Safe and Complete Contig Assembly Through Omnitigs

Tomescu, Alexandru I.

2017-06-01

Tomescu , A I & Medvedev , P 2017 , ' Safe and Complete Contig Assembly Through Omnitigs ' , Journal of Computational Biology , vol. 24 , no. 6 , pp. 590-602 . https://doi.org/10.1089/cmb.2016.0141

http://hdl.handle.net/10138/313118
https://doi.org/10.1089/cmb.2016.0141

acceptedVersion

Safe and complete contig assembly via omnitigs*

Alexandru I. Tomescu[1] and Paul Medvedev[2,3,4]

[1]Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki
P.O. Box 68, FI-00014, Helsinki, Finland
E-mail: tomescu@cs.helsinki.fi, Fax: +358 9 876 4314
[2]Department of Computer Science and Engineering, The Pennsylvania State University, USA
[3]Department of Biochemistry and Molecular Biology, The Pennsylvania State University, USA
[4]Genome Sciences Institute of the Huck, The Pennsylvania State University, USA

Abstract. Contig assembly is the first stage that most assemblers solve when reconstructing a genome from a set of reads. Its output consists of contigs – a set of strings that are promised to appear in any genome that could have generated the reads. From the introduction of contigs 20 years ago, assemblers have tried to obtain longer and longer contigs, but the following question remains: given a genome graph G (e.g. a de Bruijn, or a string graph), what are *all* the strings that can be safely reported from G as contigs? In this paper we answer this question using a model where the genome is a circular covering walk. We also give a polynomial time algorithm to find such strings, which we call omnitigs. Our experiments show that omnitigs are 66% to 82% longer on average than the popular unitigs, and 29% of dbSNP locations have more neighbors in omnitigs than in unitigs.

Keywords: genome assembly, contig assembly, safe and complete algorithm, graph algorithm, omnitig

*A preliminary version of this paper appeared in RECOMB 2016

1 Introduction

The genome assembly problem is to reconstruct the sequence of a genome using reads from a sequencing experiment. It is one of the oldest bioinformatics problems; nevertheless, recent projects such as the Genome 10K have underscored the need to further improve assemblers [9]. Current algorithms face numerous practical challenges, including scalability, integration of new data types (e.g. PacBio), and representation of multiple alleles. While handling these challenges is extremely important, assemblers do not produce optimal results even in very simple and idealized scenarios. To address this, several papers have developed better theoretical underpinnings [10,26,22,37,23,40], often resulting in improved assemblers in practice [32,42,38,1].

In most theoretical studies, the assembly problem is formulated as finding one genomic reconstruction, i.e. a single string that represents the sequence of the genome. However, the presence of repeats means that a unique genomic reconstruction usually does not exist. In practice, assemblers instead output several strings, called *contigs*, that are "promised" to occur in the genome. We refer to this restatement of the genome assembly problem as *contig assembly.* Contigs can then be used to answer biological questions (e.g. about gene content) or perform comparative genomic analysis. When mate pairs are available, contigs can be fed to later assembly stages, such as scaffolding [34,2,20] and then gap filling [35,3].

Assemblers implement different strategies for finding contigs. The common strategy is to find *unitigs*, an idea that can be traced back to 1995 [15]. Unitigs have the desired property that they can be mathematically proven to occur in *all* possible genomic reconstructions, under clear assumptions on what "genomic reconstruction" means. We will refer to strings that satisfy such a property as being *safe* (Definition 3), and will say that a contig assembly algorithm is *safe* if it outputs only safe strings. Though most assemblers have a safe strategy at their core, they also incorporate heuristics to handle erroneous data and extend contig length (e.g. bubble popping, tip removal, and path disambiguation). Properties of such heuristics, however, are difficult to prove, and this paper will focus on core algorithms that are safe.

While the unitig algorithm is safe, it does not identify *all* possible safe strings (see Figure 2). An improved safe algorithm was used in the EULER assembler [32], and further improvements were suggested based on iteratively simplifying the graph used for assembly [32,21,12,17]. However, we will show that these algorithms still do not always output all the safe strings. In fact, since the initial consideration of contig assembly 20 years ago, the fundamental question of finding *all* the safe strings of a graph remains poorly studied.

In this paper, we answer this question by giving a polynomial-time algorithm for outputting *all* the safe strings in the common genome graph models (de Bruijn and string graphs) when the genome is a circular covering walk (Section 6). The key ingredient for this result is a graph-theoretic characterization of the walks that correspond to safe strings (Section 5). We call such walks *omnitigs* and our algorithm the *omnitig algorithm.* In our experiments on de Bruijn graphs built from data simulated according to our assumptions, maximal omnitigs are on average 66% to 82% longer than maximal unitigs, and 29% of dbSNP locations have more neighbors in omnitigs than in unitigs.

Our results are naturally limited to the context of our model and its assumptions. Intuitively, we assume that (i) the sequenced genome is circular, (ii) there are no gaps in coverage, and (iii) there are no errors in the reads. A mathematically precise definition of our model will be presented in Section 4. We argue that such a model is necessary if we want to prove even the simplest results about unitigs (Section 4). Similar to previous studies, we also do not deal with multiple chromosomes or the double-strandedness of DNA and assume the genome is represented by a covering walk. As with previous papers that developed better theoretical underpinnings [31,10,26,23], it is necessary to prove results in a somewhat idealized setting. While this paper falls short of analyzing real data, we believe that omnitigs can be incorporated into practical genome analysis and assembly tools – similar to the way that error-free studies of de Bruijn [31] and paired de Bruijn graphs [23] became the basis of practical assemblers [32,40,1].

2 Related work

The number of related assembly papers is vast, and we refer the reader to some surveys [24,28]. For an empirical evaluation of the correctness of several state-of-the-art assemblers, see [36]. Here, we discuss work on the theoretical underpinnings of assembly.

There are many formulations of the genome assembly problem. One of the first asks to reconstruct the genome as a shortest superstring of the reads [30,16,15]. Later formulations referred to a graph built from the reads, such as a de Bruijn graph [10,32] or a string graph [26,38]. In an (edge-centric) de Bruijn graph, the reconstructed genome can be modeled as a circular walk covering every edge exactly once—Eulerian [32]—or at least once—a Chinese Postman tour [21,22,27,13]. In a string graph, the reconstructed genome can be modeled as a circular walk covering every node exactly once—Hamiltonian—[11,29], or at least once [27]. These models have also been considered in their weighted versions [21,27,29], or augmented to include other information, such as mate-pairs [33,23,14]. Each such notion of genomic reconstruction brought along questions concerning its validity. For example, under which conditions on the sequencing data (e.g., coverage, read length, error rate) is there at least one reconstruction [19,25], or exactly one reconstruction [5,18,32]. If there are many possible reconstructions, then what is their number [17,8] and in which aspects one is different from all others [8]. In contrast to the framework of this paper, most of these formulations deal with finding a single genomic reconstruction as opposed to a set of safe strings (i.e. contigs).

There are a few notable exceptions. In [4], Boisvert and colleagues also define the assembly problem in terms of finding contigs, rather than a single reconstruction. Nagarajan and Pop [27] observe that Waterman's characterization [41] of the graphs with a unique Eulerian tour leads to a simple algorithm for finding all safe strings when a genomic reconstruction is an Eulerian tour. They also suggest an approach for finding all the safe strings when a genomic reconstruction is a Chinese Postman tour [27]. We note, however, that in the Eulerian model, the exact copy count of each edge should be known in advance, while in the Chinese Postman model (minimizing the length of the genomic reconstruction), the solution will over-collapse all tandem repeats. Furthermore, these approaches have not been implemented and hence their effectiveness is unknown.

In practice, the most commonly employed safe strings are the ones spelled by maximal *unitigs*, where *unitigs* are paths whose internal nodes have in- and out-degree one. Figure 2 shows an example of the output of the unitig algorithm, and also illustrates that it does not identify all safe strings. The EULER assembler [32] takes unitigs a step further and identifies strings spelled by paths whose internal nodes have out-degree equal to one (with no constraint on their in-degree). It can be shown that such strings are also safe. However, the most complete characterization of safe strings that we found is given by the *Y-to-V algorithm* [22,12,17]. Consider a node v with exactly one in-neighbor u and more than one out-neighbors $w_1, \dots, w_d$. The *Y-to-V reduction* applied to v removes v and its incident edges from the graph and adds nodes $v_1, \dots, v_d$ with edges from u to v_i and from v_i to w_i, for all $1 \leqslant i \leqslant d$. The Y-to-V reduction is defined symmetrically for nodes with out-degree exactly one and in-degree greater than one. Figure 1 illustrates the definition. The

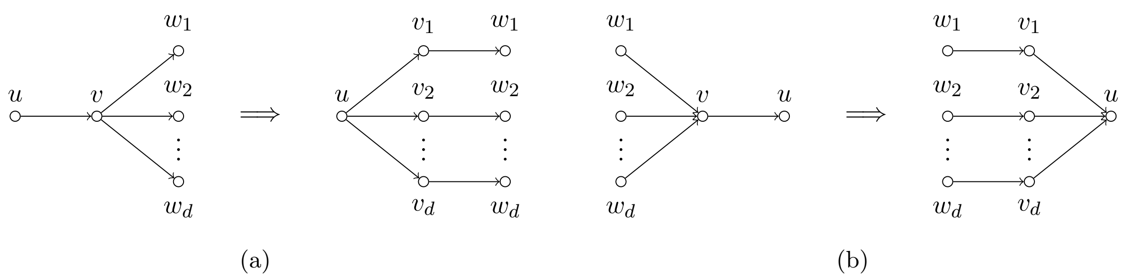


Fig. 1. The Y-to-V reduction applied to node v. In Fig. 1(b) v has in-degree exactly one; in Fig. 1(b) v has out-degree exactly one.

Y-to-V algorithm proceeds by repeatedly applying Y-to-V reductions, in arbitrary order, for as long as possible. The algorithm then outputs the strings spelled by the maximal unitigs in the final graph (see Figure 2d for an example). The Y-to-V algorithm can also be shown to be safe, but, as we will show in Figure 2, it does not always output *all* the safe strings. We are not aware of any study that compares the merits of Y-to-V contigs to unitigs, and we therefore perform this analysis in Section 8.

3 Basic definitions

Given a string x and an index $1 \leqslant i \leqslant |x|$, we define $\text{pre}(x, i)$ and $\text{suf}(x, i)$ as its length i prefix and suffix, respectively. If x and y are two strings, and $\text{suf}(x, k) = \text{pre}(y, k)$ for some $k \leqslant |x| - 1$, then we define $x \oplus^k y$ as $x[1..|x| - k]$ concatenated with y. This captures the notion of merging two overlapping strings. A k-mer of x is a substring of length k. Let R be a set of strings, which we equivalently refer to as *reads*. The *node-centric de Bruijn graph built on* R, denoted $\text{DB}^k_{\text{nc}}(R)$, is the graph whose set of nodes is the set of all k-mers of R, in which there is an edge from a node x to a node y iff $\text{suf}(x, k-1) = \text{pre}(y, k-1)$ [7]. The *edge-centric de Bruijn graph built on* R, denoted $\text{DB}^k_{\text{ec}}(R)$ is defined similarly to $\text{DB}^k_{\text{nc}}(R)$, with the difference that there is an edge from x to y iff $\text{suf}(x, k-1) = \text{pre}(y, k-1)$ *and* $x \oplus^{k-1} y$ is a substring of some string in R [10]. The *weight* of the edges of $\text{DB}^k_{\text{nc}}(R)$ and $\text{DB}^k_{\text{ec}}(R)$ is $k - 1$.

Let G be a graph, possibly with parallel edges and self-loops. The number of nodes and edges in a graph are denoted by n and m, respectively. We use $N^-(v)$ to denote the set of in-neighbors and $N^+(v)$ to denote the set of out-neighbors of a node v. A *walk* w is a sequence $(v_0, e_0, v_1, e_1, \ldots, v_t, e_t, v_{t+1})$ where $v_0, \ldots, v_{t+1}$ are nodes, and each e_i is an edge from v_i to v_{i+1}, and $t \geqslant -1$. Its *length* is its number of edges, namely $t + 1$. A *path* is a walk where the nodes are all distinct, except possibly the first and last nodes may be the same, in which case it will also be called a *cycle*. Walks and paths of length at least one are called *proper*. A walk whose first and last nodes coincide is called *circular* walk. A path (walk) with first node u and last node v will be called a *path (walk) from* u *to* v, and denoted as u-v *path (walk)*. A walk is called *node-covering* if it passes through each node of G, and *edge-covering* if it passes through each edge of G. The notions of *prefix* and *subwalk* are defined for walks in the natural way, e.g., by interpreting a walk to be a string made up by concatenating its edges. In particular, we say that a walk w_1 is a subwalk of a *circular* walk w_2 if w_1 interpreted as string is a substring of w_2 interpreted as *circular* string. In this paper we allow strings and walks to have overlapping extremities when viewed as substrings of a circular string, i.e., when aligned to a circular string (see e.g. the two omnitigs from Figure 2(f) which have an overlapping tail and head).

Let ℓ be a function labeling the nodes of G and let c be a function giving weights to the edges (intuitively, c should represent the length of overlaps). One can apply the notion of string spelled by a walk $w = (v_0, e_0, v_1, e_1, \ldots, v_t, e_t, v_{t+1})$ by defining the string *spelled by* w as $\text{spell}(w) = \ell(v_0) \oplus^{c(e_0)} \ell(v_1) \oplus^{c(e_1)} \cdots \oplus^{c(e_t)} \ell(v_{t+1})$. When the walk w is circular (thus $v_{t+1} = v_0$), then $\text{spell}(w)$ will be interpreted as the circular string obtained by overlapping the strings $\ell(v_0)$ and $\ell(v_{t+1})$.

4 Problem formulation

There are various theoretical approaches to formulating the assembly problem. Here, we adopt a model that captures the most popular ones: the node-centric de Bruijn graph, the edge-centric de Bruijn graph, and the string graph [26]. We generalize these using the notion of a *genome graph*:

Definition 1 (Genome graph). *A graph G with edge-weights given by c and node-labels is a* genome graph *if and only if (1) for every edge $e = (u, v)$, $\text{suf}(u, c(e)) = \text{pre}(v, c(e))$, and (2) for any two walks w_1 and w_2, w_1 is a subwalk of w_2 if and only if $\text{spell}(w_1)$ is a substring of $\text{spell}(w_2)$.*

Both node- and edge-centric de Bruijn graphs are genome graphs, directly by their definition. Similarly, the interested reader can verify that string graphs, as commonly defined in [26,27,22,37], are genome graphs. Intuitively, the first condition states that the edge-weights represent the length of overlaps between strings, while the second condition prohibits a certain redundancy in the graph. It can be broken if, for example, there are nodes with duplicate labels, or if some labels are substrings of others. Or, for strings graphs, it can be broken if transitive edges are not removed from the graph [26]. We now augment a genome graph with a rule defining a "genomic reconstruction."

Definition 2 (Graph model). *A* graph model *$\mathcal{G}$ is defined by*

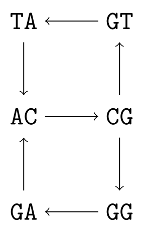
- *An algorithm that transforms a set of reads R into a genome graph, denoted by $\mathcal{G}(R)$.*
- *A rule that determines whether a walk in $\mathcal{G}(R)$ is a* genomic reconstruction*.*

Intuitively, a genomic reconstruction spells a genome that could have generated the observed set of reads R. In this paper, we consider two graph models. In the *edge-centric* model, a genomic reconstruction is a circular edge-covering walk; its underlying genome graph can be e.g. an edge-centric de Bruijn graph. In the *node-centric* model, a genomic reconstruction is a circular node-covering walk; its underlying genome graph can be a node-centric de Bruijn graph or a string graph. As mentioned in the introduction, we assume, without always explicitly stating it onwards, that $\mathcal{G}(R)$ contains at least one genomic reconstruction, and for technical reasons—see the proof of Lemma 1—that $\mathcal{G}(R)$ is always different from a single cycle. In fact, in this latter case the assembly problem is trivial.

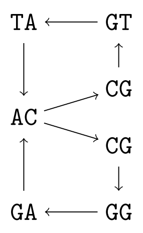
We now define the strings that belong to all genomic reconstructions.

Definition 3 (Safe string). *Given a set of reads R and a graph model $\mathcal{G}$, a string s is said to be a* safe string for $\mathcal{G}(R)$ *if for every genomic reconstruction w of $\mathcal{G}(R)$, s is a substring of $\text{spell}(w)$.*

In particular, for a node-centric (respectively, edge-centric) graph model $\mathcal{G}$, a string s is safe if for every circular node-covering (respectively, edge-covering) walk w, s is a substring of $\text{spell}(w)$. It also follows from the definitions (again assuming no gaps in coverage and no errors in the reads)



(a) Graph G



(b) Transformed graph G^T

AC → CG
CG → GT → TA → AC
CG → GG → GA → AC

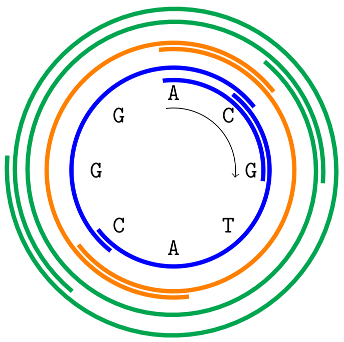
(c) Maximal unitigs of G

AC → CG → GT → TA → AC
AC → CG → GG → GA → AC

(d) Maximal unitigs of G^T

CG → GT → TA → AC → CG → GG → GA → AC → CG
CG → GG → GA → AC → CG → GT → TA → AC → CG

(e) Maximal omnitigs of G



(f) The original genome and its contigs

Fig. 2. The output of the three algorithms on the edge-centric de Bruijn graph G from (a), built from the circular string in (f). Each contig is drawn as an arc on the wheel in (f). (c) the maximal unitigs of G; (b) the Y-to-V reduction is applied to node CG and the resulting graph G^T is shown; no more reductions are applicable and G^T has two maximal unitigs, shown in (d); (e) the maximal omnitigs of G; in this particular example, they are also circular edge-covering walks of G, and one can be obtained from the other by a circular permutation. Note that this example illustrates that the Y-to-V algorithm does not always output all safe strings, because its output (d) does not contain the strings of (e).

that if the genome graph is $\mathrm{DB}^k_{\mathrm{nc}}(R)$ or $\mathrm{DB}^k_{\mathrm{ec}}(R)$, then a string is safe if it is a substring of every circular string with the same set of k-mers, or $(k+1)$-mers respectively, as R.

Solving the following problem gives all the information that can be safely retrieved from a graph model.

Definition 4 (The safe and complete contig assembly problem). *Given a set of reads R and a graph model $\mathcal{G}$, output* all *the safe strings for $\mathcal{G}(R)$.*

In this paper we solve this problem for the node- and edge-centric models defined above. In Sections 5 and 6 we first deal with the edge-centric model, and then in Section 7 we show how these results can be modified for the node-centric model.

As a technical aside, our algorithms will output only *maximal* safe strings, in the sense that they are not a substring of any other safe string. In fact, this is desirable in practice, and moreover, the set of all safe strings is the set of all substrings of the maximal ones.

A note on assumptions: Our model makes three implicit assumptions, as outlined at the end of the Introduction. Here, we observe that such assumptions are necessary to prove even the simplest desired property: that the unitig algorithm outputs only safe strings. Let $w = (v_0, e_0, v_1, e_1, v_2)$ be a unitig in an edge-centric de Bruijn graph G built from the $(k+1)$-mers of a genome S. If the genome is not circular (assumption (i)), then e.g. the last k-mer of S can be v_0, its first k-mer can be v_1, the string $v_0 \oplus^k v_1$ can appear inside S, but $v_0 \oplus^k v_1 \oplus^k v_2$ does not have to appear in S. If there are gaps in coverage (assumption (ii)), then both an in-neighbor v' and an out-neighbor v'' of v_1 may be missing from G making w look safe whereas in reality $v_0 \oplus^k v_1 \oplus^k v_2$ may not be a substring of S. If a read contains a sequencing error (assumption (iii)), then this creates a bubble in G with one of its paths being a unitig not spelling a substring of S.

5 Characterization of safe strings: omnitigs

In this section, we provide a characterization of walks that spell safe strings (see Figure 3 for an illustration). This characterization will be the basis of our omnitig algorithm in the next section.

Definition 5 (Omnitig, edge-centric model). *Let G be a directed graph and let $w = (v_0, e_0, v_1, e_1, \ldots, v_t, e_t, v_{t+1})$ be a walk in G. We say that w is a* omnitig *if and only if for all $1 \leqslant i \leqslant j \leqslant t$, there is no proper v_j-v_i path with first edge different from e_j, and last edge different from e_{i-1}.*

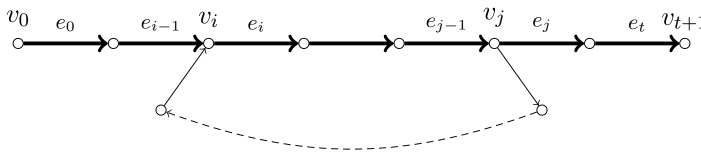


Fig. 3. An illustration of the omnitig definition, edge-centric model

The following theorem proves that the omnitigs spell all the safe strings, using the help of an intermediary characterization of omnitigs.

Theorem 1. *Given an edge-centric graph model $G = \mathcal{G}(R)$ built for a set of reads R, and a string s, the following three statements are equivalent:*

(1) *s is a safe string for G;*
(2) *s is spelled by a walk $w = (v_0, e_0, v_1, e_1, \ldots, v_t, e_t, v_{t+1})$ in G and w is an omnitig;*
(3) *s is spelled by a walk $w = (v_0, e_0, v_1, e_1, \ldots, v_t, e_t, v_{t+1})$ in G and for all $1 \leqslant j \leqslant t$ all proper v_j-v_j (circular) walks w' fulfill at least one of the following conditions:*
   (i) *the subwalk $(v_j, e_j, \ldots, v_t, e_t, v_{t+1})$ of w is a prefix w', or*
   (ii) *the subwalk $(v_0, e_0, \ldots, v_{j-1}, e_{j-1}, v_j)$ of w is a suffix of w', or*
   (iii) *w is a subwalk of w'.*

We prove Theorem 1 by proving the cyclical sequence of implications (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (1).

Proof of (1) $\Rightarrow$ (2). Assume that s is a safe string for G. By definition of a genome graph, s is spelled by a unique walk in G. Let $w = (v_0, e_0, v_1, e_1, \ldots, v_t, e_t, v_{t+1})$ be this walk, and let A be a circular edge-covering walk of G; thus A contains w as subwalk, and s is a sub-string of spell(A).

Assume for a contradiction that there exist $1 \leqslant i \leqslant j \leqslant t$, and a proper v_j-v_i path p with first edge different from e_j and last edge different from e_{i-1}. From A, we will construct another circular edge-covering walk B of G which does not contain w as subwalk, and hence, by the definition of a genome graph, also spell(B) does not contain s as sub-string. This will contradict the safeness of s. Whenever A visits node v_j, then B follows the v_j-v_i path p, then follows $(v_i, e_i, \ldots, e_{j-1}, v_j)$, and finally continues as A. To see that w does not appear as a subwalk of B, consider the subwalk $w' = (v_{i-1}, e_{i-1}, v_i, e_i, \ldots, e_{j-1}, v_j, e_j, v_{j+1})$ of w (recall that $1 \leqslant i \leqslant j \leqslant t$). Since p is proper, and its first edge is different from e_j and its last edge is different from e_{i-1}, then, by construction, the only way that w' can appear in B is as a subwalk of p. However, this implies that both v_j and v_i appear twice on p, contradicting the fact that p is a path. □

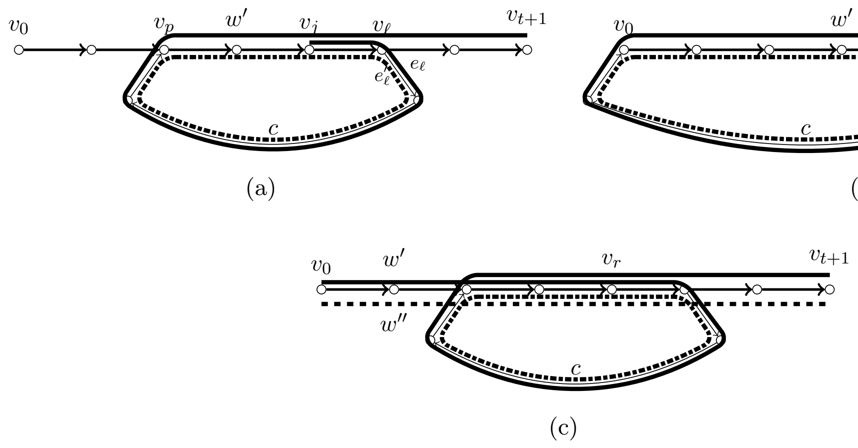


Fig. 4. Illustration of three cases in the proof of the implication (2) ⇒ (3) of Theorem 1.

Proof of (2) ⇒ (3). Suppose that w is an omnitig, and assume for a contradiction that there exists a proper v_j-v_j walk (for some $1 \leqslant j \leqslant t$) not satisfying (i)–(iii). Let w' be the shortest such walk. Since w' does not have $(v_j, e_j, \ldots, v_t, e_t, v_{t+1})$ as prefix, then there exists a first node v_ℓ on w, $j \leqslant \ell \leqslant t$, such that from v_ℓ, w' continues with an edge $e'_\ell \neq e_\ell$. Symmetrically, since w' does not have $(v_0, e_0, \ldots, v_{j-1}, e_{j-1}, v_j)$ as suffix, let v_i be the last node of w, $1 \leqslant i \leqslant j$ such that before entering v_i, the walk w' uses an edge $e'_{i-1} \neq e_{i-1}$. Let w'_0 denote the subwalk of w' between e'_ℓ and e'_{i-1} (inclusive). If w'_0 is a path, then w'' is a proper v_ℓ-v_i path, $1 \leqslant i \leqslant \ell \leqslant t$, whose first edge e'_ℓ is different from e_ℓ, and its last edge e'_{i-1} is different from e_{i-1}, which contradicts the fact that w is an omnitig. We now prove that w' is in fact a path.

Suppose for a contradiction that it is not, thus that it contains a cycle c, with $c \neq w'$. Let w'' be the walk obtained from w' by removing the cycle c. Observe that w'' is still a proper v_j-v_j walk. We show that w'' still does not satisfy (i)–(iii), which will contradict the minimality of w'. Assume for a contradiction that w'' satisfies at least one of (i), (ii), or (iii).

First, if w'' satisfies (i), this implies that the edge e'_ℓ out-going from v_ℓ belongs to c, and after traversing c, the walk w' continues through $(v_\ell, e_\ell, \ldots, v_t, e_t, v_{t+1})$ (see Figures 4(a) and 4(b)). Let v_p be the node of w with greatest index $p \in \{0, \ldots, \ell\}$ that c visits with an edge $e' = (v, v_p)$ not on w. Such a node exists because c is a cycle and it must return to v_ℓ. If $p \geqslant 1$ (see Figure 4(a)), then c does not satisfy (i)–(iii). Since c is proper and passes through v_ℓ, where $1 \leqslant \ell \leqslant t$, this contradicts the minimality of w'. Therefore, $p = 0$ (see Figure 4(b)), and thus, the initial v_j-v_j walk w' (containing c as subwalk) visits $(v_0, e_0, \ldots, v_\ell)$, and then continues through $(v_\ell, e_\ell, \ldots, v_t, e_t, v_{t+1})$. This implies that w' contains w as subwalk, which contradicts the choice of w'.

The second case when w'' satisfies (ii) is entirely symmetric.

Third, assume that w'' contains w as subwalk. Since w is not a subwalk of w', this implies that c is a proper v_r-v_r walk, for some $1 \leqslant r \leqslant t$, not satisfying (i)–(iii), which again contradicts the minimality of w' (see Figure 4(c)). This completes the proof of (2) ⇒ (3) □

Proof of (3) ⇒ (1). Assume w satisfies (3), and let A be a circular edge-covering walk of G. We need to show that w is a subwalk of A. Let $w_j = (v_0, e_0, \ldots, v_{j-1}, e_{j-1}, v_j)$ be the longest prefix of w that A ever traverses, ending at some v_j. Since A covers all edges, then it also covers e_0, and thus $j \geqslant 1$. Suppose for a contradiction that $j \neq t + 1$.

Algorithm 1: Omnitig algorithm to find all safe strings of a graph G.

```
1  extend($w$)
2      Denote $w = (v_0, e_0, v_1, e_1, \ldots, v_{t-1}, e_{t-1}, v_t)$;
3      foreach edge $e = (v_t, y)$ out-going from $v_t$ do
4          $X := (N^-(v_1) \setminus \{v_0\}) \cup \cdots \cup (N^-(v_t) \setminus \{v_{t-1}\})$;
5          let $G'$ equal $G$ minus the edge $e$;
6          if there is no path in $G'$ from $v_t$ to a node of $X$ then
7              extend($(v_0, e_0, v_1, e_1, \ldots, v_{t-1}, e_{t-1}, v_t, e, y)$);
8      if $w$ was never extended then
9          $W := W \cup \{w\}$;
10 $W := \emptyset$;
11 foreach edge $e = (u, v)$ of $G$ do
12     extend($(u, e, v)$);
13 remove from $W$ any walk that is a subwalk of another walk in $W$;
14 return $\{\text{spell}(w) \ : \ w \in W\}$;
```

Since A is circular and covers all edges of G, then after traversing w_j, the walk A eventually visits the edge e_j. The walk A may visit v_j multiple times before traversing the edge e_j. Let w' denote the subwalk of A between the last two occurrences of v_j before A traverses the edge e_j. Since w' is a proper v_j-v_j walk, $1 \leqslant j \leqslant t$, and w satisfies (3), we have that one of the following must hold:

- the walk $(v_j, e_j, \ldots, v_t, e_t, v_{t+1})$ is a prefix of w': this contradicts the fact that w' is a subwalk of A between v_j and the immediately next occurrence of e_j (since in this case w' contains e_j);
- the walk $(v_0, e_0, \ldots, v_{j-1}, e_{j-1}, v_j)$ is a suffix of w': this implies that (w', e_j, v_{j+1}) is a longer prefix of w which is a subwalk of A, contradicting the maximality of w_j;
- the walk w appears on w': since w' is a subwalk of A, this implies that also w is a subwalk of A, contradicting again the maximality of w_j.

□

6 Omnitig algorithm

In this section, we use Theorem 1 to give the omnitig algorithm (Algorithm 1) and prove that it runs in polynomial time (Theorem 2). The algorithm finds all maximal omnitigs of $\mathcal{G}(R)$, which, by Theorem 1, are exactly the maximal safe strings of $\mathcal{G}(R)$. Our algorithm is based on the following observation, which follows directly from the definition of omnitigs:

Observation 1. *Consider a walk $w' = (v_0, e_0, \ldots, e_{t-1}, v_t, e_t, v_{t+1})$ of length at least two, and consider its subwalk $w = (v_0, e_0, \ldots, e_{t-1}, v_t)$. Then w' is an omnitig if and only if (i) w is an omnitig and (ii) for all $0 \leqslant i \leqslant t-1$, there is no proper v_t-v_i path with first edge different from e_t and last edge different from e_{i-1}.*

The idea of the algorithm is to start an exhaustive traversal of G from every edge (Lines 11-12), which by definition is an omnitig, and to keep traversing edges as long as the current walk is an omnitig. An omnitig w is thus recursively constructed, by possibly extending to the right with each edge e out-going from its last vertex (Lines 3-7). If w extended with e is not an omnitig, then we abandon this extension because Observation 1 tells us that no further extension could be an omnitig. To check if this extension is an omnitig or not, it is enough to check whether condition (ii) of Observation 1 is satisfied. Condition (i) is automatically satisfied because of the structure of the algorithm–we extend only walks that are omnitigs. The omnitigs found are saved in a set W (Line 9), except for those omnitigs that are obviously non-maximal (Line 8). In the final step (Lines 13-14), we remove the non-maximal omnitigs from W and report the rest.

To check that condition (ii) is satisfied (Lines 4-6), we take the set X (Line 4) and check if there is a path starting with an edge out-going from v_t and different from e, and leading to a node of X. The correctness of this procedure can be seen as follows. If there is no such path, then we know that there is no path satisfying (ii). If we do find a path p from v_t to some in-neighbor $x \in X$ of some v_i, and p does not use v_i, then the path obtained by extending p to v_i contradicts (ii). If p contains v_i, then such an extension is not possible, because a path cannot repeat a vertex; however, we will show that p cannot use v_i by contradiction. Assume that it does, and observe that after passing through v_i, the path p cannot pass again through v_t. Let v_j, $i \leqslant j < t$, be the first vertex that p visits after v_i such that from v_j it continues with an edge $e' \neq e_j$. Let p' denote the v_j-x subpath of p from v_j until x. We obtained that p' followed by v_i is a proper v_j-v_i path with first edge different from e_j, last edge different from e_{i-1}, and $1 \leqslant i \leqslant j \leqslant t$. This contradicts the fact that w (the walk we are extending) is an omnitig.

Next, we show that the algorithm runs in polynomial time. First, we show that the number of omnitigs included in W and their length, prior to removal of non-maximal ones, is polynomial:

Lemma 1. *Let W be a set of omnitigs in an edge-centric graph model $\mathcal{G}(R)$, whose genome graph is different than a single cycle. Furthermore, suppose no omnitig in W is a prefix of another omnitig in W. Then, $|W| \leqslant nm$ and the length of any omnitig in W is $O(nm)$.*

Proof. We first show that we can visit the edges of $G = \mathcal{G}(R)$ with a circular edge-covering walk C of at most nm nodes. Let $e_0, \ldots, e_{m-1}$ be an arbitrary order of the edges of G. Since we assume that G admits one genomic reconstruction, then G is strongly connected. Thus, from every end extremity of e_i there is a path to the start extremity of $e_{(i+1) \bmod m}$, $0 \leqslant i \leqslant m-1$, of length at most $n-1$. Therefore, C can be constructed to first visit e_0, then to follow such a path until e_1, and so on until e_{m-1}, from where it follows such a path back to e_0.

By Theorem 1, we have that any omnitig of G is a subwalk of C. We can associate every $w \in W$ with all the start positions in C (in terms of nodes) where it is a subwalk. Because W does not contain walks that are prefixes of other walks, a position of C can have at most one walk associated with it. Since $|C| \leqslant nm$, W can contain at most nm walks.

It remains to prove that the length of any omnitig in W is $O(nm)$. To simplify notation, rename C as $(v_0, e_0, v_1, e_1, \ldots, v_t, e_t, v_{t+1})$ with $v_{t+1} = v_0$. Since G is different than a single cycle, then there exist v_j and v_i on C, such that $e = (v_j, v_i)$ is an edge of G, and $e \notin \{e_j, e_{i-1}\}$. Any omnitig (thus a subwalk of C) cannot contain twice v_j and v_i as internal nodes, since otherwise the proper path (v_j, e, v_i) violates the omnitig definition. Thus the length of any omnitig is $O(nm)$. □

(As an aside, it remains open whether the bound on $|W|$ can be reduced to m, which is the case for unitigs; our experiments from Section 8 suggest this may be the case in practice.) Note that Line 8 guarantees that W, prior to removal of subwalks in Line 13, satisfies the prefix condition of Lemma 1. Lemma 1 then implies that reporting one omnitig by our algorithm takes polynomial time, and there are only polynomially many omnitigs reported. Furthermore, removing the non-maximal omnitigs (Line 13) can be done in linear time in the sum of the omnitig lengths, by appropriately traversing a suffix tree constructed from them. Thus, we have our main theorem:

Theorem 2. *Let R be a set of reads and let $\mathcal{G}(R)$ be an edge-centric graph model. Algorithm 1 outputs in polynomial time all safe strings of $\mathcal{G}(R)$.*

Finally, we note some implementation details that are crucial in practice. Prior to starting, we apply the Y-to-V algorithm and the standard graph compaction algorithm to compact unitigs [6]. This significantly reduces the number of nodes/edges in the graph without changing the maximal safe strings. We also precompute all omnitigs of length two and store them in a hash table, so that whenever we want to extend the omnitig w in Line 6, we check beforehand whether the pair (e_{t-1}, e) is stored in the hash table. This significantly limits, in practice, the number of graph traversals we have to do at Line 6. Finally, we do not compute the set X every time, but instead incrementally built it up as we extend the omnitig w. Our implementation is freely available for use[1].

7 Node-centric model

In this section we obtain analogous results for node-centric models, though both the definitions, algorithms, and proofs need to modified. The following definition is similar to the one for the edge-centric model, the only addition being its second bullet (see Figure 5 for an illustration).

Definition 6 (Omnitig, node-centric model). *Let G be a directed graph and let $w = (v_0, e_0, v_1, e_1, \dots, v_t, e_t, v_{t+1})$ be a walk in G. We say that w is a* omnitig *iff the following two conditions hold:*

- *for all $1 \leqslant i \leqslant j \leqslant t$, there is no proper v_j-v_i path with first arc different from e_j, and last arc different from e_{i-1}.*
- *for all $0 \leqslant j \leqslant t$, the arc e_j is the only v_j-v_{j+1} path.*

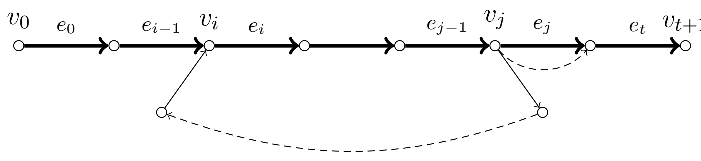


Fig. 5. An illustration of the omnitig definition, node-centric model

The following theorem is analogous to Theorem 1, and characterizes the safe strings in the node-centric model.

Theorem 3. *Given a node-centric graph model $G = \mathcal{G}(R)$ built for a set of reads R, and a string s, the following three statements are equivalent:*

(1) *s is a safe string for G;*

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

(2) *s is spelled by a walk $w = (v_0, e_0, v_1, e_1 \dots, v_t, e_t, v_{t+1})$ in G and w is a omnitig;*

(3) *s is spelled by a walk $w = (v_0, e_0, v_1, e_1 \dots, v_t, e_t, v_{t+1})$ in G and w satisfies: for all $0 \leqslant j \leqslant t$, the arc e_j of w is the only v_j-v_{j+1} path, and for all $1 \leqslant j \leqslant t$, all proper v_j-v_j (circular) walks w' fulfill at least one of the following conditions:*
   - (i) *the subwalk $(v_j, e_j, \dots, v_t, e_t, v_{t+1})$ of w is a prefix w', or*
   - (ii) *the subwalk $(v_0, e_0, \dots, v_{j-1}, e_{j-1}, v_j)$ of w is a suffix of w', or*
   - (iii) *w is a subwalk of w'.*

We analogously prove Theorem 3 by proving the cyclical sequence of implications (1) ⇒ (2) ⇒ (3) ⇒ (1).

Proof of (1) ⇒ (2). Assume that s is a safe string for R. By definition of a graph model, a safe string for R is spelled by a unique walk in a node-centric model for R. Let this walk be w, and let A be a circular node-covering walk of G (thus containing w as subwalk).

First, assume for a contradiction that there exist $1 \leqslant i \leqslant j \leqslant t$, and a proper v_j-v_i path p whose first arc is different from e_j and its last arc is different from e_{i-1}. From A, we can construct another circular node-covering walk B of G which does not contain w as subwalk, and thus spell(B) does not contain s as sub-string. This will contradict the fact that s is a safe string for G.

Whenever A visits node v_j, then B follows the v_j-v_i path p, then it follows $(v_i, e_i, \dots, e_{j-1}, v_j)$, and finally continues as A. To see that w does not appear as a subwalk of B, consider the subwalk $w' = (v_{i-1}, e_{i-1}, v_i, e_i, \dots, e_{j-1}, v_j, e_j, v_{j+1})$ of w (recall that $1 \leqslant i \leqslant j \leqslant t$). Since p is proper, and its first arc is different from e_j and its last arc is different from e_{i-1}, then, by construction, the only way that w' can appear in B is as a subwalk of p. However, this implies that both v_j and v_i appear twice on p, contradicting the fact that p is a v_j-v_i path.

Second, assume for a contradiction that there is some $0 \leqslant j \leqslant t$ and another v_j-v_{j+1} path p' than the arc e_j. Just as above, from A we can construct another node-covering walk C that avoids w (and thus spell(C) does not contain s as sub-string) as follows. Whenever A traverses the arc e_j, C traverses instead p'. The walk C is node-covering because it still covers all nodes of w, and otherwise C coincides with A. However, it does not contain w as subwalk because p' is different from the arc e_j, and, since it is a path, it cannot pass through e_j again, as otherwise it would visit twice either v_j, v_{j+1}, or both. □

The proof of (2) ⇒ (3) is identical to the corresponding proof of (2) ⇒ (3) for Theorem 1.

Proof of (3) ⇒ (1). Assume w satisfies (3), and let A be a circular node-covering walk of G. We need to show that w is a subwalk of A. Let $w_j = (v_0, e_0, \dots, v_{j-1}, e_{j-1}, v_j)$ be the longest prefix of w that A ever traverses, ending at some v_j. Since A covers all nodes, then $j \geqslant 0$. Suppose for a contradiction that $j \neq t + 1$. Since A is circular and covers all nodes of G, then after traversing w_j, the walk A eventually visits the node v_{j+1}. The walk A may, or may not, visit v_j again before visiting the node v_{j+1}.

First, suppose that after visiting v_j at the end of w_j, A visits again v_j before visiting v_{j+1}. Let w' denote the subwalk of A between the last two occurrences of v_j before visiting the node v_{j+1}. If $1 \leqslant j \leqslant t$, since w' is a v_j-v_j walk, and w satisfies (3), we have that either:

- the walk $(v_j, e_j, v_{j+1}, \dots, v_t, e_t, v_{t+1})$ is a prefix of w': this contradicts the fact that w' is a subwalk of A between v_j and the immediately next occurrence of v_{j+1}, since in this case w' would contain v_{j+1} more times;

- the walk $(v_0, e_0, \ldots, v_{j-1}, e_{j-1}, v_j)$ is a suffix of w': this implies that (w', e_j, v_{j+1}) is a longer prefix of w that is a subwalk of A, contradicting the maximality of w_j;
- the walk w appears on w': since w' is a subwalk of A, this implies that also w is a subwalk of A, contradicting again the maximality of w_j.

If $j = 0$, then by removing all cycles from w', we obtain a v_j-v_{j+1} path, different than the arc e_0, since otherwise we would contradict the maximality of w_j. But this contradicts the fact that w is satisfies (3).

Second, suppose that the walk A does not visit v_j again after w_j and before visiting v_{j+1}. Let w'' be the v_j-v_{j+1} subwalk of A between w_j and this next occurrence of v_{j+1}. The walk w'' may not be a path, but by removing all cycles from it we obtain a v_j-v_{j+1} path w'''. This path is different from e_j by the maximality of w_j, contradicting again the fact that w satisfies (3). □

Analogous to the edge-centric case, we can prove the following polynomial upper-bound on the number and length of all omnitigs.

Lemma 2. *Let W be a set of omnitigs in a node-centric graph model $\mathcal{G}(R)$, whose genome graph is different than a single cycle. Furthermore, suppose no omnitig in W is a prefix of another omnitig in W. Then, $|W| \leqslant n^2$ and the length of any omnitig in W is $O(n^2)$.*

We also leave open the question whether the bound on the number of maximal omnitigs in the node-centric model can be reduced to n. We now combine Theorem 3 and Lemma 2 for obtaining our polynomial-time safe and complete assembly algorithm.

Theorem 4. *There is a safe and complete assembly algorithm for any node-centric graph model $\mathcal{G}(R)$ built on a set R of reads, which runs in polynomial time.*

Proof. The proof is identical to the one for the edge-centric case (Algorithm 1 and Theorem 2). The only difference that needs to be made to Algorithm 1 is to check that the second bullet in the definition of omnitig for the node-centric case holds. This can be similarly performed by a single graph traversal, and only for the last edge added to the omnitig. □

8 Experimental results

We wanted to test the potential of omnitigs as an alternative to unitigs, under the assumptions of Section 4. We chose two genomes: one bacterial genome, *E.coli*, and one larger genome, Human chr10 (circularized). The graph model was the edge-centric de Bruijn graph built on the set of all $(k+1)$-mers of the genome. We used $k = 31$ and $k = 55$ for *E.coli* and chr10, respectively, according to what has been used in practice for the assembly of such genomes.

We wanted to measure the effect of omnitigs on assembly contiguity in terms of (1) increase in contig length, and (2) increase of biological context for elements of interest. To measure the increase in length, we measured the average contig length and the E-size. Since multiple contigs can cover overlapping regions, we found the E-size metric [36] to be more appropriate than the N50 metric. The E-size of a set of substrings of a genome is defined as the average, over all genomic positions i, of the mean length of all substrings spanning position i. This was computed by aligning the contigs to the reference. Table 1 shows that omnitigs exhibit significantly more contiguity than unitigs, with an average contig length that is 62-82% higher. There is very little improvement in the

Table 1. Results for $\mathrm{DB}^k_{\mathrm{ec}}(R)$, where R is the set of all $(k+1)$-mers of the genome.

	E.coli ($k = 31$)				chr10 ($k = 55$)			
	# strings	avg len	E-size	time (s)	# strings	avg len	E-size	time (s)
unitigs	1,743	2,654	33,309	< 1	259,845	546	8,344	1
Y-to-V	1,004	4,682	33,632	< 1	159,101	878	8,376	2
omnitigs	983	4,832	34,557	< 1	158,236	887	8,401	1,046

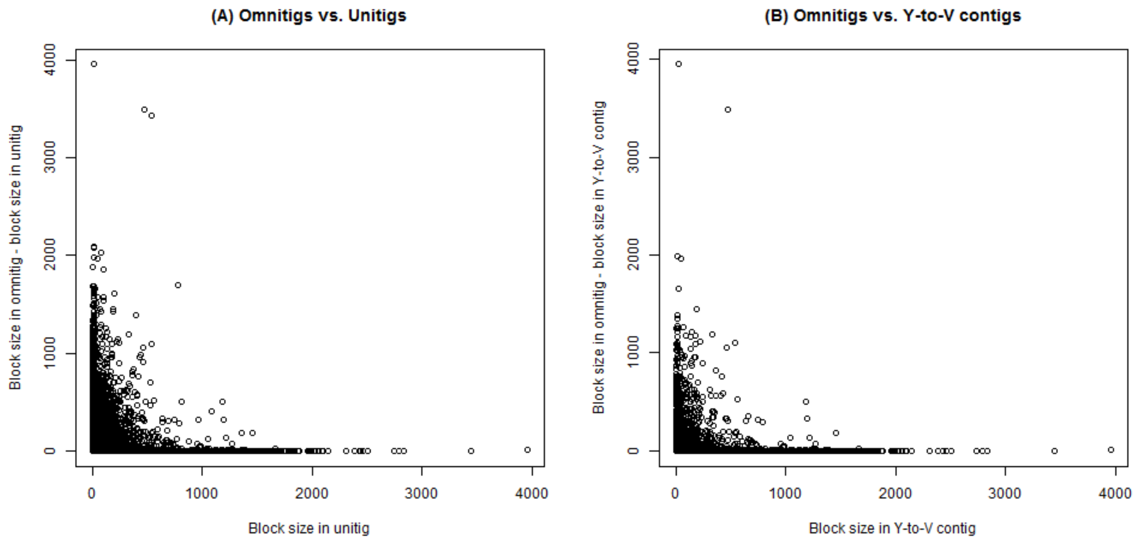


Fig. 6. The increase in SNP block size in omnitigs compared to unitigs (A) and Y-to-V contigs (B). Each point is a SNP, and the x-value is the block size of the unitig (in A) or Y-to-V contig (in B) covering it. The y-value is the increase in the block size, when compared with omnitigs. Note that the y-axis does not represent the block size, but a difference of block sizes.

E-size (1-4%), indicating that most of the improvement come from increasing the length of shorter contigs.

We wanted to also measure the potential of omnitigs to improve downstream biological analysis, relative to unitigs. Longer contigs can provide more flanking context around important genomic elements such as SNPs. One general type of study collects statistics about the relationship of each SNP to other SNPs on the same contig; such a study is necessarily limited by the number of SNPs present on the same contig [39]. We call this number the *block size* of a SNP. To see the effect of omnitigs on such a study, we identified chr10 locations of SNPs in the human population (using dbSNP), and the block size of each SNP in the omnitig vs. the unitig algorithms. Figure 6A shows that omnitigs in many cases provide more SNP context. The number of SNPs whose block size increased was ~ 1.7 million (out of ~ 5.9 million) and whose block size increased by more than 10 was ~ 137 thousand. The average number of SNPs per omnitig was 41, with only 26 per unitig. Consistent with the contiguity results of Table 1, the effect is more pronounced on contigs with less SNPs on them.

We also compared omnitigs to Y-to-V contigs. Y-to-V contigs have been proposed in the literature [22,12,17], but, to the best of our knowledge, there has not been a quantitative study comparing their merits against other contig algorithms. Omnitigs also provide more SNP context than Y-to-V contigs, with ~ 266 thousand SNPs having an increase in block size (Figure 6B). Omnitigs are only marginally better than Y-to-V contigs in terms of average contiguity (Table 1). Our results suggest that, though not as beneficial as omnitigs, Y-to-V contigs may nevertheless provide a better alternative to unitigs that is faster than the omnitig algorithm.

Table 1 also shows the wall-clock running times of our algorithms. The experiments were run on a node with two Xeon 2.53 GHz CPUs. We parallelized the omnitig algorithm so that it utilized all 8 available cores. We observe negligible running times for all algorithms on *E.coli*. On chr10, the running time of the omnitig algorithm is significantly longer (by 18mins) than the unitig or Y-to-V algorithm, though it would still not form a bottleneck in an assembly pipeline. The memory usage did not exceed 1 GB at any point, though we believe it can be significantly reduced with a more careful implementation.

9 Conclusion

There are two natural directions for future work: practical and theoretical. In the practical direction, the omnitig algorithm should be extended to handle the complexities of real data such as sequencing errors, imperfect coverage, linear genomes, and double-strandedness. This is a non-trivial task which is outside the scope of the current study, but it will be important in facilitating the application to genome analysis and assembly. In the theoretical direction, we believe that omnitigs exhibit more structure that can be exploited in a faster algorithm for finding all maximal omnitigs. We are also currently studying the graph model which a genomic reconstruction is any collection of circular walks that together cover all nodes/edges of the graph (as in metagenomic sequencing of bacteria). We are also studying the class of genome graphs admitting a single safe walk covering all of their nodes or edges, question related to the ones about unique reconstructions.

10 Acknowledgments

We would like to thank Daniel Lokshtanov for initial discussions, Rayan Chikhi for feedback on the manuscript, and Nidia Obscura Acosta for very helpful discussion on the manuscript. This work was supported in part by NSF awards DBI-1356529, IIS-1453527, and IIS-1421908 to PM, and by Academy of Finland grant 274977 to AT.

11 Author Disclosure Statement

No competing financial interests exist.

References

1. Bankevich, A., Nurk, S., Antipov, D., Gurevich, A.A., Dvorkin, M., Kulikov, A.S., Lesin, V.M., Nikolenko, S.I., Pham, S.K., Prjibelski, A.D., Pyshkin, A., Sirotkin, A., Vyahhi, N., Tesler, G., Alekseyev, M.A., Pevzner, P.A.: SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. Journal of Computational Biology **19**(5) (2012) 455–477

2. Boetzer, M., Henkel, C.V., Jansen, H.J., Butler, D., Pirovano, W.: Scaffolding pre-assembled contigs using SSPACE. Bioinformatics **27**(4) (2011) 578–579
3. Boetzer, M., Pirovano, W.: Toward almost closed genomes with gapfiller. Genome Biology **13**(6) (2012) 1–9
4. Boisvert, S., Laviolette, F., Corbeil, J.: Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. Journal of Computational Biology **17**(11) (2010) 1519–1533
5. Bresler, G., Bresler, M., Tse, D.: Optimal Assembly for High Throughput Shotgun Sequencing. BMC Bioinformatics **14**(Suppl 5) (2013) S18
6. Chikhi, R., Limasset, A., Jackman, S., Simpson, J.T., Medvedev, P.: On the representation of de bruijn graphs. In: Research in Computational Molecular Biology, Springer (2014) 35–55
7. Chikhi, R., Rizk, G.: Space-efficient and exact de Bruijn graph representation based on a Bloom filter. In: WABI. Volume 7534 of Lecture Notes in Computer Science, Springer (2012) 236–248
8. Guénoche, A.: Can we recover a sequence, just knowing all its subsequences of given length? Computer Applications in the Biosciences **8**(6) (1992) 569–574
9. Haussler, D., O'Brien, S.J., Ryder, O.A., Barker, F.K., Clamp, M., Crawford, A.J., Hanner, R., Hanotte, O., Johnson, W.E., McGuire, J.A., et al.: Genome 10K: a proposal to obtain whole-genome sequence for 10,000 vertebrate species. Journal of Heredity **100**(6) (2008) 659–674
10. Idury, R.M., Waterman, M.S.: A new algorithm for DNA sequence assembly. Journal of computational biology **2**(2) (1995) 291–306
11. Iu, Florent'ev, V.L., Khorlin, A.A., Khrapko, K.R., Shik, V.V.: Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method. Doklady Akademii nauk SSSR **303**(6) (1988) 1508–1511
12. Jackson, B.G.: Parallel methods for short read assembly. PhD thesis, Iowa State University (2009)
13. Kapun, E., Tsarev, F.: De Bruijn superwalk with multiplicities problem is NP-hard. BMC bioinformatics **14**(Suppl 5) (2013) S7
14. Kapun, E., Tsarev, F.: On NP-hardness of the paired de Bruijn sound cycle problem. In: Algorithms in Bioinformatics. Springer (2013) 59–69
15. Kececioglu, J.D., Myers, E.W.: Combinatiorial algorithms for DNA sequence assembly. Algorithmica **13**(1/2) (1995) 7–51
16. Kececioglu, J.D.: Exact and approximation algorithms for DNA sequence reconstruction. PhD thesis, University of Arizona, Tucson, AZ, USA (1992)
17. Kingsford, C., Schatz, M.C., Pop, M.: Assembly complexity of prokaryotic genomes using short reads. BMC bioinformatics **11**(1) (2010) 21
18. Lam, K., Khalak, A., Tse, D.: Near-optimal assembly for shotgun sequencing with noisy reads. BMC Bioinformatics **15**(S-9) (2014) S4
19. Lander, E.S., Waterman, M.S.: Genomic mapping by fingerprinting random clones: a mathematical analysis. Genomics **2**(3) (1988) 231–239
20. Luo, R., Liu, B., Xie, Y., Li, Z., Huang, W., Yuan, J., He, G., Chen, Y., Pan, Q., Liu, Y., et al.: Soapdenovo2: an empirically improved memory-efficient short-read de novo assembler. GigaScience **1**(1) (2012) 18
21. Medvedev, P., Brudno, M.: Maximum likelihood genome assembly. Journal of computational biology **16**(8) (2009) 1101–1116
22. Medvedev, P., Georgiou, K., Myers, G., Brudno, M.: Computability of models for sequence assembly. In: WABI. (2007) 289–301
23. Medvedev, P., Pham, S., Chaisson, M., Tesler, G., Pevzner, P.: Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. Journal of Computational Biology **18**(11) (2011) 1625–1634
24. Miller, J.R., Koren, S., Sutton, G.: Assembly algorithms for next-generation sequencing data. Genomics **95**(6) (2010) 315–327
25. Motahari, A.S., Bresler, G., Tse, D.N.C.: Information theory of DNA shotgun sequencing. IEEE Transactions on Information Theory **59**(10) (2013) 6273–6289
26. Myers, E.W.: The fragment assembly string graph. In: ECCB/JBI. (2005) 85
27. Nagarajan, N., Pop, M.: Parametric complexity of sequence assembly: Theory and applications to next generation sequencing. Journal of Computational Biology **16**(7) (2009) 897–908
28. Nagarajan, N., Pop, M.: Sequence assembly demystified. Nature Reviews Genetics **14**(3) (2013) 157–167
29. Narzisi, G., Mishra, B., Schatz, M.C.: On algorithmic complexity of biomolecular sequence assembly problem. In: Algorithms for Computational Biology. Springer (2014) 183–195
30. Peltola, H., Söderlund, H., Tarhio, J., Ukkonen, E.: Algorithms for some string matching problems arising in molecular genetics. In: IFIP Congress. (1983) 59–64

31. Pevzner, P.A.: L-Tuple DNA sequencing: computer analysis. J Biomol Struct Dyn **7**(1) (Aug 1989) 63–73
32. Pevzner, P.A., Tang, H., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences **98**(17) (2001) 9748–9753
33. Rubinov, A.R., Gelfand, M.S.: Reconstruction of a string from substring precedence data. Journal of Computational Biology **2**(2) (1995) 371–381
34. Sahlin, K., Vezzi, F., Nystedt, B., Lundeberg, J., Arvestad, L.: BESST-efficient scaffolding of large fragmented assemblies. BMC bioinformatics **15**(1) (2014) 281
35. Salmela, L., Sahlin, K., Mäkinen, V., Tomescu, A.I.: Gap filling as exact path length problem. In: Research in Computational Molecular Biology, Springer (2015) 281–292
36. Salzberg, S.L., Phillippy, A.M., Zimin, A., Puiu, D., Magoc, T., Koren, S., Treangen, T.J., Schatz, M.C., Delcher, A.L., Roberts, M.: GAGE: a critical evaluation of genome assemblies and assembly algorithms. Genome Research (2011)
37. Simpson, J.T., Durbin, R.: Efficient construction of an assembly string graph using the FM-index. Bioinformatics **26**(12) (2010) i367–i373
38. Simpson, J.T., Durbin, R.: Efficient de novo assembly of large genomes using compressed data structures. Genome Research (2011)
39. Uricaru, R., Rizk, G., Lacroix, V., Quillery, E., Plantard, O., Chikhi, R., Lemaitre, C., Peterlongo, P.: Reference-free detection of isolated SNPs. Nucleic Acids Research **43**(2) (2015) e11
40. Vyahhi, N., Pyshkin, A., Pham, S., Pevzner, P.A.: From de Bruijn graphs to rectangle graphs for genome assembly. In: Algorithms in Bioinformatics. Springer (2012) 249–261
41. Waterman, M.S.: Introduction to computational biology: maps, sequences and genomes. CRC Press (1995)
42. Zerbino, D.R., Birney, E.: Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Research **18**(5) (2008) 821–829