Genome Assembly, from Practice to Theory: Safe, Complete and Linear-Time

Massimo Cairo
Department of Computer Science, University of Helsinki, Finland

Romeo Rizzi
Department of Computer Science, University of Verona, Italy

Alexandru I. Tomescu
Department of Computer Science, University of Helsinki, Finland

Elia C. Zirondelli
Department of Mathematics, University of Trento, Italy
Department of Computer Science, University of Verona, Italy

Abstract

Genome assembly asks to reconstruct an unknown string from many shorter substrings of it. Even though it is one of the key problems in Bioinformatics, it is generally lacking major theoretical advances. 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, by describing 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. This 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.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Theory of computation → Graph algorithms analysis; Applied computing → Computational biology

Keywords and phrases Graph algorithm, strong connectivity, reachability under failures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.43

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2002.10498` [12]

Funding This work was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO) and by the Academy of Finland (grants No. 322595, 328877).

48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).
Editors: Nikhil Bansal, Emanuela Merelli, and James Worrell; Article No. 43; pp. 43:1–43:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We thank Sebastian Schmidt for useful comments, including the observation that the bound on the total length of all maximal macrotigs can be improved to $O(n)$ (from $O(m)$ initially), Shahbaz Khan and Bastien Cazaux for helpful discussions and comments.

1 Introduction

Theoretical and practical 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 [58] from short reads using the MEGAHIT assembler [39].

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. [17, 57] for phylogeny problems, [22, 6, 34] for text indexing, [23, 7, 35] for text compression, and negative results, see e.g. [4, 1, 5, 21] 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 [47]. Another reason stems from theory: historically, finding an optimal genome assembly solution is considered NP-hard under several formulations [49, 33, 32, 43, 46, 29, 48], 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 [36].

Given all these setbacks, most state-of-the-art assemblers, including e.g. MEGAHIT [39] (for short reads), or wtdbg2 [52] (for long reads), generally employ a very simple and *linear-time* strategy, dating back to 1995 [32]. They start by building an assembly graph encoding the overlaps of the reads, such as a *de Bruijn graph* [50] or an *overlap graph* [45] (graphs are directed in this paper). 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 [32]. 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: A theoretical framing of practical genome assembly. 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 [55] 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 [27, 8, 46, 53, 37, 9], while other works already considered simple linear-time generalizations of unitigs [51, 44, 30, 36], without knowing if the "assembly limit" is reached.

To make this question precise, [55] introduced the following *safe and complete* framework. Given a notion of 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 edge was called *persistent* if it appears in all solutions [28, 16, 13], for example persistent edges for maximum bipartite matchings [16]. It also has roots in other Bioinformatics works [56, 14, 24, 59] 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 [49, 33, 32], or some type of shortest walk covering all nodes or arcs of the assembly graph [51, 43, 44, 31, 29, 48, 46]. However, it is widely acknowledged [46, 48, 42, 47, 41, 36] 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, [55] 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[1]). Put otherwise, safe walks for *all* arc-covering walks are more likely to be correct than safe walks for some peculiar 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) [51]. Further, these safe paths have been generalized in [44, 30, 36] 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 [55, 54] as being exactly those that are *omnitigs*, see Definition 1, Figure 1, and Theorem 8. 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).

▶ **Definition 1** (Omnitig). *Let $W = e_0 \dots e_\ell$ be a walk. We say that a non-empty path P is a j-i* forbidden path *for W, for some $1 \le i \le j \le \ell$, 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 $1 \le i \le j \le \ell$ there exists a j-i forbidden path for W.*

Furthermore, through experiments on "perfect" human read datasets, [55] 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)

[1] For example, closed arc-covering walks are a common relaxation of the fundamental notions of closed Eulerian walk (we now pass through each arc *at least once*), and of closed Chinese postman walk (i.e. a closed arc-covering walk *of minimum length*) [26], which were mentioned in [46] as unsatisfactory models of genome assembly.

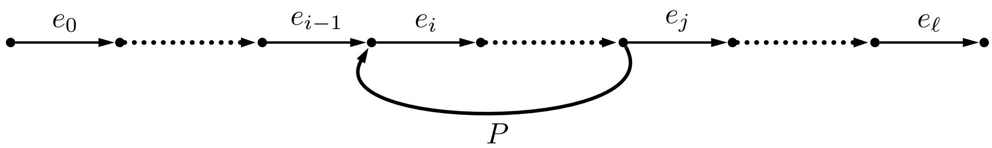


Figure 1 Walk $e_0 \dots e_\ell$ is *not* an omnitig because there is a forbidden path P.

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. [11] 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.

Our results. Our main result is an $O(m)$-size representation of all maximal omnitigs[2], 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 [11].

▶ **Theorem 2.** *Given a strongly connected graph G with n nodes and m arcs, there exists a $O(m)$-size representation of all maximal omnitigs, consisting of a set $\mathcal{M}$ of walks (*maximal macrotigs*) of total length $O(n)$ and a set $\mathcal{F}$ of arcs, such that every maximal omnitig is the univocal extension*[3] *of either a subwalk of a walk in $\mathcal{M}$, or of an arc in $\mathcal{F}$.*

Moreover, $\mathcal{M}$, $\mathcal{F}$, 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 3.** *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 using two interesting ingredients. The first is a novel graph structure (*macronodes*), obtained after a *compression* operation of "easy" nodes and arcs (Section 4). The second is a connection to a recent result by Georgiadis et al. [25] 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).

Theorem 2 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), by pre-computing the univocal extensions from any node, needed in Theorem 2. For example, given that the number of maximal omnitigs is $O(m)$ [11], this implies the following result:

[2] Note that the total length of the maximal omnitigs is at least m, since every arc is an omnitig.

[3] The *univocal extension* $U(W)$ of a walk W is obtained by appending to W the longest path whose nodes (except for the last one) have out-degree 1, and prepending to W the longest path whose nodes (except for the first one) have in-degree 1; see Section 2 for the formal definition.

▶ **Corollary 4.** *Given a strongly connected graph G with m arcs, it is possible to compute the lengths of all maximal omnitig in total time $O(m)$.*

Corollary 4 leads to a linear-time computation of various statistics about maximal omnitigs, such as minimum, maximum, and average length (useful e.g. in [15]). 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.

Significance of our results. This paper 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 [32].

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 Section 4) de Bruijn graph of human chromosome 10 (length 135 million) has 467 thousand arcs [11, 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 running time of the *quadratic* algorithm of [11] on its compressed de Bruijn graph is 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 a follow-up work [10] 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 [54, 55]. Safe walks for all these models are subsets of omnitigs [2, 10]. 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 [10].

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 [18], or enhanced with various heuristics as in the well-known BLAST aligner [3]. Also text indexes such as the FM-index [22] are extended in popular read mapping tools (e.g. [40, 38]) 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 [19, 20], because they already *synthesize all that can be correctly reconstructed from the input data.*

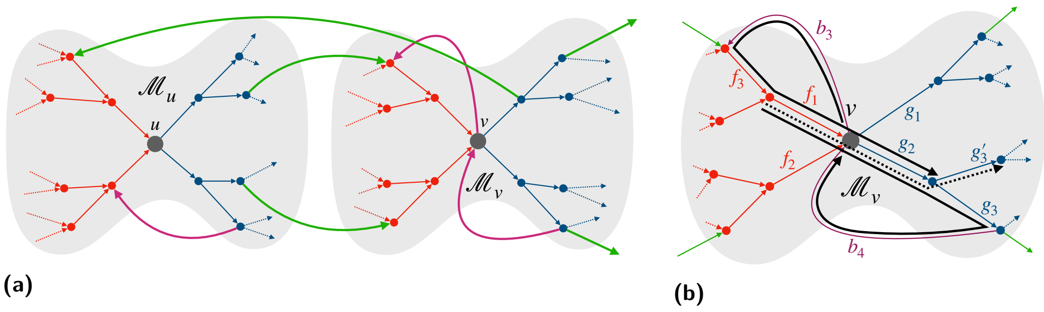


Figure 2 Figure 2a: Given a bivalent node v, the macronode $\mathcal{M}_v$ is the subgraph of G induced by the nodes reaching v with a split-free path (in red), and the nodes reachable from v with a join-free path (in blue). These two types of nodes induce the two trees of the macronode. By definition, every arc with endpoints in different 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*). Figure 2b: 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 micro-omnitig 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 $b_3f_3f_1g_2g_3b_4$.

2 Overview of the proofs

We highlight here our key structural and algorithmic contributions, and give more formal details in Sections 4 and 5. We start with the minimum terminology needed to understand this section, and defer the rest of the notation to Section 3.

Terminology. Functions $t(\cdot)$ and $h(\cdot)$ denote the *tail* node and the *head* node, respectively, of an arc or walk. We classify the nodes and arcs of a strongly connected graph as follows (see Figure 2a): (i) A node v is a *join node* if its *in-degree* $d^-(v)$ satisfies $d^-(v) > 1$, 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 its *out-degree* $d^+(v)$ satisfies $d^+(v) > 1$, 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).

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 2a and Definition 12), where each macronode $\mathcal{M}_v$ 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.

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 2b):

▶ **Theorem 5** (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 2b. 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 $\mathcal{M}_v$ (see Figure 2b and Definition 14). The *maximal left-micro omnitig* is the symmetrical omnitig Wfg. By Theorem 5, 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 2b and Definition 14; 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. As such, when an 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 *different* macronodes, and thus obtain *maximal macrotigs* (Definition 17 and Figure 5). The total size of all maximal macrotigs is $O(n)$ (Theorem 20), and they are a representation of all maximal omnitigs, except for those that are univocal extensions of the arcs of $\mathcal{F}$, see below and Theorem 21.

Algorithms. Our algorithms first build the set $\mathcal{M}$ of maximal macrotigs, and then identify maximal omnitigs inside them. The set $\mathcal{F}$ of arcs univocally extending to the remaining maximal omnitigs will be the set of bivalent arcs not appearing in $\mathcal{M}$ (Theorem 21).

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, [54, 55, 11] 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 6** (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 [25]:

▶ **Theorem 7** ([25]). *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 $G \setminus f$.*

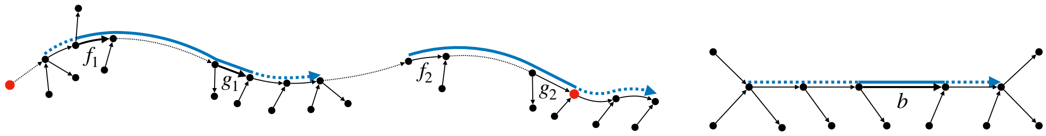


Figure 3 Any maximal omnitig is identified (in solid blue) either by a macrotig interval (from a join arc f to a split arc g; left), or by a bivalent arc b not appearing in any macrotig (right). The full maximal omnitig is obtained by univocal extension (dotted blue), extension which may go outside of the maximal macrotig.

Using the Extension Property and Theorem 7, 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 the full version of this paper 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. We use the Extension Property when building the left- and right-maximal micro omnitigs, and when identifying maximal omnitigs inside macrotigs, as follows.

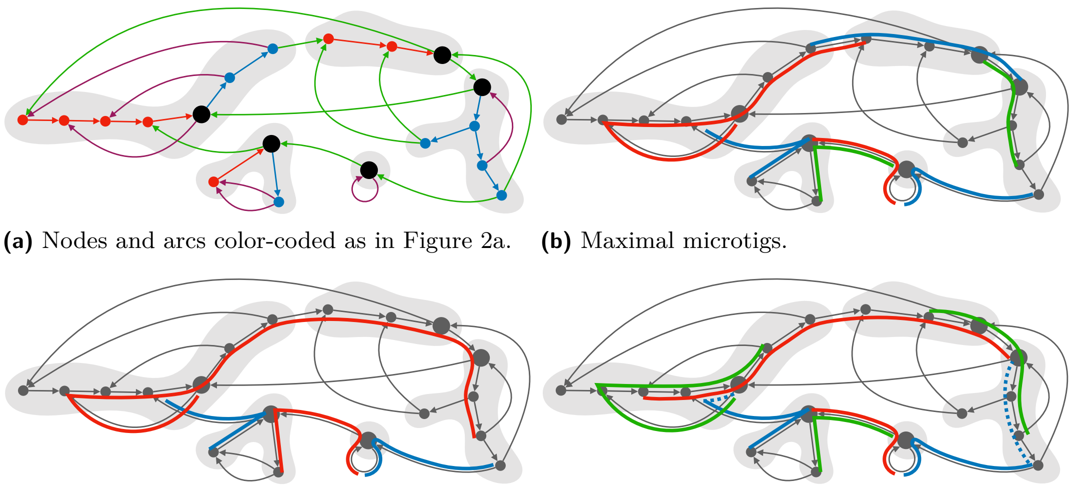
Once we have the set $\mathcal{M}$ of maximal macrotigs, we scan each macrotig with two pointers, a left one always on a join arc f, and a right one always on a split arc g (see Figure 3 and Algorithm 2). Both pointers move from left to right in such a way that the subwalk between them is always an omnitig. 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 $O(n)$. In Figure 4 we work out all these notions on a concrete example.

Comparison with previous techniques. The algorithm of [55] exhaustively extends an omnitig with every edge 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 [11] 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. In [11], 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 [25], thus the pay of $O(1)$ per omnitig extension, instead of $O(m)$ as in [54, 55, 11].

3 Basics

In this paper, a *graph* is a tuple $G = (V, E)$, where V is a finite set of *nodes*, E is a finite multi-set of ordered pairs of nodes called *arcs*. Parallel arcs and self-loops are allowed. For an arc $e \in E(G)$, we denote $G \setminus e = (V, E \setminus \{e\})$. The *reverse graph* G^R of G is obtained by reversing the direction of every arc. In the rest of this paper, we assume a fixed strongly connected graph $G = (V, E)$, with $|V| = n$ and $|E| = m \geq n$.

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



(a) Nodes and arcs color-coded as in Figure 2a. **(b)** Maximal microtigs.

(c) Maximal macrotigs.

(d) The maximal omnitigs obtained from maximal macrotigs (univocal extensions are dotted). All other maximal omnitigs are univocal extensions of the bivalent arcs not appearing in macrotigs.

Figure 4 A concrete example of the main notions of this paper. In Figures 4b–4d walks have different colors for visual distinguishability.

that W 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 *internal arcs*. A walk W is called *empty* if it has length zero, and *non-empty* otherwise. There exists exactly one empty walk $\epsilon_v = (v)$ for every node $v \in V$, and $t(\epsilon_v) = h(\epsilon_v) = v$. 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). To simplify notation, we may denote a walk $W = (v_0, e_1, v_1, \ldots, e_\ell, v_\ell)$ as a sequence of arcs, i.e. $W = e_1 \ldots e_\ell$. Subwalks of open walks are defined in the standard manner. For a closed walk $W = e_0 \ldots e_{\ell-1}$, we say that $W' = e'_0 \ldots e'_j$ 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:

▶ **Theorem 8** ([55]). *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.

Observe 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 $\mathcal{P}$ is *right-maximal* (resp., *left-maximal*) if there is no walk We (resp., eW) satisfying $\mathcal{P}$. A walk satisfying $\mathcal{P}$ is *maximal* if it is left- and right-maximal w.r.t. $\mathcal{P}$. 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 this paper our strongly connected graph G is considered to be different from a closed path, even when we do not mention it explicitly.

4 Macronodes and macrotigs

We summarize here the key results about macronodes and macrotigs, and refer the reader to the full version of this paper for the full details and proofs. 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 the full version of this paper we show how these properties can be guaranteed, by transforming any strongly connected graph G with m arcs, in time $O(m)$, into a compressed graph of constant degree and with $O(m)$ nodes and arcs. Observe that in a compressed graph all arcs are split, join or bivalent. Moreover, the following properties hold.

▶ **Observation 9.** *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.*

▶ **Lemma 10.** *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.*

▶ **Lemma 11.** *Let u be a bivalent node. No omnitig contains u twice as an internal node.*

Macronodes. We introduce a natural partition of the nodes of a compressed graph; 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. All other nodes belonging 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 2a).

▶ **Definition 12** (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 $\mathcal{M}_v$ induced by $R^+(v) \cup R^-(v)$ is called the macronode centered in v.

▶ **Lemma 13.** *In a compressed graph G, the following properties hold:*
i) *The set $\{V(\mathcal{M}_v) : v$ is a bivalent node of $G\}$ is a partition of $V(G)$.*
ii) *In a macronode $\mathcal{M}_v$, $R^+(v)$ 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.*

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. *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 combine left- and right-micro omnitigs into microtigs (which are not necessarily omnitigs themselves); recall Figure 2b.

▶ **Definition 14** (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.

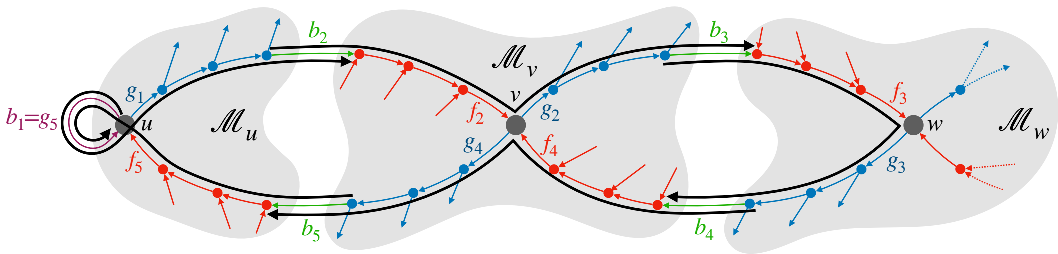


Figure 5 Three macronodes $\mathcal{M}_u, \mathcal{M}_v, \mathcal{M}_w$ (as gray areas) with arcs color-coded as in Figure 2a. Black walks mark their five maximal microtigs: $b_1 g_1 \dots b_2$, $b_i \dots f_i g_i \dots b_{i+1}$ ($i \in \{2, 3, 4\}$), $b_5 \dots f_5 g_5$ ($g_5 = b_1$). The maximal macrotig M is obtained by overlapping them on the cross-bivalent arcs b_2, b_3, b_4, b_5, i.e. $M = b_1 \dots b_2 \dots b_3 \dots b_4 \dots b_5 \dots b_1$. Notice that no arc is contained twice in M, with the exception of the self-bivalent arc b_1, appearing as the first and last arc of M.

▶ **Theorem 15** (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 $O(m)$.*

Macrotigs. Macronodes are connected with each other by bivalent arcs (Lemma 13), 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 2a.

▶ **Definition 16** (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 5.

▶ **Definition 17** (Macrotig). *Let W be any walk. W is called a* macrotig *if*

1. *W is an microtig, or*
2. *By writing $W = W_0 b_1 W_1 b_2 \dots b_{k-1} W_{k-1} b_k W_k$, where $b_1, \dots, b_k$ are all the internal bivalent arcs of W, the following conditions hold:*
   a. *the arcs $b_1, \dots, b_k$ are all cross-bivalent arcs, and*
   b. *$W_0 b_1$, $b_1 W_1 b_2$, $\dots$, $b_{k-1} W_{k-1} b_k$, $b_k W_k$ are all microtigs.*

Our structural results (including Lemmas 18 and 19 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 20.

▶ **Lemma 18.** *For any macrotig W there exists a unique maximal macrotig containing W.*

▶ **Lemma 19.** *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.*

▶ **Theorem 20** (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 $O(m)$.*

5 Maximal omnitig representation and enumeration

In the algorithms of this section we assume that the graph has constant degree. In the full version of this paper we explain how to handle the non-constant degree case.

We begin by proving the first part of Theorem 2. Theorem 20 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 18).

▶ **Theorem 21** (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 and $g \neq f$ 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 9), it is of the form $fW'g = W_1'f'g'W_2'$, with $h(f') = t(g') = u$ bivalent node. Notice that $W_1'f'g'W_2'$ is an microtig and thus it is a macrotig, by definition.

Case 2: $fW'g$ contains an internal bivalent arc b, i.e. $fW'g = W_1'bW_2'$, with W_1', W_2' non empty. By induction, $W_1'b$ and bW_2' are macrotigs and both contain a bivalent node as internal node. Suppose b is a self-bivalent arc, then both $W_1'b$ and bW_2' would contain the same bivalent node u as internal node, contradicting Lemma 11. Thus, b is a cross-bivalent arc and $W_1'bW_2'$ 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 10 and Observation 9. 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 $bW_M f_M g_M$. Notice that Wg_M is an omnitig, because M is an omnitig and the arcs of W before b are join-free, so 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 22. The number of maximal omnitigs containing an internal bivalent node is $O(n)$. This follows by Theorem 21(i), by maximality, and by the fact that the total length of maximal macrotigs is $O(n)$ (Theorem 20).

Algorithm 1 Function IsOmnitigRightExtension.

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.

  $S \leftarrow \{g' \in E(G) \mid t(g') = t(g)$ and there is a path from $h(g')$ to $h(f)$ in $G \setminus f\}$

  Return : **True** if $S = \{g\}$ and **False** otherwise

Next, we are going to prove the second, algorithmic, part of Theorem 2. By Theorem 20 we can compute the maximal macrotigs of G in time $O(m)$. We can trivially obtain in $O(m)$ time the set $\mathcal{F}$ 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 (Theorem 6).

▶ **Lemma 23.** *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.*

Proof. Consider P_W 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 $e_{i+1} \notin Z$ and $f \notin Z$. Thus Z can transformed in a forbidden path for fW, from u_i to $h(f)$. ◀

▶ **Theorem 6** (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 $G \setminus f$.*

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 $G \setminus f$. Moreover, 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 $G \setminus f$. 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 $G \setminus f$. 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 23, P does not contain f. Thus, the suffix of P from $h(g')$ is a path in $G \setminus f$ from $h(g')$ to $h(f)$. ◀

Algorithm 2 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 ← {b | b bivalent arc that does not occur in any W ∈ AllMaximalMacrotigs(G)}
2  foreach b ∈ B do output U(b)
3  foreach f*Xg* ∈ AllMaximalMacrotigs(G) do
       ▷With the notation X[f..g], we refer to the subwalk of f*Xg* starting with the
        occurrence of f in f*X (unique by Lemma 19) and ending with the occurrence
        of g in Xg* (unique by Lemma 19).
4      f ← f*, g ← nil, g' ← first split arc in Xg*
5      while g' ≠ nil do
6          while g' ≠ nil and IsOmnitigRightExtension(f, g') do
               ▷Grow X[f..g] to the right as long as possible
7              g ← g'
8              g' ← 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' ≠ 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 ← next join arc in f*X after f
```

To describe the algorithm that identifies all maximal omnitigs (Algorithm 2), we first introduce an auxiliary procedure (Algorithm 1), which uses the Extension Property (Theorem 7) and Theorem 6 to find the unique possible extension of an omnitig.

▶ **Corollary 24.** *Algorithm 1 is correct. Assuming that the graph has constant degree, we can preprocess it in time $O(m+n)$ time, so that Algorithm 1 runs in constant time.*

Maximal omnitigs are identified with a two-pointer scan of maximal macrotigs (Algorithm 2): a left pointer always on a join arc f and a right pointer always on a split arc g, recall Figure 3. For completeness, Algorithm 2 also outputs the maximal omnitigs.

▶ **Theorem 25** (Maximal omnitig enumeration)**.** *Algorithm 2 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.*

Proof. By Theorem 21, 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 18), 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 f^*Xg^*. 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 20. Each iteration of the *foreach* cycle takes time linear in the total size of the maximal macrotig X and of its output (by Corollary 24), and that the total size of all maximal macrotigs is linear, by Theorem 20. ◀

References

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. `doi:10.1109/FOCS.2015.14`.

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. `doi:10.1186/s13015-018-0122-7`.

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 Rocco A. Servedio and Ronitt Rubinfeld, editors, *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. ACM, 2015. `doi:10.1145/2746539.2746612`.

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. `doi:10.1109/FOCS.2016.56`.

6 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. `doi:10.1145/2591796.2591885`.

7 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. `doi:10.1137/1.9781611974331.ch143`.

8 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.

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

10 Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian S. Schmidt, Alexandru I. Tomescu, and Elia C. Zirondelli. Genome assembly, a universal theoretical framework: unifying and generalizing the safe and complete algorithms. *CoRR*, abs/2011.12635, 2020. `arXiv:2011.12635`.

11 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. `doi:10.1145/3341731`.

12 Massimo Cairo, Romeo Rizzi, Alexandru I Tomescu, and Elia C Zirondelli. Genome assembly, from practice to theory: safe, complete and linear-time. *arXiv preprint*, 2020. `arXiv:2002.10498`.

13 Katarína Cechlárová. Persistency in the assignment and transportation problems. *Mat. Meth. OR*, 47(2):243–254, 1998. `doi:10.1007/BF01194399`.

14 Kun-Mao Chao, Ross C. Hardison, and Webb Miller. Locating well-conserved regions within a pairwise alignment. *CABIOS*, 9(4):387–396, 1993. `doi:10.1093/bioinformatics/9.4.387`.

15 Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, June 2013. doi:10.1093/bioinformatics/btt310.

16 Marie Costa. Persistency in maximum cardinality bipartite matchings. *Oper. Res. Lett.*, 15(3):143–9, 1994. doi:10.1016/0167-6377(94)90049-3.

17 Bartłomiej Dudek and Paweł Gawrychowski. Computing quartet distance is equivalent to counting 4-cycles. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 733–743, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316390.

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

19 David Eppstein. K-best enumeration. *Bulletin of the EATCS*, 115, 2015. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/322.

20 David Eppstein. k-best enumeration. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1003–1006. Springer, New York, NY, 2016. doi:10.1007/978-1-4939-2864-4_733.

21 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. doi:10.4230/LIPIcs.ICALP.2019.55.

22 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. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892127.

23 Paolo Ferragina, Igor Nitto, and Rossano Venturini. On the bit-complexity of lempel-ziv compression. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 768–777. SIAM, 2009. doi:10.1137/1.9781611973068.

24 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, June 1992.

25 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.

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

27 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. URL: http://dblp.uni-trier.de/db/journals/bioinformatics/bioinformatics8.html#Guenoche92.

28 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. doi:10.1137/0603052.

29 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. URL: http://view.ncbi.nlm.nih.gov/pubmed/3250844.

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

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

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

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

34 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In 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*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.

35 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. doi:10.1145/3188745.3188814.

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

37 Ka-Kit Lam, Asif Khalak, and David Tse. Near-optimal assembly for shotgun sequencing with noisy reads. *BMC Bioinform.*, 15(S-9):S4, 2014. doi:10.1186/1471-2105-15-S9-S4.

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

39 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.

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

41 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. doi:10.1017/CBO9781139940023.

42 Paul Medvedev. Modeling biological problems in computer science: a case study in genome assembly. *Briefings in bioinformatics*, 20(4):1376–1383, 2019.

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

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

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

46 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.

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

48 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.

49 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.

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

51 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.

52 Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *Nature Methods*, 17(2):155–158, 2020. doi:10.1038/s41592-019-0669-3.

53 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. doi:10.1093/bioinformatics/btw450.

54 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. doi:10.1007/978-3-319-31957-5_11.

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

56 Martin Vingron and Patrick Argos. Determination of reliable regions in protein sequence alignments. *Prot. Engin.*, 3(7):565–569, 1990. doi:10.1093/protein/3.7.565.

57 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. doi:10.1137/1.9781611973730.111.

58 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 coronavirus associated with human respiratory disease in china. *Nature*, 579(7798):265–269, 2020. doi:10.1038/s41586-020-2008-3.

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