ from covering *y*. By symmetry, *x; y 2* (Sea*;* Vapor) implies *x* = *a*.

- We show that $x; y \in (\mathsf{Sea}; \mathsf{Cloud})$ implies $x = a$ and $y = b$, which contradicts our definition of aZb as being from arc to arc. This is because $x \in R^+(aZb)$ would add $y \in R^+(aZb)$ if $y \neq b$, as described above, and symmetrically $y \in R\ (aZb)$ implies $x = a$.

  Let $x; y \in (\mathsf{Cloud}; \mathsf{Sea})$. We prove that if x is an arc then aZb is R-univocal (by symmetry if y is an arc then aZb is univocal), and hence trivial. Then $x \in R\ (aZb)$ but $y \notin R\ (aZb)$, so $x = b$ because each other arc is covered by $R\ (aZb)$ using its head. Assume for a contradiction that aZb contains a join node. Then by strong connectivity an arc e entering aZb at this join node needs to be (forwards) reachable from $y = \text{HEAD}(b)$, let W be such a walk of minimum length. But then, since $y \notin R\ (aZb)$, W (being a b-e walk) needs to have aZb as subwalk, which contradicts that it is minimal.

Where it applies, the necessity of the connections follows from *strong connectivity* of the graph since no other connections exist, as proved above.

(d) Let W^P be the prefix of aZb ending in its last split node. We show that if $\mathsf{Sea}(aZb) \neq \{a\}$, then for every node and arc $x \in \mathsf{Sea}(aZb)$ with $x \neq a$ there is a path P in $\mathsf{Sea}(aZb)$ containing x which starts from an arc e with $\text{TAIL}(e) \in \mathsf{Vapor}(aZb)$ and ends in a. Notice that $\text{TAIL}(e)$ is a split-node, since P *diverges* from aZb at e, ensuring that $\text{TAIL}(e)$ is in W^P. This will prove the first part of the statement because it allows to form a cycle through a and $x \neq a$ within $W^P \cup \mathsf{Sea}(aZb)$, because the path would leave aZb latest at its last split node to enter $\mathsf{Sea}(aZb)$ (using Lemma 3.4 (e)). And this will also prove the second part by symmetry.

For a node or an arc $x \neq a$ in $\mathsf{Sea}(aZb)$, by definition there is an a-x walk inside $R^+(aZb)$ that does not have aZb as subwalk. Starting from $\text{HEAD}(a) \in \mathsf{Vapor}(aZb)$ this walk exits $\mathsf{Vapor}(aZb)$ into the $\mathsf{Sea}(aZb)$ (since $R^+(aZb) = \mathsf{Vapor}(aZb) \cup \mathsf{Sea}(aZb)$), with an arc e having $\text{TAIL}(e) \in \mathsf{Vapor}(aZb)$. By strong connectivity, there is also a path from x to a. Notice that both the paths from e to x and from x to a are completely within the $\mathsf{Sea}(aZb)$ because the only arc leaving $\mathsf{Sea}(aZb)$ is a, by Lemma 3.4 (e).

This also proves that there exists such a circular path for a if there exists some $x \in \mathsf{Sea}(aZb)$ $\neq a$. For $\mathsf{Sea}(aZb) = a$, we have no split nodes on aZb because $R^+(aZb)$ does not leave aZb. This makes $W^P = ;$ resulting in no SCC.

(c) By definition, $a \in R^+(aZb)$ and $a \notin R\ (aZb)$ when aZb is bridge-like, resulting in $a \in \mathsf{Sea}(aZb)$. Symmetrically, we prove $b \in \mathsf{Cloud}(aZb)$.

(b) By strong connectivity, there always exists an x-y path. If such a path contains aZb, then consider its prefix, say P, that ends at the first occurrence of a. Notice that if

$\mathsf{Sea}(aZb) = \{a\}$, then $y = a$, making P a path from x to y without having aZb as subwalk. Otherwise, by Lemma 3.4 (d), there is a path from a to every such $y \in \mathsf{Sea}(aZb)$ which does not contain b, which when appended to P gives us the desired walk with no subwalk aZb.

(a) The only way to reach $\mathsf{Sea}(aZb)$ from $\mathsf{Cloud}(aZb)$ is through $\mathsf{Vapor}(aZb)$ (by Lemma 3.4 (e)) so every path from $\mathsf{Sea}(aZb)$ to $\mathsf{Cloud}(aZb)$ passes through $\mathsf{Vapor}(aZb)$. Further, $\mathsf{Vapor}(aZb)$ can only be entered from $\mathsf{Sea}(aZb)$ through a (by Lemma 3.4 (e)) and can only be exited to $\mathsf{Cloud}(aZb)$ through b (by Lemma 3.4 (e)), and every such path contains the entire aZb as subwalk (by Lemma 3.3). □

3.2.1 Implementation

To obtain the hydrostructure of a walk W, we need to compute $R^+(W)$ (and symmetrically $R^-(W)$), with Algorithm 6. Note that $R^+(W)$ for avertible walks is simply G, and for bridge-like walks is the part of the graph reachable from $\textsc{start}(W)$ in $G \setminus \textsc{end}(W)$. We can identify if W is avertible while performing the traversal, by checking if the traversal reaches W again after leaving it. Thereafter, we can easily compute the Sea, Cloud, Vapor and River by processing each arc and node individually.

Algorithm 6: Forward Reachability

Input: Graph G, non-empty walk W
Output: $R^+(W)$

```
1 if W is an open path then
2 |   R⁰ ← {x ∈ V ∪ E | ∃ START(W)-x walk in G \ END(W)}
3 |   if ∃ arc e ∈ R⁰ : e ∉ W ∧ HEAD(e) ∈ W then
4 |   |   return R⁰
5 return G
```

In conclusion, we have the following:

Theorem 3.5. *The hydrostructure of any walk can be computed in $O(m)$ time.*

Proof. Each node and arc can be annotated with its membership in $R^+(W)$ and $R^-(W)$ using Algorithm 6 and its symmetric variant in $O(m)$ time. It remains to prove the correctness of Algorithm 6.

- Suppose Algorithm 6 reaches Line 5 and returns G. If W is not an open path, then by Lemma 3.3 this is correct. Otherwise, if W is an open path, there is an arc $e \in R^0$ such that $\textsc{tail}(e) \notin W \wedge \textsc{head}(e) \in W \wedge e \neq \textsc{start}(W)$. Then, there is a walk from $\textsc{start}(W)$ via e to $\textsc{end}(W)$ that does not contain W as subwalk, so W is avertible, and by Lemma 3.3 it follows that outputting G is correct, because $G = \mathsf{Vapor}(W) \subseteq R^+(W) \subseteq G$.

- Suppose Algorithm 6 reaches Line 4 and returns R^0. Then for that to be correct, by Lemma 3.3 W must be bridge-like and R^0 must be equal to $R^+(W)$.

  Assume for a contradiction that W is avertible. Then by Lemma 3.3 it holds that $\mathsf{Vapor}(W) = G$, and therefore $R^+(W) = G$. Hence, by definition of $R^+(\,)$, there is a $\textsc{start}(W)$-$\textsc{end}(W)$ walk W that does not have W as subwalk. Such a walk leaves W before reaching $\textsc{end}(W)$ to avoid having W as subwalk. But then it must enter W again at an arc e other than $\textsc{start}(W)$ before it reaches $\textsc{end}(W)$ for the first time. Therefore, $e \in R^0$ and it holds that $e \notin W$ and $\textsc{head}(e) \in W$. But then the condition in Line 3 would be false and Algorithm 6 would not reach Line 4, a contradiction.

  Therefore, W is bridge-like, and it remains to prove that $R^0 = R^+(W)$. ( ) Let $x \in R^0$. Then there is a $\textsc{start}(W)$-x walk in $G \setminus \textsc{end}(W)$, so $x \in R^+(W)$. ( ) Let $x \in R^+(W)$. Then there is a $\textsc{start}(W)$-x walk in G that does not contain W as subwalk. This walk cannot contain $\textsc{end}(W)$, since otherwise one of its prefixes until $\textsc{end}(W)$ would prove $\textsc{end}(W) \in R^+(W)$, contradicting Lemma 3.4 (c). So $x \in R^0$. □

3.3 Safety in Circular Models

In the circular models the *heart* of a walk determines its safety, as the univocal extension of a safe walk is naturally safe. Therefore, every trivial walk is naturally safe because of the covering constraint for any arc in its heart, which can be univocally extended to get the whole walk.

A non-trivial walk W with a bridge-like heart is safe as well, as every circular arc-covering walk contains all bridge-like walks by definition. If on the other hand $\mathsf{Heart}(W)$ is avertible (having $\mathsf{Vapor}(W) = G$), then a walk can cover the entire graph without having $\mathsf{Heart}(W)$ as subwalk, making it unsafe by definition. Thus, the characterization for circular safe walks (Definition 1.2) is described in the following lemmata:

Theorem 3.6 (1-Circular). *A non-trivial walk W is 1-circular safe if $\mathsf{Vapor}(\mathsf{Heart}(W))$ is a path.*

Proof. ( ) Let $\mathsf{Vapor}(\mathsf{Heart}(W))$ not be a path. Let $aZb := \mathsf{Heart}(W)$, a^0 be a sibling join arc of a and b^0 be a sibling split arc of b. Then, by Lemma 3.3, aZb is avertible. We prove that aZb is not 1-circular safe by constructing a circular arc-covering walk C that does not contain aZb. Let P be a a^0-b^0 walk such that aZb is not a subwalk of P, which exists since aZb is avertible. Let C^0 be an arbitrary circular arc-covering walk of G. If C^0 does not contain aZb the claim holds, otherwise consider C obtained from C^0, by replacing every aZb subwalk with $aZPZb$. Concluding, since aZb is not 1-circular safe, it holds that W is not 1-circular safe.

( ) Let $\mathsf{Vapor}(\mathsf{Heart}(W))$ be a path. Then $\mathsf{Vapor}(\mathsf{Heart}(W))$ is not the whole G. Moreover, by Lemma 3.3, $\mathsf{Heart}(W)$ is bridge-like, so there are $x, y \in G$ such that each x-y walk has $\mathsf{Heart}(W)$ as a subwalk. Therefore, since every circular arc-covering walk C in G contains both

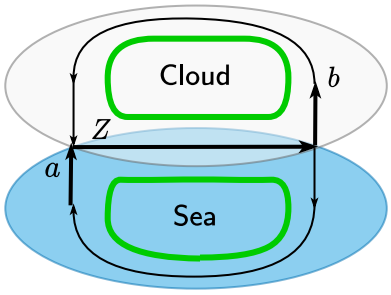

Figure 3.3: In the absence of the River, the two cycles (shown in thick green) can cover the cloud-related and sea-related SCCs of a non-trivial Heart without having aZb as subwalk.

x and y and hence a subwalk from x to y, Heart(W) is a subwalk of C (and hence 1-circular safe). Further, since C is circular, the whole W is 1-circular safe, as Heart(W) can only be traversed by also passing through its univocal extension. □

Now, for k 2, a single circular arc-covering walk can be repeated to get k circular arc-covering walks, implying that every k-circular safe walk is 1-circular safe. For the k-circular safety of a non-trivial walk W, an added issue (see Figure 3.3) is that two different circular walks can cover the sea and cloud-related strongly connected components. However, this is not possible if these components do not entirely cover the graph, i.e. the River is not empty. Thus, we get the following characterization.

Theorem 3.7 (k-Circular). *A non-trivial walk W is k-circular safe for k 2, i* Vapor(Heart(W)) *is a path and* River(Heart(W)) *is non-empty.*

Proof. ( ) We first prove that W is not k-circular safe if Vapor(Heart(W)) is not a path or River(Heart(W)) is empty.

- If Vapor(Heart(W)) is not a path, then using Theorem 3.6 W is not 1-circular safe, that means there is a circular arc-covering walk C not containing W. Thus, a collection of k copies of C is also arc-covering not containing W, proving that W is not k-circular safe.

- If Vapor(Heart(W)) is a path, and River(Heart(W)) is empty, we can prove that W is not k-circular safe for k 2 because for a non-trivial W it holds that Cloud(Heart(W)) [Vapor(Heart(W)) and Sea(Heart(W)) [Vapor(Heart(W)) are each strongly connected. Thus, two different walks can separately cover them as River(Heart(W)) is empty.

  We only prove that Cloud(Heart(W)) [Vapor(Heart(W)) is strongly connected as the other follows by symmetry. Since W is non-trivial, Heart(W) starts with a join arc a and ends with a split arc b. If Cloud(Heart(W)) = fbg then clearly it is the join sibling of a, making Cloud(Heart(W)) [Vapor(Heart(W)) a cycle and hence strongly connected. Otherwise, if Cloud(Heart(W)) 6= fbg then the strong connectivity follows by Lemma 3.4 (d).

(( ) If Vapor(Heart(W)) is a path then Heart(W) is bridge-like by Lemma 3.3. If River(Heart(W)) is not empty, since the graph is strongly connected we can enter River(Heart(W)) only from Cloud(Heart(W)) (by Lemma 3.4 (e)) using some arc. Hence, every walk covering this arc requires a subwalk from River(Heart(W)) to Cloud(Heart(W)), which passes through Sea(Heart(W)) (by Lemma 3.4 (e)). Using Lemma 3.4 (a) we get that this subwalk necessarily contains the walk W making it k-circular safe. □

Notice that our characterization does not distinguish between k when k 2, implying that all the problems of (k 2)-circular safety are equivalent. These characterizations can be directly adapted for an optimal verification algorithm, by computing the hydrostructure for the given walk in linear time. This also results in $O(m^2n)$ time enumeration algorithms using the *two pointer* algorithm on a simple circular arc-covering walk of length $O(mn)$ (see Section 3.3.1). Moreover, using the optimal $O(mn)$ time 1-circular safe algorithm [14], the maximal k-circular safe walks can also be *optimally* enumerated in $O(mn)$ time, by computing the hydrostructure for $O(n)$ non-trivial 1-circular safe walks (Theorem 1.8 (b)) and using an interesting property of 1-circular safe walks that are not k-circular safe. See Section 3.3.1 for more details on the implementation.

The following property proved to be necessary for our efficient algorithm to compute k-circular safe walks.

Lemma 3.8. *Let k 2. For a 1-circular safe walk $W = X$Heart(W)Y where* Heart(W) $= aZb$, *if W is not k-circular safe then XaZ and ZbY are k-circular safe walks.*

Proof. By symmetry of the statement, we only prove that if W is a 1-circular safe walk of G that is not k-circular safe then XaZ is a k-circular safe walk of G. This also implies that Q is non-trivial otherwise it is implicitly k-circular safe. Further, being a subwalk of the 1-circular safe W makes $Q := XaZ$ also 1-circular safe. Let Heart(Q) $= aZ\ b$, which is either aZ or a prefix of aZ. Since aZb is a non-trivial heart, there exists a sibling split arc b' of b. We claim that for any such sibling b', we have b' 2 River($aZ\ b$), which is equivalent to $b' \not\in R^+(aZ\ b\ )\ [\ R\ (aZ\ b\ )$.

Now, aZ is bridge-like (since aZb is bridge-like), so each START(aZ)-END(aZ) walk has aZ as subwalk. Since END(aZ) = TAIL(b') it follows that each a-b' walk has aZ as subwalk, and hence its prefix $aZ\ b$. Thus, $b' \not\in R^+(aZ\ b\ )$. Further, we know that b' 2 Sea(aZb) as it is surely in $R^+(aZb)$ (by definition) and not in Vapor(aZb) (for bridge-like aZb by Lemma 3.3). So any path from b' to b 2 Vapor(aZb) surely enters Vapor(aZb) through a (by Lemma 3.4 (e)). Hence it has $aZ\ b$ as a subwalk since Vapor(aZb) is a path and $aZ\ b$ is a prefix of aZb. Thus, $b' \not\in R\ (aZ\ b\ )$.

Thus $b' \not\in (R^+(aZ\ b\ )\ [\ R^+(aZ\ b\ ))$ implying that b' 2 River($aZ\ b$), and by Theorem 3.7 that $aZ\ b$ is k-circular safe. Therefore, since Heart(XaZ) $= aZ\ b$, we conclude that XaZ is k-circular safe. By symmetry ZbY is also k-circular safe. □

3.3.1 Implementation

The characterization can be directly adapted to get a verification algorithm for evaluating whether a given walk is safe in 1-circular and k-circular models. A trivial walk W is easily identified in $O(|W|)$ time by checking for univocal walks. For a non-trivial walk W the hydrostructure can be computed in linear time (Theorem 3.5) and the characterization can be verified in $O(|W|)$ time resulting in an $O(m)$ time verification algorithm for both 1-circular safe and k-circular safe walks.

Since every trivial walk is 1-circular safe and k-circular safe, they can be enumerated by computing univocal extensions of each arc in $O(mn)$ time. For reporting all non-trivial maximal 1-circular safe and k-circular safe walks, we use the two pointer algorithm on a candidate solution. A candidate solution can be computed by arranging the m arcs in some circular order, and adding the shortest path between adjacent arcs to complete the closed arc-covering walk of size $O(mn)$. To get an arc-covering solution made up of k closed walks, we can have k *identical* copies of this $O(mn)$-size walk. Now, using the two-pointer algorithm (Theorem 1.8 (a)) on one of these identical walks, with the above linear time verification algorithm we can enumerate all non-trivial maximal 1-circular safe and k-circular safe walks in $O(m^2n)$ time.

Alternatively, non-trivial k-circular safe walks can computed using non-trivial 1-circular safe walks in $O(mn)$ time. This is because the number of non-trivial 1-circular safe walks is $O(n)$ (Theorem 1.8 (b)). Computing their hydrostructure in $O(m)$ time either verifies them to be k-circular safe or its two subwalks as k-circular safe (using Lemma 3.8). Hence, using the $O(mn)$ time optimal 1-circular safe algorithm [14] we directly obtain the following result.

Theorem 3.9. *Given a strongly connected graph with m arcs and n nodes, all the maximal k-circular safe walks can be reported in $O(mn)$ time.*

3.4 Safety in Subset Covering Models

The subset covering models reduce the covering constraint to only a subset $F \subseteq E$ instead of the entire E. Note that this implies that any walk which is a solution in the E-covering model is also a solution in the F-covering model, but we can have walks which are not E-covering but F-covering. Thus, every F-covering safe walk is E-covering safe. Furthermore, if F is empty then no walk is safe, so in this section we assume $F \neq \emptyset$.

Note that, in contrast to the E-covering models, in the F-covering models, a single arc is not trivially safe. So in order to characterise their safety we extend the definition of hydrostructure for single arcs, by splitting the arc using a dummy node. Thus, a single arc can be treated as a walk with two arcs, for which the hydrostructure is already defined.

We call an arc in F an F-arc and a part of the graph *F-covered* if an F-arc belongs to it. Also, for a bridge-like walk, we define the *F-covered SCCs*, which are the maximal SCCs

of its River, and the sea- and cloud-related SCCs (see Lemma 3.4 (d)) that are F-covered. Similarly, for an avertible walk, the *F-covered SCC* is G. Furthermore, the F-covered sea and cloud-related SCCs are counted as one if they share all their F-arcs.

3.4.1 Circular Models

A trivial walk W is circular safe if Heart(W) is F-covered.

For bridge-like walks, the issue with F-covering circular walks is that a set of cycles can cover the entire F if it is covered by some F-covered SCCs, making the walks unsafe. This is not possible if an arc in F exists outside the F-covered SCCs, or the number of such SCCs are more than k. To prove the following we first state a result from [13].

Lemma 3.10. *For a trivial avertible walk aZb in a strongly connected graph G, the graph G r* Heart(aZb) *is also strongly connected.*

Theorem 3.11 (Circular F-Covering Safety). *For F E a* 1*-circular safe walk W that contains at least two arcs is k-circular F-covering safe i for the hydrostructure on W (for trivial) or* Heart(W) *(for non-trivial) it holds that*

(a) k is less than the number of F-covered SCCs, or

(b) there exists an F-covered arc in the River*, that is not in any SCC of the* River*, or*

(c) W has an F-covered trivial heart.

Proof. ( )) Assume that k is greater than or equal to the number of F-covered SCCs, there exists no F-covered arc in the River, that is not in any SCC of the River, and W does not have an F-covered trivial heart. We distinguish between W being trivial avertible, trivial bridge-like or non-trivial.

- If W is trivial and avertible but does not have an F-covered trivial heart, then by Lemma 3.10 all arcs in F can be covered by a single circular walk without traversing Heart(W). Thus, W is not k-circular F-covering safe.

- If W is trivial and bridge-like or non-trivial (in which case Heart(W) is bridge-like by Theorem 3.6), then all arcs in F are in F-covered SCCs. Indeed arcs in the River are in F-covered SCCs by hypothesis; arcs in the Cloud or Sea are relatively in the F-covered cloud-related SCC or sea-related SCC; arcs in the Vapor of non-trivial walks are both in the sea-related SCC and cloud-related SCC; and for trivial walks the arcs in the Vapor that are neither in the cloud-related SCC nor in the sea-related SCC are in the trivial heart by definition, which does not contain arcs in F by hypothesis. Therefore, since all arcs in F are in F-covered SCCs and we have at most k F-covered SCCs of which none contains the whole walk W, it follows that F can be covered with k walks not having W as subwalk, hence W is not k-circular F-covering safe.

(( )

- If W has an F-covered trivial heart then it is k-circular F-covering safe.
- If k is less than the number of F-covered SCCs, then at least one solution walk needs to intersect two SCCs. We need to distinguish three cases: (A) if these are sea-related and cloud-related (in which case the sea-related SCC and the cloud-related SCC do not share all their F-arcs), (B) if one is in the River and the other is sea-related or cloud-related, or (C) if both are in the River. In all cases, Lemma 3.4 (e) implies that this solution walk needs to traverse from Sea to Cloud, which by Lemma 3.4 (a) implies the k-circular F-covering safety of W. In case (C) this is because the River is not strongly connected if there is more than one maximal SCC.
- If there exists an F-covered arc in the River, that is not in any SCC of the River, then the River is not strongly connected. Therefore, to cover this arc with a circular walk, the River needs to be exited and entered, which by Lemma 3.4 (e) implies that a solution walk covering this arc needs to traverse from Sea to Cloud, which by Lemma 3.4 (a) implies the k-circular F-covering safety of W. □

3.5 Safety in Subset Visibility Models

In subset visibility we limit the solution of the problem to its *visible* arcs. We thus define $vis_F(W)$ as the substring of a walk W that belongs to F. Note that this does not change the solution of the problem but only its representation, which is now limited to the visible set F. So if a walk W is a solution in the E-visible model for a problem, then we have $vis_F(W)$ is a solution in the F-visible model for the problem, where F E. Abusing notation we also simply say that W is a solution for the F-visible model of the problem.

However, note that the safe solutions ignore the non-visible part of the walk to compute safety. For example Theorem 3.6 relaxes to the visible part of Vapor(Heart(W)) to being a single sequence of arcs (possibly disconnected). Henceforth, we shall continue to refer to such a sequence as a visible path despite being disconnected. Thus, relaxing the criteria for safety implies that even though the F-visible solution for a problem is the same as the E-visible solution, we have that every E-visible safe walk is F-visible safe but not vice-versa. As a result, in order to characterise the safety in the F-visible model, we relax the definition of the hydrostructure by relaxing the forward and backward reachability of a walk as follows:

Definition 3.12. *For a walk W where* $\text{END}(W); \text{START}(W)$ *2 F, we define the* restricted *forward and backward F-visible reachability as*

- $R_F^+(W) = \{x \in G \mid \exists \text{ } \text{START}(W)\text{-}x \text{ walk } W' : vis_F(W) \text{ not subwalk of } vis_F(W')\}$,
- $R_F(W) = \{x \in G \mid \exists \text{ } x\text{-}\text{END}(W) \text{ walk } W' : vis_F(W) \text{ not subwalk of } vis_F(W')\}$.

Notice that the modified hydrostructure is merely more relaxed in terms of the Vapor. For the F-visible model, Lemma 3.3 relaxes the criteria of the Vapor being more than a single path, to having a single visible path (see Figure 3.4). Similarly, the properties of the hydrostructure are adapted by considering $vis_F(W)$ instead of W.

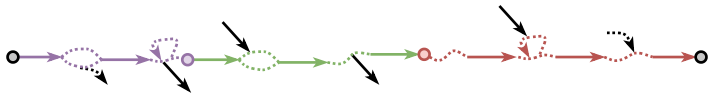

Figure 3.4: A non-trivial visible walk with visible heart in green, and visible wings in violet and red, where the dotted arcs are invisible.

The *visible* wings of a walk W_F^l and W_F^r and the visible heart $\mathsf{Heart}_F(W)$ require *visible splits* and *visible joins* in terms of only visible arcs possibly having invisible arcs (or even subgraphs) between them. Thus, for arcs $e; f \ 2 \ F$ we consider $(e; f)$ as adjacent if there exists a path from $\textsc{head}(e)$ to $\textsc{tail}(f)$ in $G \setminus F$.

3.5.1 Implementation.

In order to present the verification and enumeration algorithms for any model, it is sufficient to describe the computation of the hydrostructure. The only difference from Section 3.2 is that we shall now be using *visible* adjacency and hence *visible* splits and joins. In order to compute these efficiently we pre-compute the all-pairs reachability among endpoints of arcs in F in $G \setminus F$. This can be computed performing a traversal (BFS or DFS) from the k endpoints of arcs in F in $O(mk)$ time, where $k \quad \min(n; jFj)$. Thus, all the previous algorithms are adaptable in the F-visible model with an added expense of $O(mn)$.

Bibliography

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015.

[2] Nidia Obscura Acosta, Veli Mäkinen, and Alexandru I. Tomescu. A safe and complete algorithm for metagenomic assembly. *Algorithms for Molecular Biology*, 13(1):3:1–3:12, 2018.

[3] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[4] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58, 2015.

[5] Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466. IEEE Computer Society, 2016.

[6] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[7] Cristina Bazgan, Till Fluschnik, André Nichterlein, Rolf Niedermeier, and Maximilian Stahlberg. A more fine-grained complexity analysis of finding the most vital edges for undirected shortest paths. *Networks*, 73(1):23–37, 2019.

[8] Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 148–193. ACM, 2014.

[9] Djamal Belazzougui and Simon J. Puglisi. Range predecessor and lempel-ziv parsing. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 2053–2071. SIAM, 2016.

[10] Sébastien Boisvert, François Laviolette, and Jacques Corbeil. Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. *Journal of computational biology*, 17(11):1519–1533, 2010.

[11] G. Bresler, M. Bresler, and D. Tse. Optimal Assembly for High Throughput Shotgun Sequencing. *BMC Bioinformatics*, 14(Suppl 5):S18, 2013.

[12] Terence A Brown. The human genome. In *Genomes. 2nd edition*. Wiley-Liss, 2002.

[13] Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian Schmidt, Alexandru I Tomescu, and Elia C Zirondelli. Genome assembly, a universal theoretical framework: unifying and generalizing the safe and complete algorithms. *Submitted*, November 2020.

[14] Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I. Tomescu. An Optimal $O(nm)$ Algorithm for Enumerating All Walks Common to All Closed Edge-covering Walks of a Graph. *ACM Trans. Algorithms*, 15(4):48:1–48:17, 2019.

[15] Katarína Cechlárová. Persistency in the assignment and transportation problems. *Mat. Meth. OR*, 47(2):243–254, 1998.

[16] Kun-Mao Chao, Ross C. Hardison, and Webb Miller. Locating well-conserved regions within a pairwise alignment. *CABIOS*, 9(4):387–396, 1993.

[17] Moses Charikar and Edith Cohen, editors. *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*. ACM, 2019.

[18] Robert L Charlebois. *Organization of the prokaryotic genome*. ASM Press Washington DC, 1999.

[19] Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 06 2013.

[20] Marie Costa. Persistency in maximum cardinality bipartite matchings. *Oper. Res. Lett.*, 15(3):143–9, 1994.

[21] Marie-Christine Costa, Dominique de Werra, and Christophe Picouleau. Minimum d-blockers and d-transversals in graphs. *J. Comb. Optim.*, 22(4):857–872, 2011.

[22] Bartlomiej Dudek and Pawel Gawrychowski. Computing quartet distance is equivalent to counting 4-cycles. In Charikar and Cohen [17], pages 733–743.

[23] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[24] David Eppstein. K-best enumeration. *Bulletin of the EATCS*, 115, 2015.

[25] Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of string matching for graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[26] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 390–398, 2000.

[27] A Friemann and S Schmitz. A new approach for displaying identities and differences among aligned amino acid sequences. *Comput Appl Biosci*, 8(3):261–265, Jun 1992.

[28] Fabian Gärtner, Lydia Müller, and Peter F. Stadler. Superbubbles revisited. *Algorithms Mol. Biol.*, 13(1):16:1–16:17, 2018.

[29] Hans R Gelderblom. Structure and classification of viruses. In *Medical Microbiology. 4th edition*. University of Texas Medical Branch at Galveston, 1996.

[30] Loukas Georgiadis, Giuseppe F Italiano, and Nikos Parotsidis. Strong connectivity in directed graphs under failures, with applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1880–1899. SIAM, 2017.

[31] Meigu Guan. Graphic programming using odd and even points. *Chinese Math.*, 1:237–277, 1962.

[32] A. Guénoche. Can we recover a sequence, just knowing all its subsequences of given length? *Computer Applications in the Biosciences*, 8(6):569–574, 1992.

[33] P. L. Hammer, P. Hansen, and B. Simeone. Vertices belonging to all or to no maximum stable sets of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4):511–522, 1982.

[34] Costas S. Iliopoulos, Ritu Kundu, Manal Mohamed, and Fatima Vayani. Popping superbubbles and discovering clumps: Recent developments in biological sequence analysis. In

Mohammad Kaykobad and Rossella Petreschi, editors, *WALCOM: Algorithms and Computation - 10th International Workshop, WALCOM 2016, Kathmandu, Nepal, March 29-31, 2016, Proceedings*, volume 9627 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2016.

[35] Giuseppe F. Italiano. Amortized efficiency of a path retrieval data structure. *Theor. Comput. Sci.*, 48(3):273–281, 1986.

[36] Iu, V. L. Florent'ev, A. A. Khorlin, K. R. Khrapko, and V. V. Shik. Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method. *Doklady Akademii nauk SSSR*, 303(6):1508–1511, 1988.

[37] Benjamin Grant Jackson. *Parallel methods for short read assembly*. PhD thesis, Iowa State University, 2009.

[38] Evgeny Kapun and Fedor Tsarev. De Bruijn superwalk with multiplicities problem is NP-hard. *BMC Bioinformatics*, 14(Suppl 5):S7, 2013.

[39] John D. Kececioglu and Eugene W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1/2):7–51, 1995.

[40] John Dimitri Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction*. PhD thesis, University of Arizona, Tucson, AZ, USA, 1992.

[41] Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In Charikar and Cohen [17], pages 756–767.

[42] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 827–840. ACM, 2018.

[43] Valerie King, Satish Rao, and Rorbert Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17(3):447–474, 1994.

[44] Carl Kingsford, Michael C Schatz, and Mihai Pop. Assembly complexity of prokaryotic genomes using short reads. *BMC bioinformatics*, 11(1):21, 2010.

[45] Ka-Kit Lam, Asif Khalak, and David Tse. Near-optimal assembly for shotgun sequencing with noisy reads. *BMC Bioinform.*, 15(S-9):S4, 2014.

[46] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature Methods*, 9(4):357, 2012.

[47] Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, 31(10):1674–1676, 2015.

[48] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[49] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[50] Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, 2015.

[51] Julian R Marchesi. Metagenomics: Current innovations and future trends. *Future Microbiology*, 7(7):813–814, 2012.

[52] Paul Medvedev. Modeling biological problems in computer science: a case study in genome assembly. *Brie ngs in bioinformatics*, 20(4):1376–1383, 2019.

[53] Paul Medvedev and Michael Brudno. Maximum likelihood genome assembly. *Journal of computational biology*, 16(8):1101–1116, 2009.

[54] Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. In *WABI*, pages 289–301, 2007.

[55] Eugene W. Myers. The fragment assembly string graph. In *ECCB/JBI*, page 85, 2005.

[56] Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *Journal of computational biology*, 16(7):897–908, 2009.

[57] Niranjan Nagarajan and Mihai Pop. Sequence assembly demystified. *Nature Reviews Genetics*, 14(3):157–167, 2013.

[58] Giuseppe Narzisi, Bud Mishra, and Michael C Schatz. On algorithmic complexity of biomolecular sequence assembly problem. In *Algorithms for Computational Biology*, pages 183–195. Springer, 2014.

[59] James B Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Cpmputing*, pages 765–774, 2013.

[60] Benedict Paten, Jordan M. Eizenga, Yohei M. Rosen, Adam M. Novak, Erik Garrison, and Glenn Hickey. Superbubbles, ultrabubbles, and cacti. *J. Comput. Biol.*, 25(7):649–663, 2018.

[61] Hannu Peltola, Hans Söderlund, Jorma Tarhio, and Esko Ukkonen. Algorithms for some string matching problems arising in molecular genetics. In *IFIP Congress*, pages 59–64, 1983.

[62] P. A. Pevzner. l-Tuple DNA sequencing: computer analysis. *Journal of Biomolecular Structure & Dynamics*, 7(1):63–73, August 1989.

[63] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[64] Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *Nature Methods*, 17(2):155–158, 2020.

[65] Ilan Shomorony, Samuel H. Kim, Thomas A. Courtade, and David N. C. Tse. Information-optimal genome assembly via sparse read-overlap graphs. *Bioinform.*, 32(17):494–502, 2016.

[66] Alexandru I. Tomescu and Paul Medvedev. Safe and Complete Contig Assembly Via Omnitigs. In Mona Singh, editor, *Research in Computational Molecular Biology - 20th Annual Conference, RECOMB 2016, Santa Monica, CA, USA, April 17-21, 2016, Proceedings*, volume 9649 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2016.

[67] Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017.

[68] Martin Vingron and Patrick Argos. Determination of reliable regions in protein sequence alignments. *Prot. Engin.*, 3(7):565–569, 1990.

[69] Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680. SIAM, 2015.

[70] Fan Wu, Su Zhao, Bin Yu, Yan-Mei Chen, Wen Wang, Zhi-Gang Song, Yi Hu, Zhao-Wu Tao, Jun-Hua Tian, Yuan-Yuan Pei, Ming-Li Yuan, Yu-Ling Zhang, Fa-Hui Dai, Yi Liu, Qi-Min Wang, Jiao-Jiao Zheng, Lin Xu, Edward C. Holmes, and Yong-Zhen Zhang. A new

oronavirus associated with human respiratory disease in china. *Nature*, 579(7798):265–269, 2020.

[71] Rico Zenklusen, Bernard Ries, Christophe Picouleau, Dominique de Werra, Marie-Christine Costa, and Cédric Bentz. Blockers and transversals. *Discrete Mathematics*, 309(13):4306–4314, 2009.

[72] M Zuker. Suboptimal sequence alignment in molecular biology. alignment with error analysis. *J Mol Biol*, 221(2):403–420, Sep 1991.

Acknowledgments

First of all, I want to express my gratitude indefinitely to everyone I worked with, I have always been at ease thanks to all of you and the ideas and suggestions you gave to me are invaluable.

However, I want to give more space in thanking people and collaborators for reasons that stray beyond the work and research we did together, since these are the true gifts you gave me.

My utmost gratitude goes to Prof. Romeo Rizzi. Since the first time I met him, I have always found his passion and fun he had while working to be enlightening: passion and fun that he always tried to convey to me. Thank you for the patience you had during these years and thank you for the deep and funny discussions we had.

I would also like to thank Prof. Alexandru Tomescu. I could have never found a warmer welcome in Helsinki than the one you and the guys, Sebastian Schmidt and Shahbaz Khan, offered me. Thank you all for the laughs and time we spent together, no matter the pandemic thanks to the billion of online platforms we used.

Thanks to the group we created in Verona: Massimo Cairo, Emanuele De Natale, Piercarlo Fracasso, Lorenzo Martini, Alice Raffaele, Alessandro Rapa, Chiara Segala, Chiara Tenga, Andrea Veronese and Roberta Zitelli. Thank you for the life we shared, in hope to keep on sharing.

Thanks to all my friends in Modena. There is very little to say here a part from, again, thank you. Dearly.

Finally, thanks to my parents and closest relatives for supporting me through each hard decision I took, with unconditional love and understanding.