An Optimal $O(nm)$ Algorithm for Enumerating All Walks Common to All Closed Edge-covering Walks of a Graph

MASSIMO CAIRO, University of Trento, Italy
PAUL MEDVEDEV, Pennsylvania State University, Pennsylvania
NIDIA OBSCURA ACOSTA, Aalto University, Finland
ROMEO RIZZI, University of Verona, Italy
ALEXANDRU I. TOMESCU, University of Helsinki, Finland

In this article, we consider the following problem. Given a directed graph G, output all walks of G that are subwalks of all closed edge-covering walks of G. This problem was first considered by Tomescu and Medvedev (RECOMB 2016), who characterized these walks through the notion of *omnitig*. Omnitigs were shown to be relevant for the genome assembly problem from bioinformatics, where a genome sequence must be assembled from a set of reads from a sequencing experiment. Tomescu and Medvedev (RECOMB 2016) also proposed an algorithm for listing all maximal omnitigs, by launching an exhaustive visit from every edge.

In this article, we prove new insights about the structure of omnitigs and solve several open questions about them. We combine these to achieve an $O(nm)$-time algorithm for outputting all the maximal omnitigs of a graph (with n nodes and m edges). This is also optimal, as we show families of graphs whose total omnitig length is $\Omega(nm)$. We implement this algorithm and show that it is 9–12 times faster in practice than the one of Tomescu and Medvedev (RECOMB 2016).

CCS Concepts: • **Theory of computation → Graph algorithms analysis**; • **Applied computing → Bioinformatics**;

Additional Key Words and Phrases: Genome assembly, safe and complete algorithm, graph algorithm, edge-covering walk, strong bridge

ACM Reference format:
Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I. Tomescu. 2019. An Optimal $O(nm)$ Algorithm for Enumerating All Walks Common to All Closed Edge-covering Walks of a Graph. *ACM Trans. Algorithms* 15, 4, Article 48 (July 2019), 17 pages.
https://doi.org/10.1145/3341731

A shorter version of this article appeared as the conference paper (Cairo et al. 2017). This journal version contains missing proofs, additional figures, implementation details, as well as other improvements as part of the review process.
N.O.A. is partially supported by the Academy of Finland under the grant agreement number 314284. A.I.T. was partially supported by the Academy of Finland under the grant agreement number 274977.
Authors' addresses: M. Cairo, University of Trento, Department of Mathematics, Via Sommarive 14, 38123 Povo, Italy; email: cairomassimo@gmail.com; P. Medvedev, Pennsylvania State University, Department of Computer Science and Engineering, W316 Westgate Building, University Park, PA 16802, USA; email: pzm11@psu.edu; N. O. Acosta, Aalto University, Aalto SCI Computer Science, Konemiehentie 2, 02150 Espoo, Finland; email: nidia.obscuraacosta@aalto.fi; R. Rizzi, University of Verona, Department of Computer Science, Ca' Vignal 2, strada le Grazie 15, 37134 Verona, Italy; email: romeo.rizzi@univr.it; A. I. Tomescu, University of Helsinki, Department of Computer Science, P.O. Box 68, FI-00014 University of Helsinki, Finland; email: alexandru.tomescu@helsinki.fi.

1549-6325/2019/07-ART48
https://doi.org/10.1145/3341731

1 INTRODUCTION

In many computational problems, we are interested in just a simple answer to a problem, e.g., any shortest route from A to B. However, other problems are only approximately modeled by a mathematical formulation, and the best solution may depend on more complex criteria. One established way of coping with this is to enumerate all solutions, or only the first k best solutions (called *k-best enumeration*; see Eppstein (2015) for a survey). However, in many cases, this is unfeasible; for example, if there is a large number of optimal solutions.

Another way of coping with this problem is to output only the partial solutions that are common to all solutions. In the bioinformatics community, this problem has been studied for the sequence alignment problem (formulated as finding the character alignments common to all optimal alignments of two biological sequences, see, e.g., Vingron and Argos (1990), Vingron (1996), Chao et al. (1993), Friemann and Schmitz (1992), and Zuker (1991)), for the problem of filling a missing sequence inside a genome (formulated as finding the walks common to all walks of a given length in a graph (Salmela and Tomescu 2019)), or for the genome assembly problem, as we will review in more detail in Section 1.2.

In the combinatorial optimization community, the closest notion related to safety is that of *persistency*. Problems about persistency are often formulated as follows: we are given a problem on undirected graphs, and we need to partition the nodes or edges of the graph into (i) those present in all solutions to the problem (the *persistent* ones (Costa 1994)), (ii) those present in no solution, and (iii) those present in some solutions. This was first studied in connection with persistent edges present in all maximum matchings of a bipartite graph (Costa 1994), and later developed for more general assignment problems (Cechlárová 1998). It was also studied for all maximum stable sets of a graph (Boros et al. 2002). Persistency has also been generalized to *sets* of nodes or edges via the notions of *transversal* and *blocker*; see, e.g., Zenklusen et al. (2009), Costa et al. (2011), and Pajouh et al. (2014).

In this article, we study a graph problem motivated by the genome assembly problem from bioinformatics. More specifically, given a directed graph G, we need to find the walks of G that are common to all closed edge-covering walks of G. (A walk w is *closed*, or *circular*, if its first node is the same as its last node, and it is *edge-covering* if all edges of G appear in w.) The problem of characterizing such walks was solved in Tomescu and Medvedev (2016). These walks were called *omnitigs* (see Definition 2.1), and an algorithm for finding all maximal omnitigs was presented (*maximal* means that it cannot be extended to the left or right without losing the property of being an omnitig). The asymptotic running time of this algorithm was not fully analyzed in Tomescu and Medvedev (2016) except to say it was polynomial time. However, it is based on launching an exhaustive visit from every edge of the graph, and extending all such possible walks as long as they remain omnitigs. Its running time remained several orders of magnitude slower than popular assembly heuristics, and improving it was recognized as an important open problem.

We should also note that our problem formulation is similar to the problem of finding the walks common to all closed Eulerian walks of a graph (i.e., those that cover every edge of the graph exactly once). This problem was mentioned in Nagarajan and Pop (2009), which also proposed a solution to it based on the cycle-graph decomposition of the graph. This decomposition was used in Waterman (1995) to characterize Eulerian graphs with a unique closed Eulerian walk. However, despite this formal similarity, our problem requires a different set of techniques as the Eulerian case.

In Section 1.1 below, we summarize the main results of this article, and in Section 1.2, we explain the biological relevance of this problem.

1.1 Contributions and Approach

The main result of this article is an algorithm (Algorithm 3 from Section 4.4) running in time $O(nm)$ for outputting all maximal omnitigs of a graph (m is the number of edges, n is the number of nodes, and from here onward in this article, all graphs are directed). This algorithm is also *optimal*, in the sense that there are families of graphs for which the total length of their omnitigs is $\Omega(nm)$ (see, e.g., Figure 9).

This algorithm is based on three insights.

(1) A structural result connecting branches of a graph (i.e., edges whose source node has out-degree at least two) with left-maximal omnitigs (Theorem 3.7 from Section 3). In particular, there can be only one left-maximal omnitig ending with a given branch, and the structure of such omnitigs is almost fully characterizable. This also implies that the number of maximal omnitigs is at most m and their individual lengths are bounded by $3n - 1$. We also give families of graphs that achieve these upper bounds, showing that they are tight. Previously, only an upper bound of nm was known on the number of maximal omnitigs and an upper bound of nm on their lengths (Tomescu and Medvedev 2016).
(2) A partial order between branches, based on whether or not they are connected by "simple" omnitigs (Definition 4.2), which we prove to be acyclic. This allows us to reuse computation when recursively computing the left-maximal omnitig ending with a given branch.
(3) A connection between omnitigs and strong bridges of a graph (i.e., those edges whose removal disrupts strong connectivity (Italiano et al. 2012)). In particular, omnitigs that do *not* start with a strong bridge are easy to find (Lemma 4.7). Since there are at most $O(n)$ strong bridges in a graph, this implies that also the number of hard cases is $O(n)$, and not $O(m)$.

We also implement the new algorithm, and show in Section 5 that it is 9–12 times faster in practice than the one of Tomescu and Medvedev (2016). Finally, at the end of Section 4, we demonstrate that the Y-to-V transformation, used as pre-processing step in the implementation of Tomescu and Medvedev (2016) to simplify the input, can result in some omnitigs not being present in the output. This transformation is a well-known method (e.g., Medvedev et al. (2007), Jackson (2009), and Kingsford et al. (2010)) for reducing the graph used in assembly.

1.2 Biological Motivation

This section presents the relevance of our problem to the genome assembly problem from bioinformatics. The rest of the article can be read independently from this section.

Genome assembly is the problem of reconstructing a genome sequence from a set of reads from a sequencing experiment. Genome graphs have been the basis of most assembly algorithms. There is the *edge-centric de Bruijn graph* (Idury and Waterman 1995; Pevzner et al. 2001), where every k-mer (string of length k) of the reads becomes a node and every $(k + 1)$-mer of the reads becomes an edge, or the *node-centric de Bruijn graph*, where the nodes are the same but the edges are $(k - 1)$-overlaps between nodes. In a *string graph*, every read becomes a node and large enough non-transitive overlaps between reads are represented as edges (Myers 2014; Simpson and Durbin 2012). In Tomescu and Medvedev (2016), these graphs were unified under the "genome graph" model. Theoretical formulations of the assembly problem define what a *genome reconstruction* is: typically, this is a walk in a genome graph, subject to some constraints. For example, a genome reconstruction could be a closed edge-covering walk (Iu et al. 1988; Narzisi et al. 2014; Pevzner 1989), or a closed Eulerian walk (Kapun and Tsarev 2013; Medvedev and Brudno 2009; Medvedev et al. 2007; Nagarajan and Pop 2009).

However, algorithms to find an entire genome reconstruction are rarely implemented in practice because there is usually more than one valid genome reconstruction. When assemblers have no way to distinguish different reconstructions, they instead output *contigs*, which are stretches of DNA that are assumed to be in the genome. To bridge theory and practice, Tomescu and Medvedev (2016)) proposed an alternative formulation of the contig assembly problem. A string is considered *safe* if it is guaranteed to occur in every valid genome reconstruction. A contig assembly algorithm should ideally be safe (i.e., only outputting safe strings) and *complete* (i.e., every safe string should be output by the algorithm).

The notion of a safe and complete algorithm embodies several previous results. Contig assembly was first approached by finding *unitigs* (Kececioglu and Myers 1995), namely those paths whose internal nodes have in- and out-degree one. Later, some generalizations of unitigs have been considered. For example, Pevzner et al. (2001) considered paths whose internal nodes have out-degree one, with no restriction on their in-degree; Medvedev et al. (2007), Jackson (2009), and Kingsford et al. (2010) considered the unitigs of a genome graph simplified with the so-called *Y-to-V transformation* (we further discuss this at the end of Section 4). Although no underlying notion of genomic reconstruction was explicit in these studies, it can be shown that the resulting paths are safe for closed edge-covering walks. However, as (Tomescu and Medvedev 2016) notices, such approaches do not find all the safe strings. Also other studies have indeed given safe and complete algorithms for some reconstruction notions. Nagarajan and Pop (2009)) attribute to Waterman (1995) the characterization of the walks common to all closed Eulerian walks. For edge-weighted genome graphs, Nagarajan and Pop (2009) claim that a simple algorithm exists for finding all those walks common to all *shortest* closed edge-covering walks.

Tomescu and Medvedev (2016)) considered the genomic reconstruction notion of a closed edge-covering walk. This model is strictly more general than the above two ones, and, thus, safe strings for it are also safe for them. Moreover, it is also more realistic because the Eulerian notion assumes that all positions in the genome are sequenced exactly the same number of times, while the minimality criterion from other notions may over-collapse repeated regions. However, it still assumes that the reads are error-free, single-stranded, come from a circular genome, and every position in the genome appears in some read. We refer to Tomescu and Medvedev (2016) and the experimental results therein for further details on the practical merits of omnitigs for the genome assembly problem.

2 BACKGROUND AND NOTATION

In this article, a *graph* is a tuple $G = (V, E, s, t)$, where V is a finite set of *nodes*, E is a finite set of *edges*, and $s, t : E \to V$ assign to each edge $e \in E$ its *source node* $s(e)$ and its *destination node* $t(e)$. Parallel edges and self-loops are allowed. We say that an edge e goes *from* $s(e)$ *to* $t(e)$. The *reverse graph* of G is defined as $G^R = (V, E, t, s)$.

A *walk* on 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 $v_0, v_1, \ldots, v_\ell \in V$ are nodes and each e_i is an edge from v_{i-1} to v_i. We say that w goes *from* $s(w) = v_0$ *to* $t(w) = v_\ell$ and has *length* $|w| = \ell$. A walk w is called *empty* if $|w| = 0$, and *non-empty*, otherwise. (There exists exactly one empty walk $\epsilon_v = (v)$ for every node $v \in V$, and $s(\epsilon_v) = t(\epsilon_v) = v$.) A walk w is called *closed* if it is non-empty and $s(w) = t(w)$; otherwise, it is *open*. A *path* is a walk whose nodes $v_0, v_1, \ldots, v_\ell$ are all distinct, except that $v_\ell = v_0$ is allowed (in which case, we have either a closed or an empty path). A graph is *strongly connected* if there is a path (or, equivalently, a walk) from any node to any other node.

In the rest of this article, a strongly connected graph $G = (V, E, s, t)$ is given, with $|V| = n$ and $|E| = m \geq n$. We adopt the following conventions. Letters u, v denote nodes; letters e, f, g, h denote edges, which are identified with the corresponding length-1 walks; letters p, q, r denote paths;

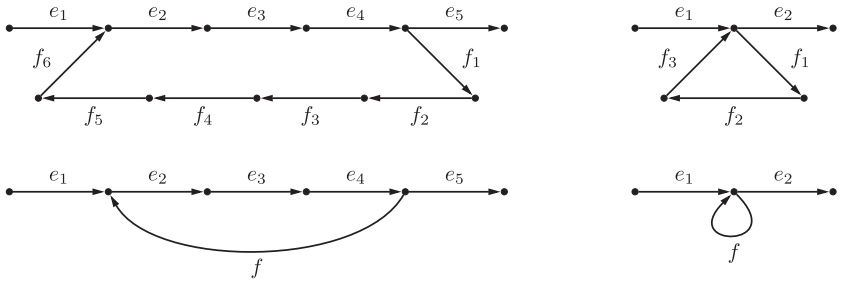

Fig. 1. Examples of walks $e_1 \cdots e_\ell$, which are not omnitigs due to the existence of a path p satisfying the conditions of Definition 2.1. In the first row, $p = f_1 \cdots f_{|p|}$ with $|p| > 1$. In the second row, $p = f$. In the left column, p is a non-empty open path. In the right column, p is a closed path.

and letters w, x, y, z denote generic walks (each letter possibly with subscripts or superscripts). Juxtaposition ww' denotes the concatenation of walks w and w', where $t(w) = s(w')$ is implicitly assumed. We start from the following definition of omnitigs offered in Tomescu and Medvedev (2016). See also Figure 1.

Definition 2.1 (Omnitig). A non-empty walk $w = e_1 \cdots e_\ell$ is an *omnitig* if, for every $1 \leq i < j \leq \ell$, there is no non-empty path from $s(e_j)$ to $t(e_i)$, with first edge different from e_j, and last edge different from e_i.

The main result from Tomescu and Medvedev (2016) is that those walks that are sub-walks of all closed edge-covering walks of a strongly connected graph are precisely its omnitigs. Clearly every edge is an omnitig and any proper subwalk of an omnitig is an omnitig. Figure 1 illustrates examples of walks that are not omnitigs. An omnitig w is *right-maximal* (resp., *left-maximal*) if there is no walk we (resp., ew), which is an omnitig. An omnitig is *maximal* if it is both left- and right-maximal. We note that in Tomescu and Medvedev (2016), two types of omnitigs were considered, depending on the genome model used. Here, we use omnitigs to refer the edge-centric omnitigs from Tomescu and Medvedev (2016).

3 STRUCTURE OF MAXIMAL OMNITIGS

In this section, we prove some structural properties of maximal omnitigs. To better understand the ways in which omnitigs might possibly overlap, we propose the notion of *branch* and *univocal walk*. A node u is called *branching* if its out-degree is more than one. In this case, any edge e with $s(e) = u$ is called a *branch*, and any two distinct edges $e \neq e'$ with $s(e) = s(e') = u$ are called *siblings*. The set of all branches is denoted by $B \subseteq E$. An edge is called an *R-branch* if it is a branch in G^R. A walk is called *univocal* if none of its edges is a branch and *R-univocal* if none of its edges is an R-branch. Figures 2 and 3 illustrate these definitions.

We start by showing some facts about branches and univocal walks.

Lemma 3.1. *If G contains at least a branch, then every univocal walk is an open path.*

Proof. A minimal counterexample is a univocal closed path p. Since every path from $s(p)$ is a prefix of p, and G is strongly connected, then p contains every node in the graph, and there are no branches. □

Lemma 3.2. *If w is an omnitig and q is a univocal path from $t(w)$, then wq is an omnitig.*

Proof. Let p be a path certifying that wq is not an omnitig by Definition 2.1. If $s(p)$ is a node of q, then a whole suffix of q is a prefix of p, since q is univocal; in this way, the property that the

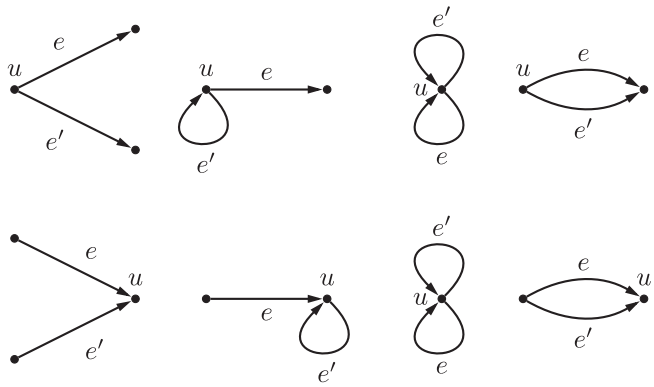

Fig. 2. Examples of edges e and e', which are sibling branches (first row) and R-branches (second row). Several cases involving parallel edges and self-loops are depicted.

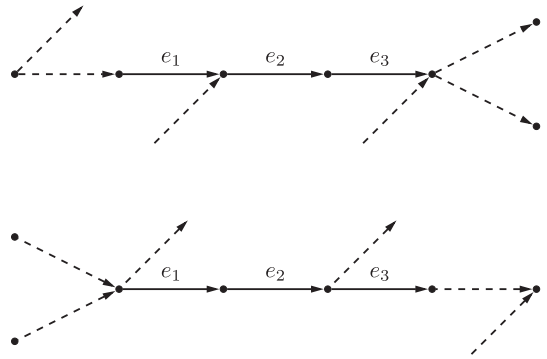

Fig. 3. Examples of univocal walk (top, solid edges only) and R-univocal walk (bottom, solid edges only). Dashed edges are allowed to exist in the graph.

first edge of p differs from e_j would be contradicted. Therefore, $s(p)$ is a node of w, but then p is a path actually certifying that w is not an omnitig—again, a contradiction. □

Lemma 3.3. *Every left-maximal omnitig contains a branch.*

Proof. Let w be a counterexample, i.e., a left-maximal omnitig, which is univocal. Let e be any edge with $t(e) = s(w)$ (at least one exists since G is strongly connected). The edge e is an omnitig, and thus, by Lemma 3.2, ew is an omnitig, violating the left-maximality of w. □

The crucial observation underlying our algorithm is that any omnitig containing a branch can be extended in an unique way to the left to obtain a left-maximal omnitig. This is expressed in Theorem 3.7 below. To prove Theorem 3.7, we need the following lemmas.

Lemma 3.4. *Let fqe be an omnitig where q is an open path and e is a branch. Take any sibling e' of e and a closed path $e'p$ starting with e'. Then, fq is a suffix of $e'p$.*

Proof. Let fqe be a minimal counterexample. Then, fqe and qe are both omnitigs, and by minimality, q is a suffix of $e'p$, whereas fq is not. Since q is an open path, then $q \neq e'p$, so q is actually a suffix of p. Thus, we can regard $e'p$ as obtained by concatenating its suffix q to its remaining prefix r, i.e., $e'p = rq$. Here, r is a non-empty path and fulfills all conditions stated in Definition 2.1: it starts with $e' \neq e$, and ends with an edge $f' \neq f$ (otherwise, fq would be a suffix

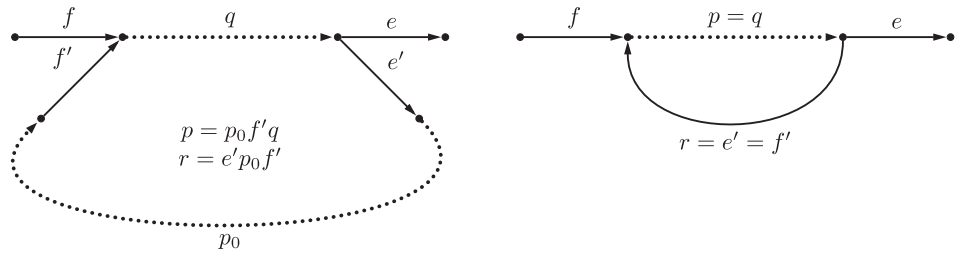

Fig. 4. Illustration of Lemmas 3.4 and 3.6 for the two cases $p \neq q$ (left) and $p = q$ (right). Dotted lines indicate (possibly empty) open paths.

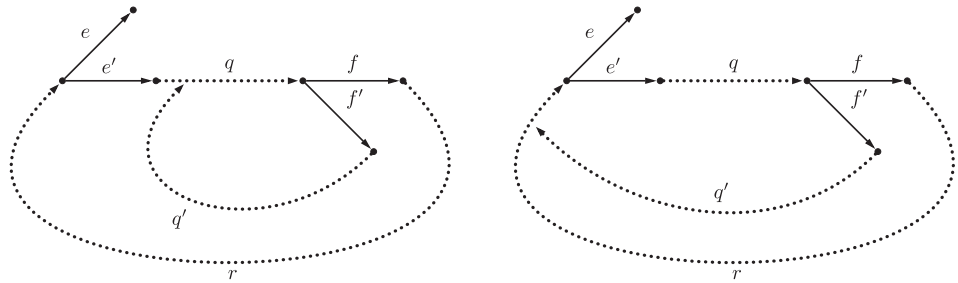

Fig. 5. Illustration of the proof of Lemma 3.5 for the two cases $t(q')$ in q (left) and $t(q')$ in r (right). Dotted lines indicate (possibly empty) open paths.

of $rq = e'p$). This shows that fqe is not an omnitig: a contradiction. See Figure 4 for an illustration of this proof. □

Lemma 3.5. *Let $e'pe$ be a walk where e and e' are siblings and $e'p$ is a closed path. Then, $e'pe$ is an omnitig iff p is univocal and e' is the only sibling of e.*

Proof. ($\Longleftarrow$) The only path satisfying Definition 2.1 must start with e', and hence be a prefix of $e'p$. ($\Rightarrow$). See Figure 5 for an illustration of this proof. First we show that e' is the only sibling of e. Let e'' be any sibling of e, and take any closed path $e''p'$. Then, $e'p$ is a suffix of $e''p'$ by Lemma 3.4. Being both closed paths, we have $e'p = e''p'$ and, in particular, $e'' = e'$.

We now prove that p is univocal. Assume not, and write $p = qfr$ where f is any branch. Let f' be a sibling of f, and $f'p'$ a closed path. Clearly, $s(f') = s(f) \neq s(e)$; hence, f' does not appear in the closed path $e'p = e'qfr$. Let q' be the shortest prefix of p', where $t(q')$ is a node of p. Observe that q' exists since $t(p') = s(f') = s(f)$ is a node of p. Moreover, the last edge of q', if any, does not appear in $e'p$. Notice that $t(q')$ is either a node of q or a node of r. If $t(q')$ is a node of q, then the path $f'q'$ shows that $e'qf$ is not an omnitig (Figure 5, left). Otherwise, if $t(q')$ is a node of r, then the path $e'qf'q'$ shows that fre is not an omnitig (Figure 5, right). In either case, $e'pe = e'qfre$ is not an omnitig: a contradiction. □

Lemma 3.6. *There is no omnitig of the form $fqrqe$ where qr is a closed path, r is non-empty, e is a branch, and f is an R-branch.*

Proof. Assume for a contradiction that $fqrqe$ is an omnitig violating the claim of the lemma. Let e' be the first edge of r. We will prove that $e' \neq e$. Write $r = e'r'$ and observe that $r'q$ is an open path, so $e'r'qe$ satisfies the hypothesis of Lemma 3.4. Let $e'' \neq e$ be a sibling of e and $e''p$ a closed path. Then, by Lemma 3.4, $e'r'q$ is a suffix of $e''p$. In fact, since both $e'r'q$ and $e''p$ are closed paths, then $e'r'q = e''p$ and $e' = e'' \neq e$, as claimed.

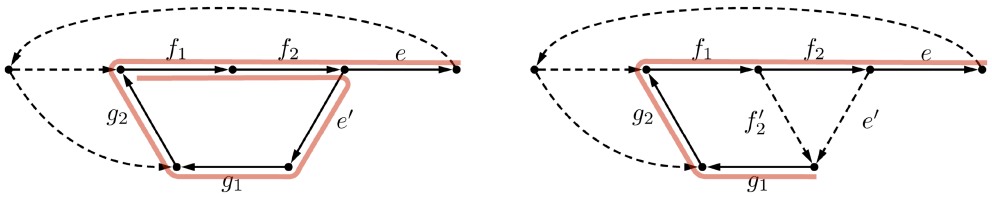

Fig. 6. Examples of graphs where the two cases of Theorem 3.7 occur, for $p = g_1g_2f_1f_2$ and $p' = f_1f_2$. In the first case (left), p is univocal and the left-maximal omnitig is $we = p'e'pe = f_1f_2e'g_1g_2f_1f_2e$. In the second case (right), p is not univocal due to the edge f_2', and the left-maximal omnitig is $we = g_1g_2f_1f_2e$. Omnitigs we are shown in red and have solid edges.

The very same argument applies on the reverse graph, since the notion of omnitig is symmetric, as well as the statement of the lemma. Therefore, also the last edge f' of r is distinct from f. Now, r is a non-empty path with first edge $e' \neq e$ and last edge $f' \neq f$. Hence, r satisfies the conditions of Definition 2.1, showing that $fqrqe$ is not an omnitig. See Figure 4 for an illustration of this proof. The following theorem characterizes the omnitigs ending with a given branch (see also Figure 6). □

Theorem 3.7. *There exists a unique left-maximal omnitig we, ending with a given branch e. Moreover, for any sibling e' of e and a closed path $e'p$, either:*

—*$we = p'e'pe$, where p' is the longest R-univocal path to $s(e)$*, or
—*we is a suffix of pe,*

where the first case occurs iff e' is the only sibling of e and p is univocal.

Proof. Consider any omnitig we. We show that we is either a suffix of pe or of the form $we = p''e'pe$, where p'' is an R-univocal path. This suffices to show that there is a unique left-maximal omnitig we, and that one of the two cases occurs.

If w is an open path, then we is a suffix of pe by Lemma 3.4. Otherwise, take the shortest suffix $e''p$ of w, which is not an open path. Since p is an open path ($e''p$ is the shortest suffix of w which is not), then $e'' = e'$ by Lemma 3.4.

Hence, a minimal counterexample for our claim is an omnitig of the form $we = fqe'pe$ where q is R-univocal (hence, an open path by Lemma 3.1 applied to the reversed graph) and f is an R-branch. Since $t(q) = t(p)$ and q is R-univocal, then q is a suffix of $e'p$. In fact, q is a suffix of p since it is open. Hence, we can write $e'p = rq$, where r is non empty, and $we = fqrqe$, violating Lemma 3.6.

Finally, the conditions in which the first case occurs are stated in Lemma 3.5, noticing that $p'e'pe$ is an omnitig iff $e'pe$ is an omnitig, by Lemma 3.2 applied in the reverse graph. □

Corollary 3.8. *There are at most m maximal omnitigs.*

Proof. Any maximal omnitig has a branch by Lemma 3.3; hence, it has the form $w = w'er$, where e is its last branch and r is univocal. By Theorem 3.7, w' is uniquely determined by e, and, by Lemma 3.2, r is the longest univocal path from $t(e)$, also uniquely determined by e. In conclusion, every omnitig has a last branch and every branch is the last branch of at most one maximal omnitig. □

Corollary 3.9. *Every maximal omnitig traverses any node at most three times, and thus has length at most $3n - 1$.*

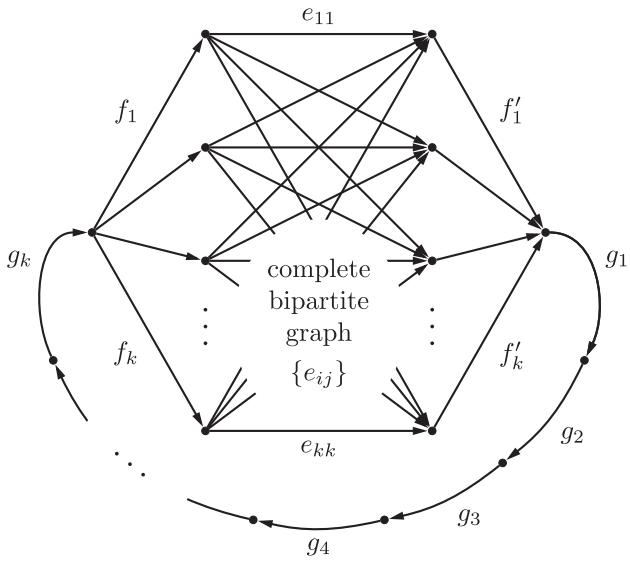

Fig. 7. A family of dense graphs G_k parametrized by $k \geq 1$ where there are $\Theta(k)$ nodes and $\Theta(k^2)$ edges, and the total length of maximal omnitigs is $\Theta(k^3)$. This shows that the bound given in Corollary 3.10 is tight, in the dense case. Indeed, the walk $w_{ij} = g_1 \cdots g_k f_i e_{ij} f'_j g_1 \cdots g_k$ is a maximal omnitig, for $1 \leq i, j \leq k$, and has length $2k + 3$.

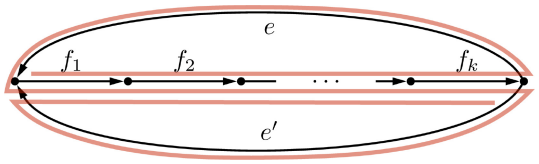

Fig. 8. A family of graphs parametrized by $k \geq 0$ where the bound given in Corollary 3.9 is tight. Let $p = f_1 \cdots f_k$. The maximal omnitigs are $pepe'p$ and $pe'pep$: both traverse each node exactly three times; $pepe'p$ is marked in red.

Proof. Any maximal omnitig has the form $w = w'er$ where e is its last branch. By Theorem 3.7, either w' is an open path, or $w = p'e'per$ where p', p, r are univocal, and hence open paths by Lemma 3.1. Consider that open paths visit each node at most once. □

Corollary 3.10. *The total length of maximal omnitigs is $O(nm)$.*

In a complete graph with node set V, $|V| \geq 3$, and edge set $V \times V$, every single edge is a maximal omnitig; hence, the bound given in Corollary 3.8 is tight. Figures 7 to 9 demonstrate graph families showing that the bounds of Corollary 3.9 and Corollary 3.10 are also tight. That is, they contain maximal omnitigs of length $3n - 1$, and the total length of the maximal omnitigs can be $\Omega(nm)$.

4 THE ALGORITHM

4.1 Extending Omnitigs

We start by considering a procedure LongestSuffix that takes an omnitig w' and an edge e with $s(e) = t(w')$, and computes the longest suffix of $w = w'e$ that is still an omnitig. A pseudo-code for such a procedure is shown in Algorithm 1 below, and it is an adaptation of the ideas given in Tomescu and Medvedev (2016).

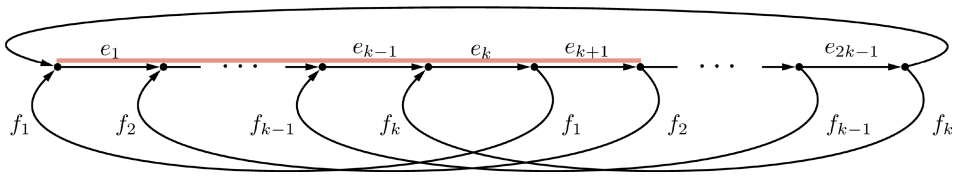

Fig. 9. A family of sparse graphs G_k parametrized by $k \geq 1$ where there are $\Theta(k)$ nodes and edges, and the total length of maximal omnitigs is $\Theta(k^2)$. This shows that the bound given in Corollary 3.10 is tight, in the sparse case. Indeed, the walk $w_i = e_i e_{i+1} \cdots e_{i+k} e_{i+k}$ is a maximal omnitig, for $1 \leq i \leq k-1$, and has length $k+1$; walk w_1 is marked in red.

ALGORITHM 1: Function LongestSuffix.

1 **Function** LongestSuffix(w)
  Input: A non-empty walk $w = w'e$ where w' is an omnitig and e is a branch.
  Returns: The longest suffix of w, which is an omnitig.
2 Denote $w = w'e = f_1 \cdots f_\ell e$.
3 Compute the set $S_e \subseteq V$ of nodes reachable from $s(e)$ without using e.
4 Let I be the largest index $i \in \{1, \ldots, \ell\}$ such that there exists an edge $g \notin \{e, f_i\}$ with $s(g) \in S_e$ and $t(g) = t(f_i)$, taking $I = 0$ if no such index exists.
5 **return** $f_{I+1} \cdots f_\ell e$

Lemma 4.1. *The function* LongestSuffix *can be implemented in* $O(m)$.

Proof. We need to show that Algorithm 1 is correct and takes $O(m)$ time. We begin by proving correctness.

Clearly, $f_{I+1} \cdots f_\ell e$ is a suffix of $w'e$. We first show it is also an omnitig. Assume the contrary. Then, by Definition 2.1, and given that $f_{I+1} \cdots f_\ell$ is an omnitig (since it is a suffix of w'), there exists an i with $I+1 \leq i \leq \ell$ and a non-empty path $r = r'g$ with $s(r) = s(e)$ and $t(r) = t(f_i)$ with first edge different from e and last edge g different from f_i. Also, r does not contain e, since it is a path, and in particular $g \neq e$. Finally, the path r' certifies that $t(r') \in S_e$. Hence, the index $i > I$ contradicts the choice of I.

We claim that, when $I > 0$, then $f_I \cdots f_\ell e$ is not an omnitig. By the definition of I, we know there exists a path r avoiding e from $s(e)$ to $s(g)$, where $g \notin \{e, f_i\}$ and $t(g) = t(f_I)$. If rg is a path, then it certifies our claim, by Definition 2.1. Thus, we assume $r = r_1 r_2$ with $t(r_1) = t(g) = s(r_2)$. If the last edge of r_1 is different from f_I, then r_1 is a path that certifies our claim, by Definition 2.1. We, hence, assume that f_I is the last edge of r_1; hence, the path r_2 does not go through $s(f_I)$. Let r_2' be the shortest non-empty suffix of r_2 with $s(r_2')$ belonging to $f_I \cdots f_\ell e$. Notice that $s(r_2') \neq s(f_I)$, because w' is an omnitig. Moreover, r_2' has no prefix that is a suffix of $f_I \cdots f_\ell e$, because $s(e)$ is the first node of r_1, and $r_1 r_2 = r$ is a path. Therefore, $r_2' g$ is a path that certifies our claim by Definition 2.1.

As for the running time, the computation of S_e at line 3 can be performed in $O(m)$ time with a graph search such as a depth-first search (DFS) or a breath-first search (BFS). The index I can be computed by scanning, for $i = 1, \ldots, \ell$, all the edges g with $t(g) = t(f_i)$. This takes $O(m)$ total time thanks to Corollary 3.9. □

4.2 A Well-Founded Order on Branches

The strategy of our full algorithm is to first pick a branch e, since, by Lemma 3.3, every maximal omnitig contains one, and then construct the only left-maximal omnitig ending with e, according

to Theorem 3.7. To this end, we may need to compute the longest suffix of $e'p$ that is an omnitig; however, this could require quadratic time to output a single left-maximal omnitig. Instead, we show that it is possible to recycle the computational effort among different branches, in order to pay linear time per-branch. We introduce the following notion of order between branches.

Definition 4.2. For any two distinct non-sibling branches $e, f \in B$, write $f \prec e$ if there exists an omnitig fpe where p is univocal.

Lemma 4.3. *For any $e \in B$, there is at most one $f \in B$ such that $f \prec e$.*

Proof. Take a sibling e' of e and a closed path $e'p$. Let f be the last branch on $e'p$ (it exists since its first edge e' is a branch), and let fq be the suffix of $e'p$ starting with f, where q is univocal. Assume $\tilde{f} \prec e$ and let $\tilde{f}\tilde{q}e$ be an omnitig with $\tilde{q}$ univocal. By Lemma 3.1, $\tilde{q}$ is an open path, and by Lemma 3.4, $\tilde{f}\tilde{q}e$ is a suffix of $e'pe$; thus, $\tilde{f} = f$ and $\tilde{q} = q$. □

Our algorithm for computing the left-maximal omnitig ending with a given branch e works as follows. We first check whether the first case of Theorem 3.7 occurs by verifying the condition provided therein. If not, then we consider the suffix fq of $e'p$ defined as in the proof of Lemma 4.3. We have two cases.

- fqe is not an omnitig. Then, an invocation of LongestSuffix(fqe) yields the only left-maximal omnitig ending with e.
- fqe is an omnitig. Then, $s(f) \neq s(e)$ since fq is open; thus, $f \prec e$. In this case, we apply the procedure recursively to the branch f, obtaining an omnitig w''. Then, the left-maximal omnitig ending with e must be a suffix of $w''qe$ and can be obtained as LongestSuffix($w''qe$).

Lemma 4.5 below is crucial in showing that the recursion is well-founded. As we will show later, thanks to memoization, this recursive application allows to reuse the computational effort and leads to a faster worst-case running time. To prove it, we will need the following observation.

Lemma 4.4. *Let $F \subseteq B$ be a non-empty set of branches, where every $f \in F$ has a sibling $f' \notin F$. There exists a closed path p, not containing any edge in F, with $s(p) = s(f)$ for some $f \in F$.*

Proof. Write $F = \{e_1, \ldots, e_k\}$ and let $S = \{s(e_i) \mid i = 1, \ldots, k\}$. For every $i = 1, \ldots, k$, fix any sibling $e'_i \notin F$ of e_i and any closed path $e'_i p_i$. Let q_i be the shortest prefix of p_i that ends with a node in S. Observe that q_i exists since $t(p_i) = s(e_i) \in S$. Moreover, q_i does not contain any edge in F. For each $i = 1, \ldots, k$, write $i \to j$ for that index $j \in \{1, \ldots, k\}$ such that $t(q_i) = s(e_j)$. Observe that, by construction, there is (exactly) one such index j for every i. In particular, this implies that the relation $\to$ contains at least a cycle, because, e.g., $\to$ can be interpreted as the edge relation on a graph with node set $\{1, \ldots, k\}$, and, thus, without sinks. Let $i_1 \to i_2 \to \cdots \to i_{t-1} \to i_t \to i_1$ be any cycle. Consider the closed walk

$$w = e'_{i_1} q_{i_1} e'_{i_2} q_{i_2} \cdots e'_{i_t} q_{i_t},$$

which starts with $s(e'_{i_1}) = s(e_{i_1}) \in S$ and does not contain any edge in F by construction. The walk w can be made into a closed path by removing any closed sub-walk strictly contained in w. □

Lemma 4.5. *The relation $\prec$ is acyclic.*

Proof. Assume that the relation $\prec$ is not acyclic. If there are cycles, then, more generally, there exist sequences $e_0 \prec e_1 \prec e_2 \prec \cdots \prec e_{t-1} \prec e_t$ where $s(e_0) = s(e_t)$ (with possibly $e_0 \neq e_t$). Take a minimal sequence of this form, where, by minimality, $e_1, \ldots, e_{t-1}$ all have distinct sources. Since $e_0 \not\prec e_t$, because $s(e_0) = s(e_t)$, it follows that $t \geq 2$. By Lemma 4.4 applied to $F = \{e_1, \ldots, e_{t-1}\}$, there exists a closed path p not containing any edge in F, with $s(p) = s(e_i)$ for some $i \in \{1, \ldots, t-1\}$, and first edge e'_i, a sibling of e_i. It suffices to show that p does not contain $e_{i-1} \in F$ because this

will bring the desired contradiction as follows. Since $e_{i-1} \prec e_i$, there exists an omnitig $e_{i-1}qe_i$ with q univocal and, by Lemma 3.1, open. By Lemma 3.4, $e_{i-1}q$ is a suffix of pe'_i, which contradicts the fact that p does not contain e_{i-1}.

If $i \geq 2$, then, by construction, p does not contain any edge in F, and, in particular, e_{i-1}. Likewise, if $i = 1$ and p does not contain e_0, then we are also done.

Otherwise, if $i = 1$ and p does contain e_0, say $p = re_0r'$, then we might as well consider the closed path $p' = e_0r'r$, which, by construction, does not contain $e_{t-1} \in F$. We then replace i with t, the path p with p', which starts with e_0, a sibling of e_t, and apply the same reasoning as above. □

4.3 Exploiting Strong Bridges

To achieve the claimed $O(nm)$ running time, we need a further improvement. We recall the definition of strong bridge in a strongly connected graph (Italiano et al. 2012).

Definition 4.6. An edge e is a *strong bridge* if, by removing e, the graph is no longer strongly connected. Equivalently, there is a pair of nodes u, v, such that every path from u to v contains e.

The lemma below states that omnitigs containing non-strong-bridges have a simpler structure.

Lemma 4.7. *If fq is an omnitig and an open path, and f is* not *a strong bridge, then q is univocal.*

Proof. A minimal counterexample is an omnitig fqe, where fqe is an open path and e is a branch. Fix a sibling e' of e, and take a closed path $e'p$ such that p does not contain f, which exists since f is not a strong bridge. By Theorem 3.7, fq is a suffix of p: a contradiction since p does not contain f. □

It is known that there are at most $2n - 2 = O(n)$ strong bridges in a given graph, and they can be computed in $O(m)$ time (Firmani et al. 2012; Italiano et al. 2012). The observation of Lemma 4.7 allows to handle those branches e, which are *not* strong bridges in a special way, and apply the full algorithm only on the $O(n)$ strong bridges. The procedure just described above is illustrated in Algorithm 2.

ALGORITHM 2: Computing the only left-maximal omnitig ending with a branch e.

```
1  Function OmnitigEndingWith(e)
       Input:   A branch e.
       Returns: The only left-maximal omnitig we.
2      Let e' be any sibling of e and e'p be any closed path starting with e'.
3      Let f be the last branch of e'p (possibly f = e') and fq the suffix of e'p starting with f.
4      Let p' be the longest R-univocal path to s(e).
5      if e has only one sibling e' and p is univocal, then return p'e'pe
6      if e is not a strong bridge, then return p'e
7      w' ← LongestSuffix(fqe)
8      if w' ≠ fqe, then return w'
9      w'' ← OmnitigEndingWith(f)                ▷ OmnitigEndingWith is memoized
10     return LongestSuffix(w''qe)
```

Lemma 4.8. *The function* OmnitigEndingWith *in Algorithm 2 is correct.*

Proof. First of all, observe that the closed path $e'p$ exists since G is strongly connected, and that q is univocal by construction. The first case of Theorem 3.7 is handled trivially by line 5. Hence, assume the second case occurs.

If e is not a strong bridge (line 6), then we can apply Lemma 3.2 and Lemma 4.7 in the reverse graph, obtaining that $p'e$ is a left-maximal omnitig. (Any omnitig $gp'e$ is an open path, by the second case of Theorem 3.7, so Lemma 4.7 applies and gp' is R-univocal, contradicting the choice of p'.)

By Lemma 3.2, fq is an omnitig (the single edge f is trivially an omnitig), so the call LongestSuffix(fqe) at line 7 conforms to the preconditions of that function. In particular, by Lemma 4.1, the returned walk w' is an omnitig. If $w' \neq fqe$, then w' is the longest suffix of pe, which is an omnitig; thus, by the second case of Theorem 3.7, it is left-maximal. Otherwise, if $w' = fqe$, then $f \prec e$ since q is univocal by construction, and fq is open (whence $s(f) \neq s(e) = t(q)$) being a suffix of p by Theorem 3.7. By the acyclicity of $\prec$ (Lemma 4.5), the recursion is well-founded; so, by induction, OmnitigEndingWith(f) returns a left-maximal omnitig w''. By Lemma 3.2, $w''q$ is also a left-maximal omnitig, and the call LongestSuffix($w''q$) at line 10 conforms to the preconditions of that function. By Lemma 4.1, the returned walk LongestSuffix($w''qe$) is a left-maximal omnitig. □

4.4 Putting All Together

The full algorithm (Algorithm 3) amounts to computing, for each branch $e \in B$, the left-maximal omnitig ending with e, and then appending the longest possible univocal suffix.

ALGORITHM 3: Computing all the maximal omnitigs.

Input: A graph G whose set of branches is B.
Returns: The set W of maximal omnitigs of G.

```
1 W ← ∅
2 for e ∈ B do
3 |   w ← OmnitigEndingWith(e)
4 |   Let p be the longest univocal path from t(e).
5 |   W ← W ∪ {wp}
6 end
7 Remove from W the non-right-maximal walks.
8 return W
```

Theorem 4.9. *Algorithm 3 is correct and can be implemented to run in time $O(nm)$.*

Proof. It is clear from Lemma 4.8 and Lemma 3.2 that Algorithm 3 terminates and returns a set W containing only left-maximal omnitigs. For correctness, we only need to show that, after the for-loop, W contains all the maximal omnitigs. Consider any maximal omnitig w. By Lemma 3.3, w contains a branch. Let e be the last branch of w, and write $w = w'ep$ where p is univocal. By Lemma 3.2, $w'e$ is left-maximal (otherwise, also $w = w'ep$ is not left-maximal), and p is the longest univocal path from $t(e)$, (otherwise, $w = w'ep$ is not right-maximal). By Lemma 4.8, in the iteration of the for-loop, relative to the branch $e \in B$, the call OmnitigEndingWith(e) returns $w'e$, and $w'ep$ is added to W.

To prove our bound on the running time, we observe that, when the function Omnitig–EndingWith returns before line 7, then it takes $O(n)$ time only. Indeed, the length of the open

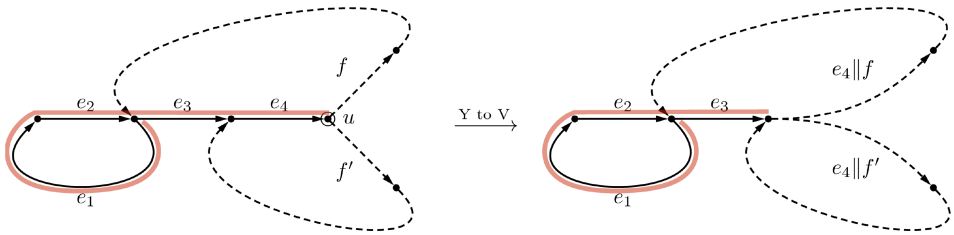

Fig. 10. Left: the walk $e_1e_2e_3e_4$ is a maximal omnitig. Right: after applying the Y-to-V reduction to node u, only the omnitig $e_1e_2e_3$ is maximal, and e_3e_4 does not appear in any omnitig.

Table 1. Wall-Clock Running Time Comparison between the Omnitig Algorithm of Tomescu and Medvedev (2016) and Our Algorithm 3

	# nodes	# edges	time by (Tomescu and Medvedev 2016)	time by Algorithm 3	# omnitigs	avg length
chr2	696,209	887,295	1,342 min	138 min	304,760	838
chr10	369,448	467,517	433 min	36 min	158,396	887
chr14	223,694	283,798	137 min	11 min	96,434	968

For fairness of comparison, the algorithms were run on a single thread, though we note that (Tomescu and Medvedev 2016) supports parallelization.

paths p and p' is $O(n)$. Moreover, when the condition at line 5 occurs, then the path p is univocal; its construction can be performed in $O(n)$ time, without running a full visit of the graph. These executions of OmnitigEndingWith account for an overall running time $O(nm)$, due to memoization, since there are $O(m)$ branches.

The execution continues after line 7 only $O(n)$ times, since the number of strong bridges is $O(n)$. In this case, the running time is dominated by the calls to LongestSuffix, which take $O(m)$ time each by Lemma 4.1. Again, due to memoization, the overall running time is $O(nm)$. The set of strong bridges is computed once at the beginning, in linear time.

It remains to show how to implement line 7 in time $O(nm)$. First, the total length of the walks in W is $O(nm)$ because to each of the $O(m)$ walks returned by OmnitigEndingWith, each of length $O(n)$ (by Corollary 3.9), we append a path, thus having length $O(n)$. One way to remove the non-right-maximal omnitigs from W is to regard each walk in W as a string over the alphabet E, construct a trie containing them, in time $O(nm)$, and remove those ending in an internal node. See Section 5 for a more direct method not using a trie. □

Finally, we would like to remark on the Y-to-V reduction. Let v be a node that has exactly one in-neighbor u and more than one out-neighbors $w_1, \ldots, w_d$. The *Y-to-V reduction applied to* v removes v and its incident edges and adds an edge from u to w_i, for all $1 \leq i \leq d$. The Y-to-V reduction was suggested as a pre-processing step to the omnitig algorithm in Tomescu and Medvedev (2016) to improve the running time. However, this reduction can destroy some omnitigs; see Figure 10. We confirm this also experimentally: in Table 1 from Section 5, we show that on chr10 data, we obtain 160 more omnitigs than in Tomescu and Medvedev (2016, Table 1).

5 IMPLEMENTATION AND EXPERIMENTS

We implemented Algorithm 3 using the code base of Tomescu and Medvedev (2016), available at https://github.com/alexandrutomescu/complete-contigs.

We now discuss additional optimizations we added to our code. They do not effect the correctness or the worst-case running time of the algorithm; however, they do improve its speed on real data.

First of all, it is not necessary to call LongestSuffix both on fqe and $w''qe$. Since it is known that the answer is a suffix of pe, in the second case of Theorem 3.7, it is sufficient to call LongestSuffix(pe) to obtain all the needed information. Indeed, despite this call may not satisfy the preconditions stated in Algorithm 1, the answer is still valid as long as it is shorter than, respectively, fqe and $w''qe$. This observation is crucial for parallelization since the computation of LongestSuffix(pe), bottleneck of the algorithm, is independent of the recursive structure and thus can be performed for each branch $e \in B$ in an easily-parallelizable for-cycle. Actually, only the length of LongestSuffix(pe) is relevant for the following execution since the path fq can be easily reconstructed from f by taking the longest univocal path q from $t(f)$. Hence, the output of the computation of LongestSuffix(pe), for each branch $e \in B$ amounts to a table of $O(m)$ numbers and edges. This phase is followed by the recursive reconstruction of the omnitigs, which is harder to parallelize but takes only time linear in the size of the output.

A second optimization, which yields significant speed-ups in practice, is to stop the DFS/BFS as soon as sufficient information is gathered. Specifically, by considering the longest R-univocal path p' to $t(p') = s(e)$, it is possible to know when OmnitigEndingWith(e) = $p'e$ without running the full visit. Indeed, as soon as we know that $s(f), s(f') \in S_e$, for two distinct R-branches f and f' with $t(f) = t(f') = s(p')$ (and distinct from e), then $p'e$ is necessarily left-maximal.

Finally, line 7 of Algorithm 3 can be implemented without a trie (as suggested in the proof of Theorem 4.9). For each wp that we add to W, let e_{wp} denote that branch that created w, namely w = OmnitigEndingWith(e_{wp}). If e_{wp} appears in another left-maximal omnitig w' not as the last branch of w', then, by Theorem 3.7, wp is a proper prefix of w'. Thus, it is not right-maximal and can be removed from W.

Each such branch that is not the last one in a maximal omnitig can be detected and marked during the execution of Algorithm 2, as follows. If we return in line 5, then we mark e' together with all branches in p'. If we return in line 6, then we mark all branches in p'. If execution reaches line 10, then we mark f.

As already discussed in Tomescu and Medvedev (2016), the notion of genomic reconstruction intended as a single closed edge-covering walk corresponds to reads (i) sequenced from a single circular chromosome, (ii) without gaps in coverage, (iii) without errors, and (iv) without reverse complements. We leave as future work the removal of these assumptions and focus the experiments in this article on the running time improvements of the new algorithm compared to the one in Tomescu and Medvedev (2016). As such, and since the practical merits of omnitigs for genome assembly were discussed in Tomescu and Medvedev (2016), we performed three experiments as follows. We circularized three reference sequences of human chromosomes 2, 10, and 14. Each had a length of 243, 136, and 107 million nucleotides, respectively. For each, we simulated reads of length $k = 55$ covering every position in the genome, and built the edge-centric de Bruijn graph, using $k = 55$. This type of genome graph is a common one on which contig assembly is performed. The algorithms were run on a machine with Intel Xeon 2.10GHz CPUs. Because the Y-to-V transformation is not omnitig-preserving, we disabled it from the code of Tomescu and Medvedev (2016).

As shown in Table 1, the new algorithm was 9–12 times faster on a single thread, suggesting that our theoretical improvements indeed translate into faster running times. For the largest dataset, our algorithm took just over 2 hours, while Tomescu and Medvedev (2016) took over 22 hours. We also observe, as expected, that the running time depends on the size of the graph and on the number of omnitigs, and not on their length.

6 CONCLUSION

Apart from its application to genome assembly, the problem addressed in this article is a fundamental graph theoretical one. It also fits into a line of research for finding all partial solutions common to natural notions of walks in graphs, such as Eulerian walks (Waterman 1995) or shortest edge-covering walks (Nagarajan and Pop 2009).

We presented here an algorithm for finding all maximal omnitigs and showed that it can be an order of magnitude faster than a previous one based on exhaustive visits. Indeed, the running time of this algorithm is $O(nm)$, a bound that cannot be improved as it is, since we also exhibited a family of instances where the total size of the maximal omnitigs is $\Omega(nm)$. It remains open whether there exists an output-sensitive algorithm with a running time linear in the size of the output.

When applied to genome assembly, our algorithm remains significantly slower than finding unitigs. However, we believe that an embarrassingly parallel implementation is possible, and that it will improve running time by another order of magnitude in practice.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for comments that improved the presentation of this article.

REFERENCES

Endre Boros, Martin C. Golumbic, and Vadim E. Levit. 2002. On the number of vertices belonging to all maximum stable sets. *Discrete Appl. Math.* 124, 1–3 (2002), 17–25. DOI: https://doi.org/10.1016/S0166-218X(01)00327-4

Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I. Tomescu. 2017. Optimal omnitig listing for safe and complete contig assembly. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4–6, 2017, Warsaw, Poland (LIPIcs)*, Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter (Eds.), Vol. 78. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 29:1–29:12. DOI: https://doi.org/10.4230/LIPIcs.CPM.2017.29

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

Kun-Mao Chao, Ross C. Hardison, and Webb Miller. 1993. Locating well-conserved regions within a pairwise alignment. *CABIOS* 9, 4 (1993), 387–396. DOI: https://doi.org/10.1093/bioinformatics/9.4.387 arXiv:http://bioinformatics.oxfordjournals.org/content/9/4/387.full.pdf+html

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

Marie-Christine Costa, Dominique de Werra, and Christophe Picouleau. 2011. Minimum d-blockers and d-transversals in graphs. *J. Comb. Optim.* 22, 4 (2011), 857–872. DOI: https://doi.org/10.1007/s10878-010-9334-6

David Eppstein. 2015. K-best enumeration. *Bulletin of the EATCS* 115 (2015). Retrieved from http://eatcs.org/beatcs/index.php/beatcs/article/view/322.

Donatella Firmani, Giuseppe F. Italiano, Luigi Laura, Alessio Orlandi, and Federico Santaroni. 2012. Computing strong articulation points and strong bridges in large scale graphs. In *Experimental Algorithms*, Ralf Klasing (Ed.). Springer Berlin, Berlin Germany, 195–207.

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

R. M. Idury and M. S. Waterman. 1995. A new algorithm for DNA sequence assembly. *J. Comput. Biol.* 2, 2 (1995), 291–306.

Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. 2012. Finding strong bridges and strong articulation points in linear time. *Theor.Comput. Sci.* 447 (Aug. 2012), 74–84. https://doi.org/10.1016/j.tcs.2011.11.011.

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

Benjamin Grant Jackson. 2009. *Parallel Methods for Short Read Assembly*. Ph.D. Dissertation. Iowa State University.

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

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

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

Paul Medvedev and Michael Brudno. 2009. Maximum likelihood genome assembly.*J. Comput. Biol.* 16, 8 (2009), 1101–1116.

Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. 2007. Computability of models for sequence assembly. In *Proceedings of the 7th International Workshop on Algorithms in Bioinformatics (WABI 2007), Philadelphia, PA, September 8–9, 2007 (Lecture Notes in Computer Science)*, Raffaele Giancarlo and Sridhar Hannenhalli (Eds.), Vol. 4645. Springer, 289–301.

Gene Myers. 2014. Efficient local alignment discovery amongst noisy long reads. In *Proceedings of the 14th International Workshop on Algorithms in Bioinformatics (WABI 2014), Wroclaw, Poland, September 8-10, 2014. (Lecture Notes in Computer Science)*, Daniel G. Brown and Burkhard Morgenstern (Eds.), Vol. 8701. Springer, 52–67.

Niranjan Nagarajan and Mihai Pop. 2009. Parametric complexity of sequence assembly: Theory and applications to next generation sequencing. *J. Comput. Biol.* 16, 7 (2009), 897–908.

Giuseppe Narzisi, Bud Mishra, and Michael C. Schatz. 2014. On algorithmic complexity of biomolecular sequence assembly problem. In *Proceedings of the 1st International Conference on Algorithms for Computational Biology (AlCoB 2014), Tarragona, Spain, July 1–3, 2014 (Lecture Notes in Computer Science)*, Adrian Horia Dediu, Carlos Martín-Vide, and Bianca Truthe (Eds.), Vol. 8542. Springer, 183–195.

Foad Mahdavi Pajouh, Vladimir Boginski, and Eduardo L. Pasiliao. 2014. Minimum vertex blocker clique problem. *Networks* 64, 1 (2014), 48–64. DOI: https://doi.org/10.1002/net.21556

Pavel A. Pevzner. 1989. L-tuple DNA sequencing: Computer analysis. *J. Biomol. Struct. Dyn.* 7, 1 (Aug. 1989), 63–73.

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

Leena Salmela and Alexandru I. Tomescu. 2019. Safely filling gaps with partial solutions common to all solutions. *IEEE/ACM Trans. Comput. Biol. Bioinf.* 16, 2 (2019), 617–626. https://doi.org/10.1109/TCBB.2017.2785831

Jared T. Simpson and Richard Durbin. 2012. Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.* 22, 3 (2012), 549–556.

Alexandru I. Tomescu and Paul Medvedev. 2016. Safe and complete contig assembly via omnitigs. In *Proceedings of the 20th Annual Conference on Research in Computational Molecular Biology (RECOMB 2016), Santa Monica, CA, April 17–21, 2016, (Lecture Notes in Computer Science)*, Mona Singh (Ed.), Vol. 9649. Springer, 152–163.

Martin Vingron. 1996. Near-optimal sequence alignment. *Curr. Opin. Struct. Biol.* 6, 3 (June 1996), 346–352. DOI: https://doi.org/10.1016/s0959-440x(96)80054-6

Martin Vingron and Patrick Argos. 1990. Determination of reliable regions in protein sequence alignments. *Protein Eng.* 3, 7 (1990), 565–569. DOI: https://doi.org/10.1093/protein/3.7.565 arXiv:http://peds.oxfordjournals.org/content/3/7/565.full.pdf+html

Michael S. Waterman. 1995. *Introduction to Computational Biology: Maps, Sequences and Genomes*. Chapman & Hall / CRC Press.

Rico Zenklusen, Bernard Ries, Christophe Picouleau, Dominique de Werra, Marie-Christine Costa, and Cédric Bentz. 2009. Blockers and transversals. *Discrete Math.* 309, 13 (2009), 4306–4314. DOI: https://doi.org/10.1016/j.disc.2009.01.006

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

Received January 2019; revised June 2019; accepted June 2019