Optimal Omnitig Listing for Safe and Complete Contig Assembly[*]

Massimo Cairo[†1], Paul Medvedev[2], Nidia Obscura Acosta[3], Romeo Rizzi[‡4], and Alexandru I. Tomescu[§5]

1 **University of Trento, Trento, Italy**
`massimo.cairo@unitn.it`

2 **The Pennsylvania State University, State College, PA, USA**
`pashadag@cse.psu.edu`

3 **Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, Helsinki, Finland**
`obscura.nidia@helsinki.fi`

4 **Department of Computer Science, University of Verona, Verona, Italy**
`romeo.rizzi@univr.it`

5 **Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, Helsinki, Finland**
`alexandru.tomescu@helsinki.fi`

Abstract

Genome assembly is the problem of reconstructing a genome sequence from a set of reads from a sequencing experiment. Typical formulations of the assembly problem admit in practice many genomic reconstructions, and actual genome assemblers usually output *contigs*, namely substrings that are promised to occur in the genome. To bridge the theory and practice, Tomescu and Medvedev [RECOMB 2016] reformulated contig assembly as finding all substrings common to all genomic reconstructions. They also gave a characterization of those walks (*omnitigs*) that are common to all closed edge-covering walks of a (directed) graph, a typical notion of genomic reconstruction. An algorithm for listing all maximal omnitigs was also proposed, by launching an exhaustive visit from every edge.

In this paper, 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].

1998 ACM Subject Classification G.2.2 Graph Theory, E.1 Data Structures, J.3 Life and Medical Sciences

Keywords and phrases genome assembly, graph algorithm, edge-covering walk, strong bridge

Digital Object Identifier 10.4230/LIPIcs.CPM.2017.29

[*] We acknowledge partial support from the Academy of Finland grant 274977 to A.I.T. and grant 284598 (CoECGR) to N.O.A., from the European Union's Horizon 2020 Marie Skłodowska-Curie grant agreement No 690941 to M.C. and A.I.T., and from NSF awards DBI-1356529, CCF-1439057, IIS-1453527, and IIS-1421908 to P.M.

[†] First author contribution.

[‡] Shared last author contribution.

[§] Shared last author contribution.

1 Introduction

Genome assembly is the problem of reconstructing a genome sequence from a set of reads from a sequencing experiment. It is one of the oldest problems in bioinformatics, but many challenges remain. For example, assemblers for novel sequence technologies such as Oxford nanopore are still only in development. Assembly of heterogeneous tumor data is also a challenge. Many of these challenges can be met by building on top of existing assembly algorithms. However, recent directions to improve the theoretical underpinning of assembly have the potential to improve assembly across a wide breadth of scenarios.

Genome graphs have been the basis of most assembly algorithms. There is the *edge-centric de Bruijn graph* [2, 15], 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 [11, 16]. In a recent paper [17], 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 (i.e., circular) walk covering every edge of the genome graph exactly once [14, 8, 13] (to be called *edge-covering* in the ongoing), or a closed Eulerian walk [9, 10, 12, 5].

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 proposed in [17] 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).

Previous work. The notion of a safe and complete algorithm embodies several previous results. Contig assembly was first approached by finding *unitigs* [6], namely those paths whose internal nodes have in- and out-degree one. Later, some generalizations of unitigs have been considered. For example, [15] considered paths whose internal nodes have out-degree one, with no restriction on their in-degree; [10, 4, 7] 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 [17] notices, such approaches do not find all the safe strings. Other studies have indeed given safe and complete algorithms for some reconstruction notions. Nagarajan and Pop [12] attribute to [18] the characterization of the walks common to all closed Eulerian walks. For edge-weighted genome graphs, [12] claims that a simple algorithm exists for finding all those walks common to all *shortest* closed edge-covering walks.

Tomescu and Medvedev [17] 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 notion 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.

In [17] a characterization of those walks common to all such genomic reconstructions was given. These walks were called *omnitigs* (see Definition 1), and an algorithm for finding all maximal omnitigs was presented. We refer to [17] for further details on the practical merits of omnitigs, as opposed to e.g., unitigs. The asymptotic running time of this algorithm was not fully analyzed in [17] 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 are omnitigs. Its running time remained several orders of magnitude slower than finding unitigs, and improving it was recognized as an important open problem.

Contributions and approach of this paper. The main result of this paper is an algorithm (Algorithm 3) 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 in this paper 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 4).

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 8). 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 [17]. This is encouraging also from a practical point of view, because the popular (maximal) unitigs have the same tight asymptotic bounds on their number and individual length (but not on their total length, which is m).
2. A partial order between branches, based on whether or not they are connected by "simple" omnitigs (Definition 13), 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 [3]). In particular, omnitigs that do *not* start with a strong bridge are easy to find (Lemma 17). 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 [17]. Finally, at the end of Section 4 we demonstrate that the Y-to-V transformation, used as pre-processing step in the implementation of [17] to simplify the input, can result in shorter maximal omnitigs. This transformation is a well-known method (e.g. [10, 4, 7]) for reducing the genome graph, maintaining the property that its unitigs spell safe strings.

2 Background and notation

In this paper, 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 $s, t \colon E \to V$ 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 $w = (v_0, e_1, v_1, e_2, \dots, v_{\ell-1}, e_\ell, v_\ell)$, $\ell \geq 0$, where $v_0, v_1, \dots, v_\ell \in V$ 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*

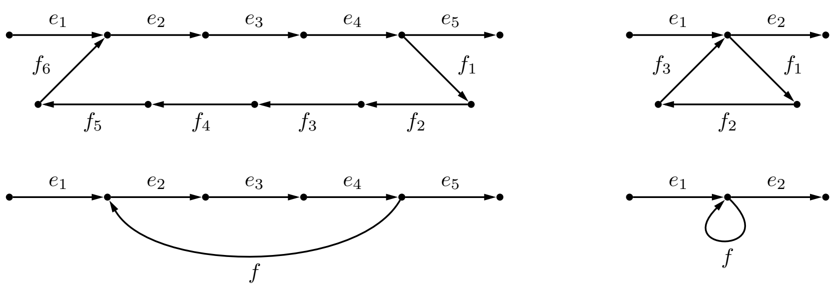

Figure 1 Examples of walks $e_1 \cdots e_\ell$ which are not omnitigs, due to the existence of a path p satisfying the conditions of Definition 1. In the first row, $p = f_1 \cdots f_{|p|}$ with $|p| > 1$. In the second row, $p = f$. In the left column, p is a non-empty open path. In the right column,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

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 paper, 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, 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 [17].

▶ **Definition 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 [17] 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 [17] two types of omnitigs were considered, depending on the genome model used. Here, we use omnitigs to refer the edge-centric omnitigs from [17].

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.

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

▶ **Lemma 2.** *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.** *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 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 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 4.** *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 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 8 below. To prove Theorem 8, we need the following lemmas.

▶ **Lemma 5.** *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 1: it starts with $e' \neq e$, and ends with an edge $f' \neq f$ (otherwise fq would be a suffix of $rq = e'p$). This shows that fqe is not an omnitig: a contradiction. ◀

▶ **Lemma 6.** *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 1 must start with e', and hence be a prefix of $e'p$. ($\Longrightarrow$). 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 5. 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. Otherwise, if $t(q')$ is a node of r, then the path $e'qf'q'$ shows that fre is not an omnitig. In either case $e'pe = e'qfre$ is not an omnitig: a contradiction. ◀

▶ **Lemma 7.** *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 5. Let $e'' \neq e$ be a sibling of e,

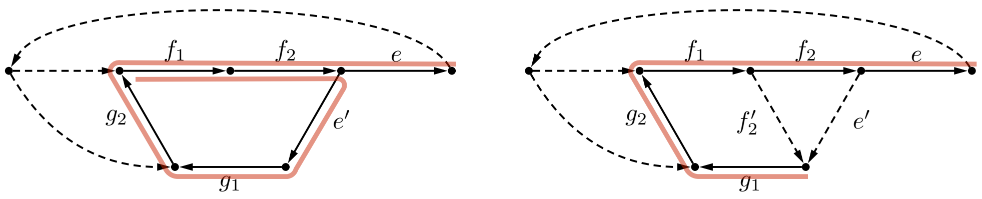

Figure 2 Example of graphs where the two cases of Theorem 8 occur, for $p = g_1 g_2 f_1 f_2$ and $p' = f_1 f_2$. In the first case (left), p is univocal and the left-maximal omnitig is $we = p'e'pe = f_1 f_2 e' g_1 g_2 f_1 f_2 e$. In the second case (right) p is not univocal due to the edge f'_2, and the left-maximal omnitig is $we = g_1 g_2 f_1 f_2 e$. Omnitigs we are shown in red and have solid edges.

and $e''p$ a closed path. Then, by Lemma 5, $e'r'q$ 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.

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 1, showing that $fqrqe$ is not an omnitig. ◀

▶ **Theorem 8.** *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 5. 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 5.

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 2 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 7.

Finally, the conditions in which the first case occurs are stated in Lemma 6, noticing that $p'e'pe$ is an omnitig iff $e'pe$ is an omnitig, by Lemma 3 applied in the reverse graph. ◀

▶ **Corollary 9.** *There are at most m maximal omnitigs.*

Proof. Any maximal omnitig has a branch by Lemma 4; hence it has the form $w = w'er$, where e is its last branch and r is univocal. By Theorem 8, w' is uniquely determined by e, and, by Lemma 3, 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 10.** *Every maximal omnitig traverses any node at most three times, and thus has length at most $3n - 1$.*

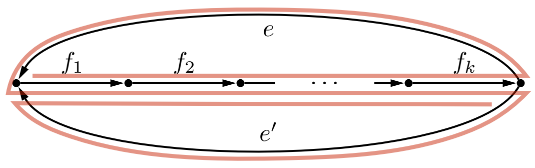

Figure 3 A family of graphs parametrized by $k \geq 0$ where the bound given in Corollary 10 is tight. Let $p = f_1 \cdots f_k$. The maximal omnitigs are $pepe'p$ and $pe'pep$: both traverse each node exactly three times; $pepe'p$ is marked in red.

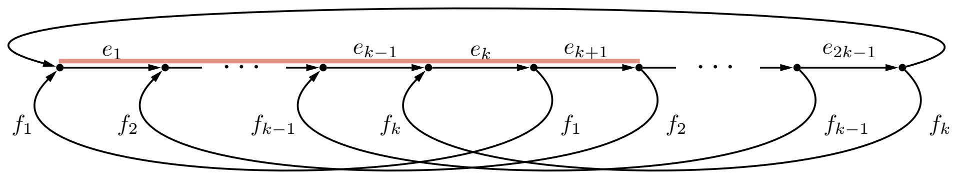

Figure 4 A family of sparse graphs G_k parametrized by $k \geq 1$ where there are $\Theta(k)$ nodes and edges, and the total length of maximal omnitigs is $\Theta(k^2)$. This shows that the bound given in Corollary 11 is tight, in the sparse case. Indeed, the walk $w_i = e_i e_{i+1} \cdots e_{i+k} e_{i+k}$ is a maximal omnitig, for $1 \leq i \leq k-1$, and has length $k+1$; walk w_1 is marked in red.

Proof. Any maximal omnitig has the form $w = w'er$ where e is its last branch. By Theorem 8, either w' is an open path, or $w = p'e'per$ where p', p, r are univocal, and hence open paths by Lemma 2. Consider that open paths visit each node at most once. ◀

▶ **Corollary 11.** *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 9 is tight. Figures 3 and 4 demonstrate graph families showing that the bounds of Corollary 10 and Corollary 11 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

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, and it is an adaptation of the ideas given in [17].

▶ **Lemma 12.** *The function* LongestSuffix *can be implemented in $O(m)$.*

The strategy of our algorithm is to first pick a branch e, since by Lemma 4 every maximal omnitig contains one, and then construct the only left-maximal omnitig ending with e, according to Theorem 8. To this end, we may need to compute the longest suffix of $e'p$ which 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 13.** 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.

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 $\hat{\imath}$ 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 $\hat{\imath} = 0$ if no such index exists.
5     return $f_{\hat{\imath}+1} \cdots f_\ell e$
```

▶ **Lemma 14.** *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 2, $\tilde{q}$ is an open path, and by Lemma 5, $\tilde{f}\tilde{q}e$ 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 8 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 14. 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 15 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.

▶ **Lemma 15.** *The relation $\prec$ is acyclic.*

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 [3].

▶ **Definition 16.** 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 17.** *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 8, fq is a suffix of p: a contradiction since p does not contain f. ◀

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      $w' \leftarrow$ LongestSuffix($fqe$)
8      if $w' \neq fqe$ then return $w'$
9      $w'' \leftarrow$ OmnitigEndingWith($f$)          ▷ OmnitigEndingWith is memoized
10     return LongestSuffix($w''qe$)
```

Algorithm 3: Computing all the maximal omnitigs.

```
1  $W \leftarrow \emptyset$
2  for $e \in B$ do
3      $w \leftarrow$ OmnitigEndingWith($e$)
4      Let $p$ be the longest univocal path from $t(e)$.
5      $W \leftarrow W \cup \{wp\}$
6  end
7  Remove from $W$ the non-right-maximal walks.
8  return $W$
```

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 [3, 1]. The observation of Lemma 17 allows to handle those branches e which are *not* strong bridges is a special way, and apply the full algorithm only on the $O(n)$ strong bridges. The procedure just described is illustrated in Algorithm 2.

▶ **Lemma 18.** *The function* OmnitigEndingWith *in Algorithm 2 is correct.*

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.

▶ **Theorem 19.** *Algorithm 3 is correct and can be implemented to run in time $O(nm)$.*

Proof. It is clear from Lemma 18 and Lemma 3 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 4, w contains a branch. Let e be the last branch of w, and write $w = w'ep$ where p is univocal. By Lemma 3, $w'e$ 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 18, 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 OmnitigEndingWith returns before line 7, then it takes $O(n)$ time only. Indeed, the length of

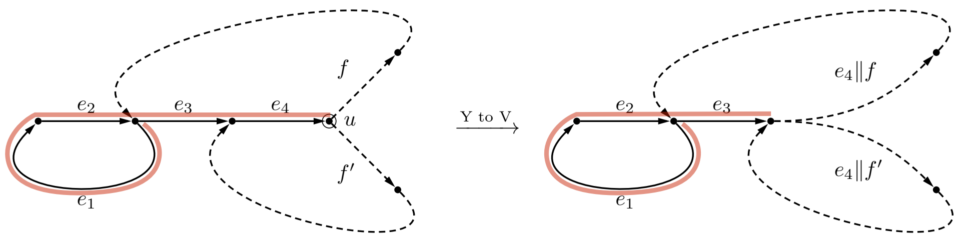

Figure 5 Left: the walk $e_1e_2e_3e_4$ is a maximal omnitig. Right: after applying the Y-to-V reduction to node u, only the omnitig $e_1e_2e_3$ is maximal, and e_3e_4 does not appear in any omnitig.

the open paths p and p' is $O(n)$. Moreover, when the condition at line 5 occurs, then the path p is univocal, and 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 12. 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 10), 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. ◀

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 [17] to improve the running time. However, this reduction can destroy some omnitigs, see Figure 5.

5 Experimental results

We implemented Algorithm 3 using the code base of [17].[1] We focused our experiments on measuring the running time improvements, since the practical merits of omnitigs for genome assembly were discussed in [17]. 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 [17]. We circularized three reference sequences of human chromosomes 2, 10, and 14. Each had a length of 243, 136 and 107 million nucleotides, respectively. We built the edge-centric de Bruijn graph for each, using $k = 55$. This is a typical genome graph on which contig assembly is performed.

As shown in Table 1, our 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 [17] took over 22 hours. We also observe, as expected, that the running time depends on the size of the graph and the number of omnitigs, and not on their length.

[1] Available at `https://github.com/alexandrutomescu/complete-contigs`.

Table 1 Wall-clock running time comparison between the omnitig algorithm of [17] and our Algorithm 3.For fairness of comparison, the algorithms were run on a single thread, though we note that [17] supports parallelization.

	# nodes	# edges	time by [17]	time by Algorithm 3	# omnitigs	avg len (bp)
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

6 Conclusion

Apart from its application to genome assembly, the problem addressed in this paper 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 [18] or shortest edge-covering walks [12]. We presented here an optimal $O(nm)$ 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. 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.

Acknowledgements. We thank Martin Milanič for pointing us to persistent solutions and blockers.

References

1 Donatella Firmani, Giuseppe F. Italiano, Luigi Laura, Alessio Orlandi, and Federico Santaroni. Computing strong articulation points and strong bridges in large scale graphs. In Ralf Klasing, editor, *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA 2012)*, volume 7276 of *LNCS*, pages 195–207, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-30850-5_18.

2 Ramana M. Idury and Michael S. Waterman. A new algorithm for DNA sequence assembly. *J. Comput. Biol.*, 2(2):291–306, 1995. doi:10.1089/cmb.1995.2.291.

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

4 Benjamin Grant Jackson. *Parallel methods for short read assembly.* PhD thesis, Iowa State University, 2009. URL: http://lib.dr.iastate.edu/etd/10704.

5 Evgeny Kapun and Fedor Tsarev. De Bruijn superwalk with multiplicities problem is NP-hard. *BMC Bioinformatics*, 14(S-5):S7, 2013. doi:10.1186/1471-2105-14-S5-S7.

6 John D. Kececioglu and Eugene W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1/2):7–51, 1995. doi:10.1007/BF01188580.

7 Carl Kingsford, Michael C. Schatz, and Mihai Pop. Assembly complexity of prokaryotic genomes using short reads. *BMC Bioinformatics*, 11(1):21, 2010. doi:10.1186/1471-2105-11-21.

8 Yuri P. Lysov, Vladimir L. Florentiev, Alexandr A. Khorlin, Konstantin R. Khrapko, and Valentine V. Shik. Determination of the nucleotide sequence of dna using hybridization with oligonucleotides. A new method. *Dokl. Akad. Nauk SSSR*, 303(6):1508–1511, 1988. URL: http://view.ncbi.nlm.nih.gov/pubmed/3250844.

9 Paul Medvedev and Michael Brudno. Maximum likelihood genome assembly. *J. Comput. Biol.*, 16(8):1101–1116, 2009. doi:10.1089/cmb.2009.0047.

10 Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. In Raffaele Giancarlo and Sridhar Hannenhalli, editors, *Proceedings of the 7th International Workshop on Algorithms in Bioinformatics (WABI 2007)*, volume 4645 of *LNCS*, pages 289–301. Springer, 2007. doi:10.1007/978-3-540-74126-8_27.

11 Gene Myers. Efficient local alignment discovery amongst noisy long reads. In Daniel G. Brown and Burkhard Morgenstern, editors, *Proceedings of the 14th International Workshop on Algorithms in Bioinformatics (WABI 2014)*, volume 8701 of *LNCS*, pages 52–67. Springer, 2014. doi:10.1007/978-3-662-44753-6_5.

12 Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly: Theory and applications to next generation sequencing. *J. Comput. Biol.*, 16(7):897–908, 2009. doi:10.1089/cmb.2009.0005.

13 Giuseppe Narzisi, Bud Mishra, and Michael C. Schatz. On algorithmic complexity of biomolecular sequence assembly problem. In Adrian-Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *Proceedings of the 1st International Conference on Algorithms for Computational Biology (AlCoB 2014)*, volume 8542 of *LNCS*, pages 183–195. Springer, 2014. doi:10.1007/978-3-319-07953-0_15.

14 Pavel A. Pevzner. L-Tuple DNA sequencing: computer analysis. *J. Biomol. Struct. Dyn.*, 7(1):63–73, August 1989. URL: http://www.tandfonline.com/doi/abs/10.1080/07391102.1989.10507752.

15 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci. U.S.A.*, 98(17):9748–9753, 2001. doi:10.1073/PNAS.171285098.

16 Jared T. Simpson and Richard Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.*, 22(3):549–556, 2012. doi:10.1101/GR.126953.111.

17 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly via omnitigs. In Mona Singh, editor, *Proceedings of the 20th Annual Conference on Research in Computational Molecular Biology (RECOMB 2016)*, volume 9649 of *LNCS*, pages 152–163. Springer, 2016. doi:10.1007/978-3-319-31957-5_11.

18 Michael S. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes*, volume 1 of *Chapman & Hall/CRC Interdisciplinary Statistics*. CRC Press, 1995. URL: https://www.crcpress.com/9780412993916.