Safely Filling Gaps with Partial Solutions Common to All Solutions

Leena Salmela and Alexandru I. Tomescu

Abstract—Gap filling has emerged as a natural sub-problem of many *de novo* genome assembly projects. The gap filling problem generally asks for an s-t path in an assembly graph whose length matches the gap length estimate. Several methods have addressed it, but only few have focused on strategies for dealing with multiple gap filling solutions and for guaranteeing reliable results. Such strategies include reporting only unique solutions, or exhaustively enumerating all filling solutions and heuristically creating their consensus. Our main contribution is a new method for reliable gap filling: filling gaps with those sub-paths common to all gap filling solutions. We call these partial solutions *safe*, following the framework of (Tomescu and Medvedev, RECOMB 2016). We give an efficient safe algorithm running in $O(dm)$ time and space, where d is the gap length estimate and m is the number of edges of the assembly graph. To show the benefits of this method, we implemented this algorithm for the problem of filling gaps in scaffolds. Our experimental results on bacterial and on conservative human assemblies show that, on average, our method can retrieve over 73 percent more safe and correct bases as compared to previous methods, with a similar precision.

Index Terms—de novo assembly, gap filling, safe solutions, dynamic programming, graph connectivity

1 Introduction

A common gap filling instance is the reconstruction of the missing sequence between contigs, once their relative order and distance is known from the scaffolding step [1], [9], [14], [21], [23], [29]. Another one is filling the missing sequence between the two reads in a paired-end read [18], [28], [36], [40], [42]. In projects with a known reference, gap filling is applied when assembling novel long insertions in a donor, given length estimates derived from mate-pair alignments to the reference [20].

In the gap filling problem we are given a sequence X containing gaps (e.g., scaffolds produced by a *de novo* assembler) and a set of sequencing reads. A gap filler aims at filling the gaps in X using the sequencing reads. A natural formulation of the gap filling problem builds an assembly graph of the sequencing reads, and fills a gap by finding an s-t path whose length is consistent with the gap length estimate (here s and t correspond to the two sequences flanking the gap) [25], [26], [40]. (In this paper we allow paths to have repeated vertices.) Since this problem is NP-hard [19], [25], [26], some methods use the heuristic of a shortest s-t path whose length is also consistent with the gap length estimate [15], [40]. Other methods are based on various extension heuristics when navigating the assembly graph from s to t [14], while others patch a gap only with those reads whose pair reliably maps to one of the two sequences flanking the gap [1], [35]. In [25], [26] we showed that the gap filling problem can be solved by dynamic programming in pseudo-polynomial time by an algorithm scaling to real data. On the bacterial and human datasets in the GAGE benchmarks [27], our method can fill in total 76 percent more gaps with 136 percent more sequence than previous methods, with a comparable error level.

In this paper we focus on gap fillers that take as input a set of sequences containing gaps and a set of short paired-end or mate-pair sequencing reads. The gap filling problem has also been addressed using preassembled contigs or long reads [7], [13]. Additionally, assembly reconciliation tools such as GAM [3], GAA [41], GAM-NGS [37], and Mix [30] are also able to perform some gap filling, but their aim is rather to merge two or more assemblies into a coherent single assembly.

As new methods filling more gaps and scaling to larger genomes are being developed, the strategies for decreasing the error rate and guaranteeing reliable results become more important. These are even more crucial when considering clinical applications, for example in detecting novel variants in tumors [24]. In Section 1.1 we review some reliable gap filling criteria proposed so far in the literature.

In this paper we address the gap filling problem with the "safe and complete" framework proposed in [33] for the contig assembly problem. This framework defines an algorithm for a problem to be: (i) *safe*, if it returns only partial solutions that are common to *all* solutions to the problem (these common partial solutions are also called *safe*), and (ii) *complete*, if it returns *all* safe solutions. In terms of the gap filling problem, a safe solution is a path of the assembly graph that is a sub-path of all possible s-t paths whose length matches the gap length estimate. This approach applies to all cases when the entire solution is not unique. For example, in Fig. 1 we show a real assembly graph instance with thousands of

• *The authors are with the Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, 00014 Helsinki, Finland. E-mail: {lmsalmel, tomescu}@cs.helsinki.fi.*

Manuscript received 30 Jan. 2017; revised 22 Nov. 2017; accepted 13 Dec. 2017. Date of publication 15 Jan. 2018; date of current version 29 Mar. 2019. (Corresponding author: Leena Salmela.)
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCBB.2017.2785831

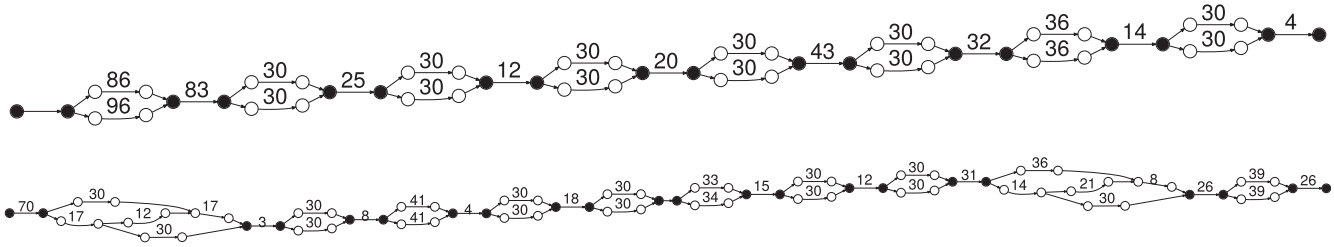

Fig. 1. Two de Bruijn graphs ($k = 31$) on staph data corresponding to two gaps in the SGA assembly. They consist of all vertices and edges on an s-t path of length $\ell \in [d - \delta, d + \delta]$ (graphs $G_{d,\delta}(s, t)$); notice that these two instances are already acyclic. We represent unary paths by numbers indicating their length. The sub-paths in black are safe i.e., common to all s-t paths. *Top:* The gap length estimated by SGA is 488, and so we look for paths whose length is approximately $d^* + k = 519$ (the path in the de Bruijn graph includes the left and right flanking k-mers adding k to the length). The closest value for which there is at least one s-t path is $d = 521$. In total, there are 256 paths with length in the allowed interval. The safe sub-paths have length 234 and the precision of Gap2Seq Safe on these sub-paths is 100 percent. *Bottom:* The gap length estimated by SGA is 547 resulting in a path length estimate of 578, but the closest value for which there is at least one s-t path is $d = 559$. In total, there are 2,304 paths whose length is in the allowed interval. The safe sub-paths have length 214 and the precision of Gap2Seq Safe on these sub-paths is 100 percent. Notice that most of the bubbles of this graph are caused by single nucleotide mismatches (sequencing errors, SNVs present in the sequenced cell population, or SNV-like differences between different copies of a repeat), because their two parallel unary paths have k internal nodes.

possible filling paths, but nevertheless most of the gap is safe. To transform these safe paths into a filling sequence usable in downstream analysis, we fill the gap with an arbitrary filling path, in which we mark the safe and un-safe sub-sequences (e.g., in upper- and lower-case). Fig. 2 shows a simple example.

1.1 Previous Work on Safe Solutions

The safest criteria for having a reliable filling sequence is that the filling path is *unique*. This strategy is mentioned by Wetzel et al. [40] in connection with s-t paths whose length matches the gap length estimate. Due to the computational complexity of this latter problem, Wetzel et al. [40] fill a gap with the heuristic criterion of finding a unique *shortest* s-t path only. To our knowledge, Konnector [36] is the only method dealing with multiple s-t paths in a systematic way (Konnector underlies Sealer [21], which is a gap filler of scaffolds). It exhaustively enumerates all s-t paths in a de Bruijn graph (up to a user defined threshold), and attempts filling a gap only when there are at most a given number of paths (Sealer's manual recommends 10). It then heuristically computes a multiple alignment of the strings spelled by these paths and takes as filling sequence this consensus sequence (ambiguous positions are encoded with IUPAC codes). Other methods [18] iteratively extend the filling path with a new character if this is well-represented in the consensus of the reads overlapping it.

The idea of this paper originates from a more general research question in bioinformatics about partial solutions common to all solutions. The typical example is the contig assembly problem, where contigs are those strings that are part of all possible genomic reconstructions from an assembly graph (be they Eulerian, Hamiltonian, or just node/edge covering paths)—see e.g., [33]. Another example is the sequence alignment problem, where some studies consider reliable partial alignments [12], [16], [34], or base-pairings common to all optimal or almost-optimal alignments [4], [38]. In the combinatorial optimization community this problem is sometimes called *persistency*: persistent edges present in all maximum matchings of a bipartite graph [5], or persistent vertices present in all maximum stable sets of a graph [2].

1.2 Contributions of the Paper

The main contribution of this paper is a new approach for reliable gap filling based on the notion of safe solution with a strong theoretical foundation.

Second, we propose an efficient safe algorithm for the gap filling problem, running in time $O(dm)$, where d is the gap length estimate and m is the number of edges of the assembly graph. The algorithm is based on finding all sub-paths present in all s-t paths of an unweighted DAG constructed from the assembly graph. As opposed to [21], [36], this avoids exhaustively enumerating all paths, thus scaling to instances (such as in Fig. 1) with an arbitrary number of paths.

Third, we implement this algorithm as a gap filler of scaffolds. Although this algorithm is not also complete, our experiments on the bacterial assemblies and conservative human assemblies from the GAGE study [27] show that it retrieves, on average, over 73 percent more safe and correct bases as compared to previous methods differentiating between ambiguous and unambiguous positions, with a similar precision.

2 Methods

2.1 Problem Definition

In this paper by *graph* we mean a directed graph. Paths can have repeated vertices, and we will use the term *simple path* when we need to emphasize that a path does not have repeated vertices. For example, all paths in a directed acyclic graph (DAG) are simple. A path from s to t is called an s-t path. By *maximal* path satisfying a property $\mathcal{P}$ we mean a path that is not a sub-path of another path satisfying property $\mathcal{P}$. In an edge-weighted graph, the *length* of a path equals the sum of the weights of its edges.

We re-state below the Exact Path Length problem from [19], with the restriction that the edge lengths are positive.

Problem 1 (Exact Path Length (EPL) with positive weights). *Given a directed graph $G = (V, E)$, a weight function $w : E \to \mathbb{Z}_+$, two vertices s and t, and an integer d, find an s-t path in G of length d.*

In [25], [26] we showed that also this restricted version of the problem is NP-complete, by giving a reduction from the

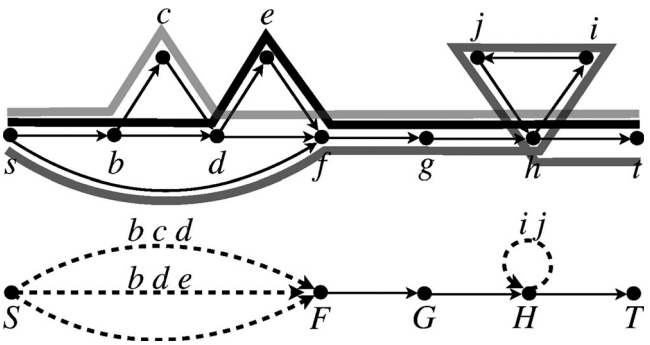

Fig. 2. *Top:* A directed graph G with all edges of weight 1. The three s-t paths in G of length 7 are drawn in light gray, dark gray, and black. The maximal paths that are sub-paths of all these three paths are (s), (f, g, h) and (h, t). *Bottom:* The gap filling solution is an arbitrary s-t path of length d, in which these three safe sub-paths are marked (e.g., in upper-case): $SbdeFGHT$.

Subset Sum problem [8]. Nonetheless, we showed that it can be efficiently solved by dynamic programming in pseudo-polynomial time $O(dm)$, where m is the number of edges of G.

We briefly sketch our modeling from [25], [26] of the gap filling problem by the EPL problem, and refer the reader to [25], [26] for other technical details. The input graph can be either a de Bruijn graph [22] or a string graph [17] built on the entire collection of reads. Our implementation uses a de Bruijn graph; in this way, all vertices are labeled by the k-mers of the reads and all edges have weight 1. Given the estimate d on the gap length and vertices s and t representing the sequences flanking the gap, we look for an s-t path whose length is within a range (i.e., distance) δ around $d + k$, and is closest to $d + k$. The spelling of this path is the string used to fill the gap.

One way to define the safe and complete gap filling problem is as follows; see also Fig. 2.

Problem 2 (Safe and complete gap filling). *Given a directed graph $G = (V, E)$, a weight function $w : E \to \mathbb{Z}_+$, two vertices s and t, and an integer d, find* all *maximal paths in G that are sub-paths of* all *solutions to the EPL problem on this instance. That is, find* all *maximal paths in G that are sub-paths of* all *s-t paths in G of length d.*

However, as mentioned above, the gap length d is usually inexact, since it has been obtained from previous upstream analyses (e.g., the scaffolding step). We can thus assume to have also a likely maximum distance δ between d and the true gap length. As such, we can consider a generalization of Problem 2 in which we look for paths common to all EPL problems for all lengths in the given interval $[d - \delta, d + \delta]$. More formally, we have the following problem:

Problem 3 (Safe and complete gap filling—extended). *Given a directed graph $G = (V, E)$, a weight function $w : E \to \mathbb{Z}_+$, two vertices s and t, and integers d, δ, find* all *maximal paths in G that are sub-paths of* all *s-t paths of length in the interval $[d - \delta, d + \delta]$.*

Clearly, Problem 3 generalizes Problem 2 when taking $\delta = 0$. Moreover, to simplify our complexity bounds, we will assume throughout this paper that $\delta = O(d)$.

We leave open the existence of an (efficient) algorithm for these two safe *and complete* gap filling problems (Problems 2 and 3). However, in the next section we propose an efficient safe algorithm for them. We will show experimentally, in Section 4, that this algorithm applied to Problem 3 has very good performance both in terms of running time and amount of safe paths that it retrieves.

2.2 A Safe Algorithm Based on Paths Common to All s-t Paths in a DAG

We assume here that the length of each edge of G is 1, otherwise we can subdivide an edge of length w into w edges of length 1 (observe that de Bruijn graphs already satisfy this property). Given G, we denote by $G_{d,\delta}(s, t)$ the graph made up of all the vertices and edges on all s-t paths in G of length $\ell \in [d - \delta, d + \delta]$. See Fig. 1 for two graphs $G_{d,\delta}(s, t)$, built on real read data for two gaps in the SGA assembly. The following lemma shows that we can compute $G_{d,\delta}(s, t)$ within the time bound $O(dm)$.

Lemma 1. *Given a graph $G = (V, E)$, vertices $s, t \in V$, and distance d and range $\delta = O(d)$, the graph $G_{d,\delta}(s, t)$ can be constructed in time $O(dm)$, where $m = |E|$.*

Proof. Let $a(v, \ell)$ equal 1 if there is an s-v path of length $\ell \leq d + \delta$, and 0 if no such path exists. Values $a(\cdot, \cdot)$ can be computed in time $O(dm)$ by dynamic programming [25], [26].

Also, for every $i \in \{0, \ldots, d + \delta\}$, let V_i be the set of vertices v of G such that there is an s-v path in G of length i, and a v-t path of length in the interval $[\max(0, d - i - \delta), d - i + \delta]$. It is clear that $V(G_{d,\delta}(s, t)) = \cup_{i=0}^{d+\delta} V_i$.

To construct the sets V_i and the graph $G_{d,\delta}(s, t)$, we initialize $V_{d+\delta} := \{t\}$ if $a(t, d + \delta) = 1$, otherwise $V_{d+\delta} := \emptyset$ For the remaining values $i \in \{0, \ldots, d + \delta - 1\}$, in decreasing order, we compute V_i as follows.

If $i \in [d - \delta, d + \delta)$ and $a(t, i) = 1$, then initialize as above $V_i := \{t\}$, otherwise $V_i := \emptyset$. To V_i we add all in-neighbors v of the vertices of V_{i+1} for which $a(v, i) = 1$ holds. During this computation we can also extract all edges from such vertices v to vertices in V_{i+1}.

There are $d + \delta + 1$ sets V_i, thus each edge of G is inspected at most $O(d)$ times. Thus, the complexity of computing $G_{d,\delta}(s, t)$ is $O(dm)$. □

Given $G_{d,\delta}(s, t)$, we compute the graph $G^{SCC}_{d,\delta}(s, t)$ of strongly connected components of $G_{d,\delta}(s, t)$. The graph $G^{SCC}_{d,\delta}(s, t)$ is usually defined to have as vertices, sets of vertices of $G_{d,\delta}(s, t)$, i.e., its strongly connected components. However, to simplify notation, if a strongly connected component of $G_{d,\delta}(s, t)$ consists of a single vertex v, we consider that the corresponding vertex of $G^{SCC}_{d,\delta}(s, t)$ is v, instead of $\{v\}$. Such strongly connected component will be called *trivial*. Overloading notation, we denote by s also the vertex of $G^{SCC}_{d,\delta}(s, t)$ corresponding to the strongly connected component of $G_{d,\delta}(s, t)$ to which s belongs, and similarly for t. By the construction of $G_{d,\delta}(s, t)$, the undirected graph underlying $G^{SCC}_{d,\delta}(s, t)$ is connected, s is the unique source of $G^{SCC}_{d,\delta}(s, t)$ and t is the unique sink of $G^{SCC}_{d,\delta}(s, t)$. Moreover, it is a standard result that $G^{SCC}_{d,\delta}(s, t)$ is a DAG and can be computed in time linear in the size of $G_{d,\delta}(s, t)$ [31].

The following lemma is the key for transferring sub-paths of all s-t paths in $G^{SCC}_{d,\delta}(s, t)$ to safe paths in G. See also Fig. 3.

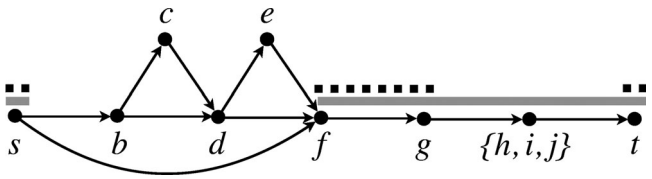

Fig. 3. The DAG $G^{SCC}_{d,\delta}(s,t)$ of strongly connected components of $G_{d,\delta}(s,t)$. Here, G is the graph from Fig. 2, and $G_{d,\delta}(s,t)$ is the subgraph of G made up only of all those vertices and edges on all s-t paths of G of length $d = 7$. In this example, we assume $\delta = 0$. The only non-trivial strongly connected component of $G_{d,\delta}(s,t)$ is $\{h,i,j\}$. The paths (s) and $(f,g,\{h,i,j\},t)$, in gray, are the maximal sub-paths of all s-t paths of G^{SCC}_d. Their sub-paths (s), (f,g) and (t), dashed black lines, are the maximal ones made up only of vertices corresponding to trivial strongly connected components of $G_{d,\delta}(s,t)$. They are also sub-paths of all s-t paths of G of length d.

Lemma 2. *Let P be a path in $G^{SCC}_{d,\delta}(s,t)$ that is a sub-path of all s-t paths in $G^{SCC}_{d,\delta}(s,t)$, and let Q be a sub-path of P made up of only vertices corresponding to trivial strongly connected components of $G_{d,\delta}(s,t)$. Then Q is a sub-path of all s-t paths of $G_{d,\delta}(s,t)$ of length in the interval $[d-\delta, d+\delta]$.*

Proof. It is enough to show that Q is a sub-path of *all* s-t paths of $G_{d,\delta}(s,t)$, since among the s-t paths of $G_{d,\delta}(s,t)$ there are also all those of all lengths in $[d-\delta, d+\delta]$. To show this, let R be an arbitrary s-t path in $G_{d,\delta}(s,t)$. Let R^{SCC} be the path in $G^{SCC}_{d,\delta}(s,t)$ made up of those strongly connected components whose vertices are used by R (observe that R^{SCC} is indeed an s-t path in $G^{SCC}_{d,\delta}(s,t)$). By the assumption on P, we have that P is a sub-path of R^{SCC}. By transitivity, it follows that also Q is a sub-path of R^{SCC}. Since Q is made up of only vertices corresponding to trivial strongly connected components of $G_{d,\delta}(s,t)$, we get that Q is a sub-path of R. □

We will now show that all such paths P from Lemma 2 can be found efficiently, using Tarjan's algorithms for finding all bridges and cut vertices of an undirected graph [11], [32].

Theorem 1. *Let $H = (V,E)$ be a DAG, with a unique source s, a unique sink t, and where the undirected graph underlying it is connected. We can find all maximal paths that are sub-paths of all s-t paths of H in time $O(m)$, where $m = |E|$.*

Proof. It is enough to find all vertices and edges that appear on all s-t paths of H; we can then pick an arbitrary s-t path P and remove from P those edges e and those vertices v (and edges incident to v) that do not appear on all s-t paths of H.

Let H^U be the undirected graph underlying H. Recall that a *bridge* in H^U is an edge whose removal makes H^U disconnected, and a *cut vertex* is a vertex whose removal makes H^U disconnected. We now prove that an edge e appears on all s-t paths of H if and only if the corresponding undirected edge e^U of H^U is a bridge.

($\Rightarrow$) Suppose that e appears on all s-t paths of H. Since s is a source and t is a sink, then any undirected s-t path in H^U must be directed from s to t in H. Therefore, e^U also belongs to all undirected s-t simple paths in H^U. Therefore, removing e^U disconnects s and t, and thus H. Therefore, e^U is a bridge.

($\Leftarrow$) Let e be an edge of H and suppose that the undirected edge e^U of H^U is a bridge. Suppose for a contradiction that there is an s-t path P in H on which e does not appear, and let P^U be the corresponding undirected path in H^U. In any DAG with a unique source s and a unique sink t, for every vertex v there is an s-v path P_v and a $v-t$ path Q_v. Let P^U_v and Q^U_v be the corresponding undirected $s-v$ and $v-t$ paths in H^U. By removing e^U from H^U, at least one of P^U_v or Q^U_v still remains in $H^U - \{e^U\}$, for every vertex v. Moreover, P^U is still a path of $H^U - \{e^U\}$, since e does not appear on P. In summary, from every vertex v there is a path in $H^U - \{e^U\}$ to either s or t, and s and t are connected by a path in $H^U - \{e^U\}$. Thus, H^U is connected, which contradicts the fact that e^U is a bridge of H^U.

In an entirely analogous manner we can prove that a vertex v appears on all s-t paths of H if and only if v is a cut vertex of H^U. All bridges and cut vertices of H^U can be found in time $O(m)$ using Tarjan's algorithms [11], [32], which gives the desired result. □

The safe algorithm for the gap filling problem, running in time $O(dm)$, is summarized in Algorithm 1. It starts by computing the graph $G_{d,\delta}(s,t)$ made up of all the vertices and edges of G on all s-t path of all lengths $\ell \in [d-\delta, d+\delta]$. It then considers its graph $G^{SCC}_{d,\delta}(s,t)$ of strongly connected components, and finds, using Theorem 1, all paths of $G^{SCC}_{d,\delta}(s,t)$ common to all s-t paths of $G^{SCC}_{d,\delta}(s,t)$. By Lemma 2 we know that these paths are safe for the gap filling problem.

Algorithm 1. An Efficient Safe Algorithm for the Gap Filling Problem

INPUT: A directed graph G, two vertices s and t, a length d and a range δ.
OUTPUT: An s-t path of length $\ell \in [d-\delta, d+\delta]$ in G with some vertices and edges belonging to all s-t of all lengths $\ell \in [d-\delta, d+\delta]$ marked.

1 compute $G_{d,\delta}(s,t)$; (by Lemma 1)
► *all s-t paths of length $\ell \in [d-\delta, d+\delta]$ in G are also s-t paths of length ℓ in $G_{d,\delta}(s,t)$, but $G_{d,\delta}(s,t)$ may also have other s-t paths;*
2 compute $G^{SCC}_{d,\delta}(s,t)$; (using [31])
► *Lemma 2 tells how sub-paths of all s-t paths of $G^{SCC}_{d,\delta}(s,t)$ induce sub-paths of all s-t paths of all lengths $\ell \in [d-\delta, d+\delta]$ in G;*
► *To find all sub-paths of all s-t paths of $G^{SCC}_{d,\delta}(s,t)$ we apply the proof of Theorem 1, as follows:*
3 compute the set B of bridges of the undirected graph underlying $G^{SCC}_{d,\delta}(s,t)$; (using [32])
4 compute the set C of cut vertices of the undirected graph underlying $G^{SCC}_{d,\delta}(s,t)$; (using [11])
5 find one s-t path P in G whose length belongs to $[d-\delta, d+\delta]$; (i.e., solve Problem EPL)
6 mark as safe the edges of P belonging to B;
7 mark as safe the vertices of P corresponding to trivial strongly connected components and belonging to C;
8 **return** P.

3 Implementations

We implemented the algorithm described in Section 2.2 as version 2.1 of Gap2Seq [25], [26], which is our tool for filling gaps in scaffolds.

Gap2Seq v1.0 [25], [26] builds a de Bruijn graph of order k from all the reads, in which all k-mers have abundance at least r, using GATB [6]. We used the default values ($k = 31$ and $r = 2$) on the bacterial data sets and $k = 61$ and $r = 2$ on

the human chromosome 14 data. For each gap, it locates in the de Bruijn graph the last e k-mers (default $e = 10$) of the left flanking contig (and adds s as in-neighbor of these vertices), and the first e k-mers of the right flanking contig (and adds t as out-neighbor of these vertices). It then computes the values $a(v, \ell)$ equalling 1 if there is an s-v path of length ℓ, and 0 if no such path exists (recall proof of Lemma 1).

It then finds the value d' that is closest to the estimated gap length (i.e., marked by N's inside the scaffold) and for which there is at least one s-t path of length d'. This value d' is the one used in later steps. If no such value exists within a user specified distance δ to the gap length estimate (default $\delta = 500$), then the gap is left unfilled. Gap2Seq v1.0 traces back in this dynamic programming table an arbitrary s-t path of length d' (see [25], [26] for further implementation details and optimizations).

Gap2Seq v2.1 uses the values $a(v, \ell)$ computed previously, for all $\ell \leq d + \delta$, to construct the graph $G_{d,\delta}(s, t)$ and continue as in Algorithm 1. The default value for δ is, as above, $\delta = 500$. It then outputs the spelling of an arbitrary s-t path P of length d', where d' is chosen as above for Gap2Seq v1.0. However, in the spelling of P safe bases are now marked in upper-case, and un-safe ones in lower-case. A base is considered safe if at least one k-mer (i.e., vertex in the de Bruijn graph) containing that base has been classified as safe. Otherwise a base is considered un-safe.

Gap2Seq v2.1 runs in parallel on the gap level, whereas Gap2Seq v1.0 is parallelized only on the scaffold level. We included a parameter (`-all-upper`) in Gap2Seq v2.1 which omits running Algorithm 1 and reports all filled bases in upper case, thus mimicking the behavior of Gap2Seq v1.0. In the experimental results Gap2Seq Safe denotes Gap2Seq v2.1 with default parameters and Gap2Seq denotes Gap2Seq v2.1 run with the `-all-upper` parameter.

We compared the quality of our method to other tools for filling gaps in scaffolds, namely GapFiller [1], SOAPdenovo's [14] stand alone tool GapCloser, and Sealer [21] (part of the ABySS distribution). GapFiller v1.10 and GapCloser v1.12 were run with default parameters. GapFiller accepts either BOWTIE or BWA read alignments to the contigs, and we tested both versions. Sealer v1.9 was run as suggested in the manual (using parameters `-k90 -k80 -k70 -k60 -k50 -k40 -k30 -j 8 -P 10`). Thus, Sealer builds different de Bruijn graphs for $k \in \{90, 80, \ldots, 30\}$, and for each instance it runs the gap filler Konnector [36]. It then merges all gap filling solutions into a single solution. Parameter $P = 10$, suggested by the manual, forces Konnector to fill a gap only if there are at most 10 s-t paths. We also tested other larger values of P, but $P = 10$ gave the best results: see Section 4.4.

For GapFiller and GapCloser, all bases in filled gaps were considered safe because the tools do not differentiate between safe and unsafe bases. Sealer outputs ambiguous bases using IUPAC codes and so we considered bases outputted as A, C, G or T as safe, and all other bases as unsafe. For Gap2Seq Safe we consider only bases that were declared safe by Algorithm 1. We also implemented a version of Gap2Seq in which we fill a gap only if there is a unique s-t path of length d [40], and declare all its bases as safe, which we call Gap2Seq Unique.

TABLE 1
Read Data Sets Used in the Evaluation

Organism	Library	Mean insert size	SD of insert size	Read length	Coverage
staph	short frag	180	30	101	45x
staph	long frag	3,500	300	37	45x
rhodo	short frag	180	30	101	45x
rhodo	long frag	3,500	300	101	45x
human14	short frag	155	20	101	42x
human14	long frag	2,500	381	101	26x

4 Experimental Evaluation

We experimented with the data sets from the GAGE benchmark [27], using the scaffolds produced by nine genome assemblers for *Staphylococcus aureus*, *Rhodobacter sphaeroides* and human chromosome 14 data (hereafter called staph, rhodo, and human14, respectively). The GAGE read sets from these organisms are described in Table 1. See the GAGE study [27] for details on the nine assemblers and their scaffolds. We report aggregate results over scaffolds produced by different assemblers. The detailed results for scaffolds produced by each assembler are available in the Supplementary Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2017.2785831. All experiments were run on a 32 GB RAM machine with 8 cores.

As observed in [26], the quality of the gap filling results heavily depends on the quality of the previous contig assembly and scaffolding steps. Thus, as done in [26], we will divide the results on human14 depending on the quality of the initial assembly. We classified these initial assemblies as: *conservative*, if they have less than 10 misassemblies (ABySS and SGA), *moderate*, if they have less than 100 misassemblies (ABySS2, Allpaths-LG, CABOG), and *aggressive* otherwise (Bambus2, MSR-CA, SOAPdenovo, Velvet). From the Supplementary Material, available online, one can also observe that all aggressive assemblies had in fact more than 1,000 misassemblies, and also most bacterial assemblies fall into the conservative category.

4.1 Evaluating the Practical Strategy from Section 2.2

Reducing $G_{d,\delta}(s, t)$ to its DAG of strongly connected components $G^{SCC}_{d,\delta}(s, t)$ makes unretrievable the safe paths inside non-trivial strongly connected components. However, as shown in Table 2, most of the graphs $G_{d,\delta}(s, t)$ do not contain any non-trivial strongly connected components (i.e., they are DAGs themselves and $G^{SCC}_{d,\delta}(s, t) = G_{d,\delta}(s, t)$).

Next, we tested how much overhead Algorithm 1 adds to the running time of Gap2Seq, by measuring the wall clock running time and the peak memory usage of both Gap2Seq and Gap2Seq Safe on the bacterial genomes (Table 3). We observe that the runtime increases by less than 25 percent. The increase in runtime is largely due to Gap2Seq Safe searching for all paths in the interval $[d - \delta, d + \delta]$, whereas Gap2Seq does not need to look for paths that deviate from d more than the optimal solution.

TABLE 2
Statistics of the Extracted Subgraphs $G_{d,\delta}(s,t) = (V_{d,\delta}(s,t), E_{d,\delta}(s,t))$ and $G^{SCC}_{d,\delta}(s,t) = (V^{SCC}_{d,\delta}(s,t), E^{SCC}_{d,\delta}(s,t))$ on the Staph, Rhodo, and Human14 Data Sets

Organism	Gaps	Gap Length	$\lvert V_{d,\delta}(s,t)\rvert$	$\lvert E_{d,\delta}(s,t)\rvert$	Nontrivial SCCs in $G_{d,\delta}(s,t)$	Vertices in nontrivial SCCs in $G_{d,\delta}(s,t)$	$\lvert V^{SCC}_{d,\delta}(s,t)\rvert$	$\lvert E^{SCC}_{d,\delta}(s,t)\rvert$
staph	853	423.0	1,030.8	1,043.2	0.197	417.5	613.5	617.2
rhodo	776	459.0	1,184.0	1,268.0	0.084	318.4	865.6	871.3
human14	33,568	234.0	11,276.1	11,570.8	0.230	8,404.5	2,871.9	2,934.7

The presented numbers are averages over the graphs $G_{d,\delta}(s,t)$ and $G^{SCC}_{d,\delta}(s,t)$ constructed on all gaps on all assemblies.

TABLE 3
Sum of Running Times and Peak Memory Usage over All Assemblies

Gap filler	staph		rhodo		human14	
	Runtime (s)	Peak memory (GB)	Runtime (s)	Peak memory (GB)	Runtime (s)	Peak memory (GB)
GapCloser	220	0.747	272	0.609	17,630	11.982
GapFiller Bowtie	2,308	0.175	4,277	0.244	237,328	3.725
GapFiller BWA	4,034	0.195	10,371	0.260	364,425	3.731
Gap2Seq	454	0.531	35,732	16.675	5,399,851	28.219
Gap2Seq Safe	567	0.528	41,652	16.125	6,108,602	28.182
Sealer	1,167	0.528	1,524	0.528	39,461	0.646

TABLE 4
The Ratio of Bases in Unaligned Filled Gaps over the Total Number of Filled Bases (Columns "% of Total Filled"), and over the Total Number of Gap Bases in the Original Scaffolds (Columns "% of Total Gap") Aggregated over All Assemblies for Which the Methods Could Run

Gap filler	staph		rhodo		human14 (conservative)		human14 (moderate)		human14 (aggressive)	
	% of total filled	% of total gap	% of total filled	% of total gap	% of total filled	% of total gap	% of total filled	% of total gap	% of total filled	% of total gap
GapCloser	16%	8%	37%	2%	28%	15%	32%	12%	60%	12%
GapFiller Bowtie	63%	9%	69%	2%	53%	27%	61%	11%	81%	20%
GapFiller BWA	57%	10%	64%	4%	52%	29%	65%	15%	83%	27%
Gap2Seq	14%	12%	7%	2%	5%	1%	25%	5%	33%	1%
Sealer	11%	4%	1%	0%	1%	0%	7%	0%	13%	1%
Gap2Seq Safe	15%	12%	8%	2%	5%	1%	26%	5%	33%	1%
Gap2Seq Unique	0%	0%	0%	0%	0%	0%	0%	0%	19%	0%

4.2 Evaluating the Filled Sequence

4.2.1 Global Evaluation

There are two perspectives of evaluating the filled scaffolds. The first one is to evaluate the quality of the entire filled scaffold. This can be done by aligning it to the reference genome and measuring overall assembly metrics with a tool like QUAST [10]: e.g., NGA50 [27] and the number of misassemblies. In this case, it is undesirable to mask the unsafe bases with N's, since the safe bases may form too short sequences to be aligned reliably. This could lead to, e.g., artificially decreasing the NGA50 value or increasing the number of misassemblies. Recall also that gaps usually arise in complex, often repeated regions of the genome, making their alignment even more difficult. Thus, the safe bases outputted by Gap2Seq Safe cannot be used alone (i.e., without the unsafe ones) for a global assembly evaluation. Considering both safe and unsafe bases is equivalent to the output of Gap2Seq v1.0.

In [25], [26] we performed a global assembly evaluation using QUAST v2.3 [10] for GapCloser, GapFiller and Gap2Seq v1.0. In the Supplementary Material, available online, of this paper we complement that evaluation with Sealer's results. Recall that Sealer outputs ambiguous bases using IUPAC codes. For the same reason that we cannot use only safe bases in an overall evaluation, we replaced each such IUPAC code with one of the bases it encodes, at random. Sealer offers no other conversion indication, and this is also inline with e.g., Gap2Seq, which chooses a filling s-t path at random.

Tables 2 and 3 from the Supplementary Material, available online, show that Gap2Seq v1.0 generally outperforms also Sealer on the bacterial data sets. Tables 4, 5, and 6 from the Supplementary Material, available online, show that on the human14 data set Gap2Seq v1.0 outperforms Sealer on the conservative assemblies, whereas Sealer is better on the aggressive assemblies. See [26] for a detailed discussion on such an evaluation. These tables also contain the absolute numbers on the length of all gaps in the original assemblies, and the length of the filled sequences.

TABLE 5
Precision and Recall

Gap filler	staph		rhodo		human14 (conservative)		human14 (moderate)		human14 (aggressive)	
	precision	recall	precision	recall	precision	recall	precision	recall	precision	recall
GapCloser	0.839	0.449	0.587	0.027	0.714	**0.379**	0.660	**0.242**	0.390	**0.079**
GapFiller Bowtie	0.365	0.049	0.296	0.008	0.459	0.232	0.378	0.069	0.186	0.045
GapFiller BWA	0.424	0.078	0.341	0.020	0.478	0.269	0.339	0.078	0.171	0.056
Gap2Seq	**0.841**	**0.668**	**0.918**	**0.205**	**0.947**	0.243	**0.732**	0.138	**0.663**	0.026
Sealer	0.891	0.291	0.979	0.034	0.986	0.120	0.922	0.057	**0.869**	**0.039**
Gap2Seq Safe	0.952	**0.503**	0.949	**0.172**	0.968	**0.216**	0.795	**0.089**	0.763	0.022
Gap2Seq Unique	**1.000**	0.250	**0.996**	0.063	**0.995**	0.090	**0.989**	0.006	0.811	0.012

We show aggregate values over all assemblies. Sealer was run for a week on the SGA assembly of staph and it did not finish. The gaps in the SGA assembly of staph on which Sealer did not finish running were not used to penalize its recall. We marked in bold the best results among rows 1–4, and among rows 5–8, respectively.

4.2.2 Gap-Level Evaluation

The second perspective is to evaluate how correct are the bases reported as safe. This pertains especially to Gap2Seq Safe, Gap2Seq Unique and Sealer. As mentioned above, we cannot align to the reference safe sequences alone, because they may be too short to be aligned reliably. Therefore, using again QUAST v2.3 [10] we produced alignments to the reference of the entire filled scaffolds. These scaffolds were the ones with both safe and unsafe bases for Gap2Seq Safe, and with IUPAC codes converted to bases for Sealer, as explained above. For Gap2Seq Unique, all bases are safe by definition. As mentioned before, for GapFiller, GapCloser and Gap2Seq v1.0 we considered all bases as being safe. For each tool, we classified each filled gap as:

- *aligned*: if QUAST reported an alignment of the gap filling sequence together with its flanking 100 bases on both sides from the filled scaffold; if a gap is closer than 100 bases from the extremity of the scaffold, the flank is that sequence until the extremity;
- *unaligned*: otherwise.

This alignment condition ensures that the filling sequence truly belongs between the corresponding flanking contigs. For example, we want to classify as an error a safe sequence in the middle of a gap that has no occurrence in the reference between its two flanking contigs. Recall again that gaps often correspond to repeated regions.

In Table 4 we show how many filled gap bases belong to unaligned filling sequences. When unable to fill a gap completely, GapCloser and GapFiller attempt to find a partial filling sequence by extending the flanking sequences. QUAST is unable to form continuous alignments for such partially filled gaps and thus, GapCloser and GapFiller produce larger amounts of filled bases that are not aligned according to the above criteria. When compared to methods producing only complete filling sequences (Sealer and Gap2Seq Unique), Gap2Seq Safe is the method penalized the most by this additional correctness condition. Nonetheless, we will see below that its performance over *all* gaps (aligned, unaligned or unfilled) is still better than Sealer's and Gap2Seq Unique's on the bacterial data sets and on conservative human assemblies.

Having an *aligned* filled sequence, we counted how many of its safe bases match the corresponding bases in the reference genome. Each such match was considered a *correct* safe base. An *incorrect* safe base is thus one that either does not belong to an aligned filling sequence, or it does not match the base to which it was aligned. Having these numbers, we computed the following accuracy metrics:

- *precision* = *#correct safe bases* / *#safe bases*, where "#safe bases" is the total number of bases declared safe (in both aligned or unaligned filled gaps); thus precision measures how many of *all* safe bases appear in an *aligned* filling sequence and match the corresponding characters in the reference;
- *recall* = *#correct safe bases* / *#total gap bases*, where "#total gap bases" is the total number of gap bases (i.e., characters N) in the original input scaffolds, independently of whether they were aligned, unaligned or unfilled; thus recall measures how many of *all* original gap bases appear as safe in an *aligned* filling sequence, and also match the corresponding characters in the reference.

TABLE 6
Precision and Recall of Gap2Seq Safe and Sealer, When Restricted to Gaps Where Gap2Seq Finds More Than 10 s-t Paths of Length $\ell \in [d-\delta, d+\delta]$

Data set	No. of gaps	Sum of gap lengths	Gap2Seq Safe		Sealer	
			Precision	Recall	Precision	Recall
staph	263 (23%)	157,828 (36%)	0.858	0.365	0.873	0.227
rhodo	143 (5%)	122,529 (8%)	0.802	0.443	0.958	0.077
human14 conservative	2,370 (11%)	911,254 (7%)	0.879	0.693	0.968	0.238
human14 moderate	1,175 (12%)	647,165 (15%)	0.751	0.427	0.852	0.113
human14 aggressive	4,265 (5%)	1,063,668 (1%)	0.615	0.391	0.913	0.265

The percentages in columns 2 and 3 indicate the ratio of the total number of gaps, and of the total gap length of the assembly, respectively. Recall is relative to these gaps with more than 10 paths.

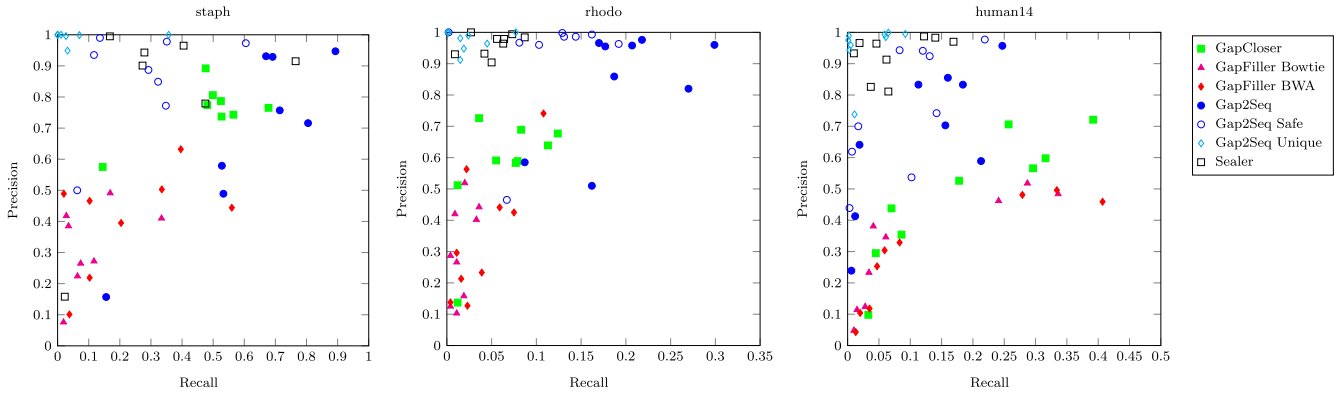

Fig. 4. Precision and recall of the gap fillers for each assembly of staph, rhodo, and human14. See the supplementary material, available online, for the exact values for each assembly.

Intuitively, precision measures how correct the safe bases are, and recall measures how much of all the gaps of the original scaffolds was filled correctly by safe bases. Ultimately, recall is the most important metric, but at the same time bases should be declared safe with a high precision. See Table 5 and Fig. 4 for results on precision and recall.

4.3 Evaluating Complex Gaps

We also evaluated how many gaps have more than 10 s-t paths of length $\ell \in [d-\delta, d+\delta]$, their proportion of all the gaps, and what is the precision and recall of Gap2Seq Safe and Sealer on these gaps. We chose the value 10 because this is the value of the parameter P for which Sealer gave best results. See Table 6.

4.4 Evaluating Sealer for Multiple Bounds on the Number of Filling Paths

We studied how Sealer's performance is influenced by the parameter P, the maximum number of s-t paths allowed for a gap to be filled. Sealer's manual suggests setting $P = 10$, but we tested $P \in \{10, 100, 1000, 10000\}$. As we can see from Table 7, the results do not improve for larger P. This behavior might be due to the heuristic strategy of computing an iterative alignment of the increasing number of possible filling sequences of a gap. Sealer's running time for varying P might also be explained by the additional internal choice that Sealer makes concerning the de Bruijn graph with which it works (recall that it tries $k \in \{90, 80, \ldots, 30\}$). Since $P = 10$ gives the best results, all other experimental results concerning Sealer refer only to parameter $P = 10$.

5 Discussion

5.1 Bacterial Data Sets

Among the gap fillers that do not differentiate between safe and unsafe bases (GapCloser, GapFiller and Gap2Seq), Gap2Seq has the best overall precision and recall, with almost double precision and almost 8 times better recall on the rhodo data set. This is consistent with our previous results [25], [26] and due to the fact that we look for paths in the entire de Bruijn graph between the gap flanks, and that we are guided by the gap length estimate. However, we note that the precision and recall of GapCloser and GapFiller are adversely affected by the high proportion of unaligned bases, as noted in Section 4.2.2. Nevertheless, as discussed in Section 4.2.1 and Supplementary Material, available online, Gap2Seq globally outperforms GapCloser and GapFiller.

Gap2Seq Unique provides a baseline for the performance of gap fillers distinguishing between safe and unsafe bases as even a simple method should be able to classify as safe gaps with a unique path. The precision of Gap2Seq Unique is very high but its recall, especially on rhodo, is low.

On the staph data Gap2Seq Safe has a higher precision than Sealer, but on the rhodo assemblies the precision of Sealer is higher. However, the overall recall of Gap2Seq Safe is clearly better than the recall of Sealer: 73 percent better recall on staph, and 406 percent better recall on rhodo. Our experiments suggest this is the case because Gap2Seq Safe scales to gaps where the number of paths is large. Indeed, Table 6 (and recall also Fig. 1) shows that a non-negligible proportion of the total gap length is complex, and Gap2Seq Safe has a precision and recall consistent with the full results. Table 6 shows that Sealer is able to deal with

TABLE 7
Precision, Recall, and Total Running Time of Sealer, over All Assemblies on Staph and Rhodo, for Varying Values of its P Parameter

	staph			rhodo		
P	Precision	Recall	Runtime (s)	Precision	Recall	Runtime (s)
10	0.891	0.291	1,167	0.979	0.034	1,524
100	0.680	0.274	893	0.863	0.034	1,532
1,000	0.683	0.278	931	0.780	0.033	2,785
10,000	0.681	0.278	918	0.774	0.033	1,578

some of these complex gaps because it tries filling them for multiple values of k. However, especially on the rhodo data set, the recall of Sealer is low. Note that the same heuristic strategy of using multiple values of k could also be applied to Gap2Seq, but then it is harder to define what a safe base is.

Comparing the best of the above two categories (i.e., Gap2Seq and Gap2Seq Safe), we observe that Gap2Seq Safe is able to correctly classify as safe a large proportion of the total gap length correctly retrievable by Gap2Seq. This shows that our definition of safe filling path is a well-motivated one. Recall also that the output of Gap2Seq Safe also contains the unsafe bases. Thus the remaining sequences that add up to the recall of Gap2Seq come from bubbles and other more complex regions of the assembly graph. We leave it as future work to identify and extract the safe bases from these regions. The precision of Gap2Seq Safe is better than that of Gap2Seq on most assemblies, with a more significant increase on staph than on rhodo.

5.2 Human14 Data Set

As observed also in [26], the results on human assemblies heavily depend on the quality of the initial assembly. For example, on the ABySS assembly (conservative), Gap2Seq generally fills paths or reports that no path exists, whereas on the SOAPdenovo assembly (aggressive), Gap2Seq abandons many longer gaps because it exceeds the memory limit [26, Fig. 5].

On the conservative and moderate assemblies, where Gap2Seq outperforms Sealer in the global evaluation (recall Section 4.2.1), we see that also Gap2Seq Safe generally outperforms Sealer on the gap level evaluation (Table 5): on conservative assemblies Gap2Seq Safe has almost double recall, with a small drop in precision, and on moderate assemblies Gap2Seq maintains the double recall, with a larger drop in precision. This suggests that Sealer avoids making wrong predictions by not filling complex gaps with many paths. This interpretation is also supported by the drop in recall. Similar to the results on bacterial genomes, Table 6 shows that Gap2Seq Safe can fill many complex gaps with safe bases correctly. Sealer is also able to fill some of these gaps due to its strategy of trying multiple values of k but it clearly has a lower recall than Gap2Seq Safe.

We also observe that Gap2Seq Safe has a similar behavior with respect to Gap2Seq as in bacterial genomes. Namely, the precision always increases, with only a slight drop in recall.

To conclude, Gap2Seq Safe is naturally limited by the performance of Gap2Seq, which is best seen on complex gaps and on aggressive assemblies. On correct initial human14 assemblies, Gap2Seq Safe shows a similar behavior as in the bacterial data sets.

6 Conclusion

In this paper we studied safe (partial) solutions to a gap filling instance, and showed that they are well-motivated in practice, especially on correct initial assemblies. Indeed, on bacterial genomes and on conservative human assemblies, our method can detect as reliable at least 73 percent more gap bases as compared to the few previous methods distinguishing between ambiguous and unambiguous filling bases, with a comparable precision level.

To improve the runtime of Gap2Seq, we could apply a similar approach as when extending Gap2Seq for genotyping insertions [39]. Similar to GapFiller [1], for each gap we would first determine a subset R' of reads whose mates align on the flanks of the gap. The traversal of the graph when closing this particular gap would then be limited to those vertices derived from the reads R'.

Our method outputs entire filling sequences—thus being compatible with later downstream analyses—but it cannot improve global assembly metrics such as NGA50. However, by marking the safe bases, it can provide a more informed answer on the gap filling sequence. This is a challenging problem considering that gaps arise in complex regions of a genome.

Gap2Seq 2.1 is freely available at http://www.cs.helsinki.fi/u/lmsalmel/Gap2Seq/.

Acknowledgments

The authors thank the anonymous reviewers for their insightful comments which helped us to improve the manuscript. This work was supported by the Academy of Finland [284598 (CoECGR); 308030 and 314170 to L.S.; 274977 to A.I.T.]. An earlier version of this paper appeared as a short abstract in the *Proceedings of WABI '16, Workshop on Algorithms in Bioinformatics*, LNBI 9838, Springer, 2016, page xiv.

References

[1] M. Boetzer and W. Pirovano, "Toward almost closed genomes with GapFiller," *Genome Biol.*, vol. 13, no. 6, 2012, Art. no. R56.
[2] E. Boros, M. C. Golumbic, and V. E. Levit, "On the number of vertices belonging to all maximum stable sets of a graph," *Discrete Appl. Math.*, vol. 124, no. 1–3, pp. 17–25, 2002.
[3] A. Casagrande, C. Del Fabbro, S. Scalabrin, and A. Policriti, "GAM: Genomic assemblies merger: A graph based method to integrate different assemblies," in *Proc. IEEE Int. Conf. Bioinf. Biomed.*, 2009, pp. 321–326.
[4] K.-M. Chao, R. C. Hardison, and W. Miller, "Locating well-conserved regions within a pairwise alignment," *Comput. Appl. Biosci.*, vol. 9, no. 4, pp. 387–396, 1993.
[5] M.-C. Costa, "Persistency in maximum cardinality bipartite matchings," *Operations Res. Lett.*, vol. 15, no. 3, pp. 143–149, 1994.
[6] E. Drezen, et al., "GATB: Genome assembly & analysis tool box," *Bioinf.*, vol. 30, no. 20, pp. 2959–2961, 2014.
[7] A. C. English, et al., "Mind the gap: Upgrading genomes with Pacific Biosciences RS long-read sequencing technology," *PLoS One*, vol. 7, 2012, Art. no. e47768.
[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: Freeman, 1979.
[9] S. Gnerre, et al., "High-quality draft assemblies of mammalian genomes from massively parallel sequence data," *Proc. Nat. Academy Sci. United States America*, vol. 108, no. 4, pp. 1513–1518, Dec. 2011.
[10] A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler, "QUAST: Quality assessment tool for genome assemblies," *Bioinf.*, vol. 29, no. 8, pp. 1072–1075, 2013.
[11] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Commun. ACM*, vol. 16, no. 6, pp. 372–378, Jun. 1973.
[12] L. Jaroszewski, W. Li, and A. Godzik, "In search for more accurate alignments in the twilight zone," *Protein Sci.*, vol. 11, no. 7, pp. 1702–1713, 2002.
[13] S. Kosugi, H. Hirakawa, and S. Tabata, "GMcloser: Closing gaps in assemblies accurately with a likelihood-based selection of contig or long-read alignments," *Bioinf.*, vol. 31, no. 23, pp. 3733–3741, 2015.
[14] R. Luo, et al., "SOAPdenovo2: An empirically improved memory-efficient short-read de novo assembler," *Gigasci.*, vol. 1, 2012, Art. no. 18.
[15] P. Medvedev and M. Brudno, "Maximum likelihood genome assembly," *J. Comput. Biol.*, vol. 16, no. 8, pp. 1101–1116, Aug. 2009.

[16] H. T. Mevissen and M. Vingron, "Quantifying the local reliability of a sequence alignment," *Protein Eng. Des. Selection*, vol. 9, no. 2, pp. 127–132, 1996.
[17] E. W. Myers, "The fragment assembly string graph," *Bioinf.*, vol. 21, no. suppl_2, pp. ii79–ii85, 2005.
[18] F. Nadalin, F. Vezzi, and A. Policriti, "GapFiller: A de novo assembly approach to fill the gap within paired reads," *BMC Bioinf.*, vol. 13, no. Suppl 14, 2012, Art. no. S8.
[19] M. Nykänen and E. Ukkonen, "The exact path length problem," *J. Algorithms*, vol. 42, no. 1, pp. 41–53, 2002.
[20] S. Pabinger, et al., "A survey of tools for variant analysis of next-generation genome sequencing data," *Briefings Bioinf.*, vol. 15, no. 2, pp. 256–278, Jan. 2013.
[21] D. Paulino, R. L. Warren, B. P. Vandervalk, A. Raymond, S. D. Jackman, and I. Birol, "Sealer: A scalable gap-closing application for finishing draft genomes," *BMC Bioinf.*, vol. 16, no. 1, 2015, Art. no. 230.
[22] P. A. Pevzner, H. Tand, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proc. Nat. Acad. Sci. USA*, vol. 98, no. 17, pp. 9748–9753, 2001.
[23] P. A. Pevzner and H. Tang, "Fragment assembly with double-barreled data," *Bioinf.*, vol. 17, no. suppl 1, pp. S225–S233, 2001.
[24] T. J. Pugh, et al., "The genetic landscape of high-risk neuroblastoma," *Nature Genetics*, vol. 45, no. 3, pp. 279–284, Mar. 2013.
[25] L. Salmela, K. Sahlin, V. Mäkinen, and A. I. Tomescu, "Gap filling as exact path length problem," in *Proc. Int. Conf. Res. Comput. Mol. Biol.*, 2015, pp. 281–292.
[26] L. Salmela, K. Sahlin, V. Mäkinen, and A. I. Tomescu, "Gap filling as exact path length problem," *J. Comput. Biol.*, vol. 23, no. 5, pp. 347–361, 2016. Extended version of [25].
[27] S. L. Salzberg, et al., "GAGE: A critical evaluation of genome assemblies and assembly algorithms," *Genome Res.*, vol. 22, no. 3, pp. 557–567, Dec. 2012.
[28] D. H. Silver, S. Ben-Elazar, A. Bogoslavsky, and I. Yanai, "ELOPER: Elongation of paired-end reads as a pre-processing tool for improved de novo genome assembly," *Bioinf.*, vol. 29, no. 11, pp. 1455–1457, 2013.
[29] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. M. Jones, and I. Birol, "ABySS: A parallel assembler for short read sequence data," *Genome Res.*, vol. 19, pp. 1117–1123, 2009.
[30] H. Soueidan, et al., "Finishing bacterial genome assemblies with Mix," *BMC Bioinf.*, vol. 14, no. Suppl 15, 2013, Art. no. S16.
[31] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, 1972.
[32] R. E. Tarjan, "A note on finding the bridges of a graph," *Inf. Process. Lett.*, vol. 2, no. 6, pp. 160–161, 1974.
[33] A. I. Tomescu and P. Medvedev, "Safe and complete contig assembly via omnitigs," in *Proc. Int. Conf. Res. Comput. Mol. Biol.*, 2016, pp. 152–163.
[34] M. L. Tress, O. Graña, and A. Valencia, "SQUARE—Determining reliable regions in sequence alignments," *Bioinf.*, vol. 20, no. 6, pp. 974–975, 2004.
[35] I. J. Tsai, T. D. Otto, and M. Berriman, "Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps," *Genome Biol.*, vol. 11, no. 4, 2010, Art. no. R41.
[36] B. P. Vandervalk, et al., "Konnector: Connecting paired-end reads using a bloom filter de Bruijn graph," in *Proc. IEEE Int. Conf. Bioinf. Biomed.*, Nov. 2014, pp. 51–58.
[37] R. Vicedomini, F. Vezzi, S. Scalabrin, L. Arvestad, and A. Policriti, "GAM-NGS: Genomic assemblies merger for next generation sequencing," *BMC Bioinf.*, vol. 14, no. Suppl 7, 2013, Art. no. S6.
[38] M. Vingron and P. Argos, "Determination of reliable regions in protein sequence alignments," *Protein Eng.*, vol. 3, no. 7, pp. 565–569, 1990.
[39] R. Walve, L. Salmela, and V. Mäkinen, "Variant genotyping with gap filling," *PLoS One*, vol. 12, no. 9, 2017, Art. no. e0184608.
[40] J. Wetzel, C. Kingsford, and M. Pop, "Assessing the benefits of using mate-pairs to resolve repeats in de novo short-read prokaryotic assemblies," *BMC Bioinf.*, vol. 12, no. 1, 2011, Art. no. 95.
[41] G. Yao, L. Ye, H. Gao, P. Minx, W. C. Warren, and G. M. Weinstock, "Graph accordance of next-generation sequence assemblies," *Bioinf.*, vol. 28, no. 1, pp. 13–16, 2012.
[42] A. V. Zimin, G. Marçais, D. Puiu, M. Roberts, S. L. Salzberg, and J. A. Yorke, "The MaSuRCA genome assembler," *Bioinf.*, vol. 29, no. 21, pp. 2669–2677, 2013.

Leena Salmela received the DSc (Tech) degree in computer science from the Helsinki University of Technology, Finland, in 2009. Since 2009, she has worked with the University of Helsinki, Finland, where she currently holds the position of Academy of Finland research fellow. Her research interests include string algorithms and bioinformatics.

Alexandru I. Tomescu received the PhD degree in computer science from the University of Udine, Italy, in 2012. After spending six months with the Technical University in Berlin, Germany, he joined the Genome-scale Algorithmics Group with the University of Helsinki, Finland, where he currently holds an Academy of Finland Postdoctoral Researcher Fellowship.