Bioinformatics, 32, 2016, i494–i502
doi: 10.1093/bioinformatics/btw450
ECCB 2016

Information-optimal genome assembly via sparse read-overlap graphs

Ilan Shomorony[1], Samuel H. Kim[1], Thomas A. Courtade[1,*] and David N. C. Tse[1,2,*]

[1]Department of Electrical Engineering & Computer Sciences, University of California, Berkeley, CA, USA and [2]Department of Electrical Engineering, Stanford University, Stanford, CA, USA

*To whom correspondence should be addressed.

Abstract

Motivation: In the context of third-generation long-read sequencing technologies, read-overlap-based approaches are expected to play a central role in the assembly step. A fundamental challenge in assembling from a read-overlap graph is that the true sequence corresponds to a Hamiltonian path on the graph, and, under most formulations, the assembly problem becomes NP-hard, restricting practical approaches to heuristics. In this work, we avoid this seemingly fundamental barrier by first setting the computational complexity issue aside, and seeking an algorithm that targets *information limits*. In particular, we consider a basic feasibility question: when does the set of reads contain enough *information* to allow unambiguous reconstruction of the true sequence?
Results: Based on insights from this information feasibility question, we present an algorithm—the Not-So-Greedy algorithm—to construct a sparse read-overlap graph. Unlike most other assembly algorithms, Not-So-Greedy comes with a performance guarantee: whenever information feasibility conditions are satisfied, the algorithm reduces the assembly problem to an Eulerian path problem on the resulting graph, and can thus be solved in linear time. In practice, this theoretical guarantee translates into assemblies of higher quality. Evaluations on both simulated reads from real genomes and a PacBio *Escherichia coli* K12 dataset demonstrate that Not-So-Greedy compares favorably with standard string graph approaches in terms of accuracy of the resulting read-overlap graph and contig N50.
Availability: Available at github.com/samhykim/nsg
Contact: courtade@eecs.berkeley.edu or dntse@stanford.edu
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Modern DNA-sequencing pipelines are based on a two-step process. First, tens or hundreds of millions of fragments from random and unknown locations of the DNA sequence are read via *shotgun sequencing*. These fragments, called reads, are then merged to each other with the goal of recovering the true unknown genome. The task of reconstructing the original sequence from a large number of short reads is known as the *Assembly Problem* and is one of the fundamental algorithmic problems in bioinformatics.

While the assembly problem (AP) does not have a unique 'canonical' formulation agreed upon by all bioinformaticians, the problem is widely regarded as computationally hard (Nagarajan and Pop, 2009; Medvedev *et al.*, 2007). On one hand, a classical formalization, based on the 'Occam's razor' (or parsimony) principle, models the AP as the *Shortest Common Superstring* problem (SCSP). Since SCSP is known to be NP-hard, early studies on the problem focused on the development of approximation algorithms (Blum *et al.*, 1994; Ming, 1990; Tarhio and Ukkonen, 1988). Recent works on assembly algorithms, on the other hand, tend to model the AP as the problem of finding an appropriate path on a graph constructed from the reads. Graph-based approaches to AP are usually classified into two categories: approaches based on *read-overlap graphs* and approaches based on *de Bruijn graphs*.

In a read-overlap graph, each vertex corresponds to a read and edges are used to represent the overlaps between reads. In this setting, the objective is to find a path that visits every node once; i.e. a Hamiltonian path (or a generalized Hamiltonian path, if we allow nodes to be visited multiple times). (We use the term Generalized

Hamiltonian path following the terminology in Nagarajan and Pop (2009). A path that visits every node at least once is also refered to in the literature as a spanning path.) As shown in Nagarajan and Pop (2009), if we model the AP as the problem of finding the shortest generalized Hamiltonian path or a generalized Hamiltonian path of a desired length, we have an NP-hard formulation.

Part of the excitement surrounding the introduction of the de Bruijn graph-based formulation (Pevzner *et al.*, 2001) is that the sequence is no longer a generalized Hamiltonian path, but rather a path that traverses every *edge* on the graph. In a de Bruijn graph (Pevzner, 1995), vertices correspond to length-K substrings—or K-mers—extracted from the reads, and two K-mers are connected by an edge if they appear consecutively in a read. Since all edges now belong to the sequence path, we look for an *Eulerian* path (if we require each edge to be traversed exactly once) or a *Chinese Postman* path (Medvedev *et al.*, 2007) (if we require each edge to be traversed at least once), both of which can be found in polynomial time. Nevertheless, in order to make sure that the resulting sequence is consistent with all the reads, one must require the Eulerian (or Chinese Postman) path to contain the set of short paths corresponding to each read. This additional constraint results in a problem known as the Eulerian Superpath Problem (Pevzner *et al.*, 2001), which is also known to be NP-hard (Nagarajan and Pop, 2009).

In light of all these computational hardness results, it is natural to presume that the AP is fundamentally difficult. However, if we leave aside the question of the computational tractability of these formulations, is it clear that they lead to the reconstruction of the *true* sequence? As pointed out by Medvedev and Brudno, 2009, parsimony-based formulations can fail at producing the true sequence because long repeats tend to be under-represented in the shortest sequence that is consistent with the data. For this reason, it is unclear whether the worst-case time complexity of combinatorial optimization based formulations is a meaningful measure of the hardness of the AP. Furthermore, one can easily devise instances of the AP where the absence of reads from some part of the sequence deems the task of reconstructing the true sequence impossible. In such cases, the computational hardness of the problem is irrelevant as there is not enough *information* for the sequence to be recovered, and the optimal solution under any formulation will in general be incorrect. Therefore, a more careful treatment of the assembly task should be concerned with two questions:

- When is there enough information in the set of reads to render the AP feasible?
- Can one devise efficient algorithms which recover the true sequence for the feasible instances of the AP?

The present paper should be understood as following the line of work of Medvedev and Brudno (2009), Nagarajan and Pop (2009) and Medvedev *et al.* (2007) in the study of the basic formulation of the AP and the computational challenges associated with it, but having the aforementioned questions as starting point.

This information-theoretic approach to the AP was first considered by Bresler *et al.* (2013). The authors characterized classes of instances of the AP where the set of reads do not contain enough information for the true sequence to be unambiguously reconstructed. In particular, two types of repeat patterns, illustrated in Figure 1, were identified as the main informational bottlenecks: interleaved repeats and triple repeats. Based on these observations, Bresler *et al.* (2013) derived *bridging* conditions that the set of reads must satisfy to prevent the two ambiguous scenarios in Figure 1a and b from occurring. In addition, Bresler *et al.* (2013) introduced a de Bruijn

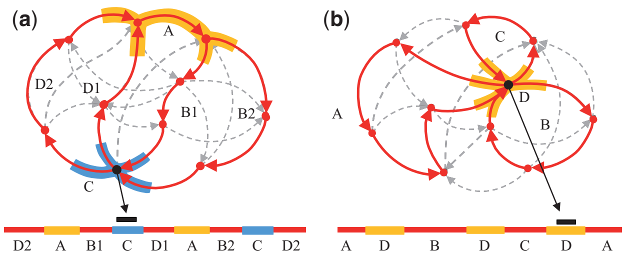

Fig. 1. The two basic scenarios where the set of reads does not contain enough information to allow perfect assembly. The grey dashed edges correspond to edges in the read-overlap graph that are not part of the true path (shown in red). (a) The existence of a pair of interleaved repeats (A and C) that are both longer than the read length L (the black bar) makes it impossible for any algorithm to distinguish between the cycles A–B1–C–D1–A–B2–C–D2 and A–B1–C–D2–A–B2–C–D1. (b) The existence of a triple repeat that is longer than L makes it impossible for any algorithm to distinguish between the cycles A–B–C and A–C–B

graph-based algorithm—Multibridging—whose performance nearly matches this information-theoretic lower bound. More precisely, by considering a probabilistic framework where N reads of length L are sampled independently and uniformly at random from the sequence, the (N, L) *feasibility region*, illustrated in Figure 2, was characterized for a variety of genomes. One of the main features of these regions is a critical read length ℓ_{crit}, below which assembly is infeasible. This critical read length (defined in Supplementary Material G) can be in the thousands even for small bacterial genomes, underscoring the importance of long-read technologies.

Extending the ideas of de Bruijn graph-based assemblers (which were rather successful in the assembly of short-read sequencing data (Pevzner *et al.*, 2001; Peng *et al.*, 2010; Zerbino and Birney, 2008)) to fully exploit the power of third-generation long-read sequencing is quite difficult (Lin *et al.*, 2016). The first reason is that by shredding the reads into K-mers, it is harder to take advantage of the full length of the reads in order to resolve repeats and obtain long *contigs* (unambiguously assembled segments of the genome). The second one is that the de Bruijn graph construction is very sensitive to read errors, as many false K-mers are created, and the high error rates associated with long-read sequencing technologies make this approach quite challenging. Read-overlap approaches, on the contrary, by not breaking the reads into small K-mers, and by connecting reads that contain long *approximate* overlaps, have the potential to generate much longer contigs, and in an noise-robust way. In fact,

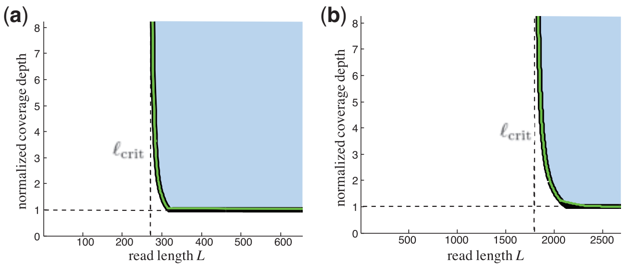

Fig. 2. Feasibility region for (a) *R. sphaeroides* and (b) *S. aureus*, for a target error probability $\epsilon = 0.01$. The thick black curve is a feasibility lower bound for any algorithm, and the green line represents the performance of the Multibridging algorithm (Bresler *et al.*, 2013). In the vertical axis, the normalized coverage depth is N/N_{LW}, where $N_{LW} \approx \frac{G}{L}\log\left(\frac{G}{\epsilon}\right)$ is the Lander–Waterman coverage depth (see Supplementary Material G)

most available long-read assemblers (Chin *et al.*, 2013; Li, 2015; Berlin *et al.*, 2015) make use of the read-overlap approach.

However, the current theoretical understanding of the read-overlap framework is limited. The existence of repeats in the sequenced genome creates many *spurious* edges on the read-overlap graph. This makes the problem of identifying the correct sequence on the graph very challenging, and naturally cast as the (generalized) Hamiltonian path problem. Two natural questions arise:

1. Can read-overlap based approaches achieve the information limits characterized by Bresler *et al.* (2013)?
2. For the information-feasible instances of the assembly problem, can one find the true path on the read-overlap graph efficiently?

In this work, we answer both these questions in the affirmative. Inspired by insights from the information-theoretic necessary conditions introduced by Bresler *et al.* (2013), we describe a new algorithm called Not-So-Greedy for the construction of a *sparse* read-overlap graph. Although sparse, we show that this read-overlap graph is 'sufficient' for assembly, in the sense that with high probability the true sequence corresponds to a path on the graph as long as we are in the information-theoretic feasibility region (see Fig. 2). Furthermore, the Not-So-Greedy algorithm proceeds by pruning and condensing the resulting read-overlap graph, ultimately producing a graph where the true sequence corresponds to an Eulerian path, and can be found in linear time. Therefore, just like the de Bruijn graph was used as a way to cast the AP as an Eulerian path problem, the techniques introduced in this paper can be seen as an alternative path towards an Eulerian path problem, but via sparse read-overlap graphs. We conclude that while the AP when considered as a general optimization problem may be computationally intractable, the information-feasible instances can be efficiently solved.

2 Preliminaries

Let x be a string of ℓ symbols from the alphabet $\Sigma = \{A, C, G, T\}$. Let $|x| = \ell$ be the length of the string, and $x[i]$ be its ith symbol. A substring of x is a contiguous interval of the symbols in x, and is denoted as $x[i:j] \triangleq (x[i], x[i+1], \ldots, x[j])$. A substring of the form $x[1:\ell]$ is called a prefix (or an ℓ-prefix) of x and will be denoted by x_ℓ. Similarly, a substring of the form $x[|x|-\ell+1:|x|]$ is a suffix (or an ℓ-suffix) of x and will be denoted by x^ℓ. We say that strings x and y have an *overlap* of length ℓ if the ℓ-suffix of x and the ℓ-prefix of y are the same sequence; i.e. $x^\ell = y_\ell$.

In the AP, the genome s is an unknown sequence of length $|s| = G$ which we wish to assemble from a set of N reads $\mathcal{R}$. Throughout the paper, we will make two simplifying assumptions about the set of reads: (a) all reads in $\mathcal{R}$ have length L and (b) the reads in $\mathcal{R}$ are error-free. The first assumption is made mainly to simplify the exposition. The second assumption is motivated by the existence of overlapper tools (such as DAligner (Myers, 2014) and Minimap (Li, 2015)), which can efficiently identify sufficiently long overlaps between reads, and can be used as a first step to the algorithm herein described. By focusing on error-free reads, we emphasize our contribution in the context of building a sparse, but sufficient, read-overlap graph.

For ease of exposition, we will assume that s is a circular sequence of length G; i.e. $s[t+G] = s[t]$ for any t. This way we will avoid edge effects and a read $x \in \mathcal{R}$ can correspond to any substring $s[t:t+L-1]$, for $t = 1, \ldots, G$. We will use the standard probabilistic model for shotgun sequencing (based on the sampling model of Lander and Waterman, 1988). This means that each of the N reads is drawn independently and uniformly at random from the set of length-L substrings of s, $\{s[t:t+L-1] : t = 1, \ldots, G\}$.

2.1 Read-overlap graphs

A read-overlap graph $\mathcal{G} = (V, E, w, \text{st})$ is a directed weighted graph (V, E), with edge weights $w : E \to \{1, 2, \ldots\}$ (the overlaps), where each vertex $v \in V$ has an associated string $\text{st}(v)$ from the set of reads; i.e. $\mathcal{R} = \{\text{st}(v) : v \in V\}$. If $(u, v) \in E$, then u and v must overlap by $w(u, v)$ symbols; i.e. $\text{st}(u)^{w(u,v)} = \text{st}(v)_{w(u,v)}$. For simplicity, we will only consider maximal overlaps (so that for nodes u and v there can be at most one edge (u, v)). An example of a read-overlap graph is shown in Figure 3a. If $(u, v) \in E$, we will use $\text{st}(u) \oplus \text{st}(v)$ to denote the merging of $\text{st}(u)$ and $\text{st}(v)$ with an overlap of length $w(u, v)$; i.e. the concatenation of $\text{st}(u)$ and $\text{st}(v)^{L-w(u,v)}$. Moreover, if we have a path $p = (v_1, \ldots, v_k)$ in a read-overlap graph, we let $\text{st}(p) = \text{st}(v_1) \oplus \ldots \oplus \text{st}(v_k)$ be the string corresponding to path p. The assembly problem corresponds to the problem of finding a path $p = (v_1, v_2, \ldots, v_N)$ on $\mathcal{G}$ such that $\text{st}(p) = s$. Due to our assumption of a circular genome, we will be instead interested in finding a cycle $c = (v_1, v_2, \ldots, v_N, v_1)$ on $\mathcal{G}$ such that $\text{st}(c) = s$ up to cyclic shifts. Let $\mathcal{I}(v) = \{u \in V : (u, v) \in E\}$, $\mathcal{O}(v) = \{u \in V : (v, u) \in E\}$, and define $\Delta_{\text{in}}(v) = |\mathcal{I}(v)|$ and $\Delta_{\text{out}}(v) = |\mathcal{O}(v)|$ as the in-degree and the out-degree of a vertex v in $\mathcal{G}$, and $\Delta(\mathcal{G}) = \max_{v \in V} \max[\Delta_{\text{in}}(v), \Delta_{\text{out}}(v)]$.

One of the challenges of trying to identify the cycle c corresponding to s on a read-overlap graph is the fact that the graph may contain many spurious edges that are not part of c. In fact, if all overlaps of size at least 1 are included in $\mathcal{G}$, it is easy to see that we have $|E| = O(N^2)$, making even the construction of such a graph impractical, as N can be of the order of 10^5–10^8. However, the definition of $\mathcal{G}$ above does not specify *which* overlaps should in fact be included as edges. Hence, one may choose to construct a sparse version of the graph by not including edges that seem unlikely to correspond to adjacent vertices in the cycle c, such as edges corresponding to small overlaps.

2.2 String graphs

A general technique to sparsify read-overlap graphs was introduced by Myers (2005), through the paradigm of the string graph. The main idea of a string graph is to prune the read-overlap graph by removing all *transitive* edges. An edge (u, v) whose corresponding string can be equivalently obtained via the two-hop path (u, z, v) (i.e. $\text{st}(u, v) = \text{st}(u, z, v)$) is called a transitive edge, and can be removed from the graph without affecting the existence of a cycle c corresponding to the true sequence s (Fig. 3b). We point out that several long-read assemblers (Chin *et al.*, 2013; Li 2015) utilize a string graph construction.

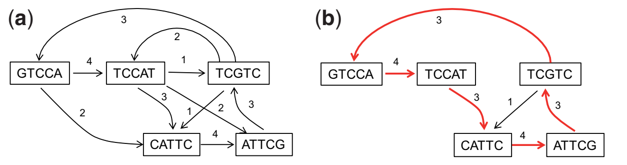

Fig. 3. (a) Read-overlap graph from five reads of length 5. The weight of each directed edge (u, v) indicates the size of the matching suffix of u and prefix of v. (b) In the string graph paradigm, transitive edges are removed from the graph, making it easier to identify a Hamiltonian cycle (red edges). Notice that some spurious edges (not part of the Hamiltonian cycle) remain on the graph after the transitive reduction

In addition to this transitive reduction (which can be performed in linear time (Myers, 2005)), the standard string graph paradigm also utilizes a minimum overlap parameter θ, and edges corresponding to overlaps smaller than θ are not included. As it turns out, θ controls an important tradeoff: a large value of θ gives rise to a sparser string graph, possibly missing many 'true' edges, while a small value of θ leads to a denser graph, possibly containing many spurious edges and consequently yielding shorter contigs. This point is illustrated in Figure 4. As we see in Figure 4b, even if we choose θ to maximize some objective such as N50, the spurious and missing edges in the string graph still cause the assembly to be fairly fragmented.

Such considerations naturally raise some questions. Is the global minimum overlap θ the right approach to control the sparsity of the string graph? Is the spurious/missing edges tradeoff fundamental or is it possible to simultaneously minimize the number of spurious edges and missing edges? As it turns out, the Not-So-Greedy algorithm, described in Section 3 constructs a sparse read-overlap graph which, under some conditions, simultaneously achieves the goals of no spurious edges and no missing edges. As shown in Figure 4b, this yields a more contiguous assembly, and a single contig is obtained with high probability at the Lander–Waterman coverage depth $N_{LW}L/G$.

2.3 The greedy algorithm

As the name suggests, the Not-So-Greedy algorithm takes inspiration on a simple greedy approach, but tries to be more flexible in its choice of edges to include in the graph. The Greedy algorithm [3] for assembly can be equivalently thought of as a rather extreme approach to constructing a sparse read-overlap graph.

As described in detail in Supplementary Material A, in this approach, for each vertex $u \in V$ we include at most one edge (u, v), chosen in a greedy fashion. However, as shown in Bresler *et al.* (2013), as long as there are repeats in s that are longer than L, a greedy approach will fail with probability at least 1/2. Therefore, as illustrated in Figure 5, for a target error probability $\epsilon < 1/2$, a greedy algorithm can only succeed if $L > \ell_{\text{repeat}}$, where ℓ_{repeat} is the length of the longest repeat in s. Hence, the region of (N, L) pairs where the algorithm succeeds is strictly suboptimal when compared to the information-theoretic lower bound. As we describe in Section 3, in order to achieve the remainder of the information-feasibility region, it suffices to consider an approach where we first consider *two* candidates for successor (and predecessor) of each node, and later prune unnecessary edges by analyzing local graph structures.

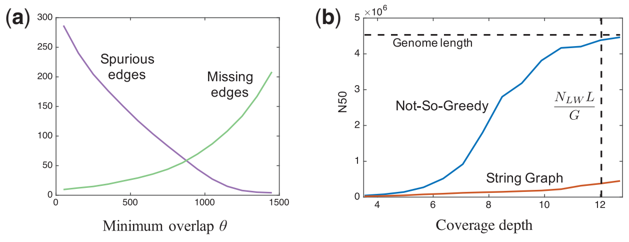

Fig. 4. (a) Number of spurious and missing edges on the string graph as a function of θ, for 10 000 error-free reads from *E. coli* K12 for $L = 3200$. (b) N50 as a function of the number reads (for $L = 3200$) obtained from string graph (using SGA (Simpson and Durbin, 2010)) with optimal choice of θ, and from the Not-So-Greedy algorithm. The N50 values were averaged over twenty realizations for each value of the number of reads

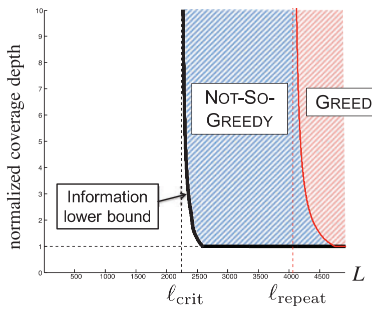

Fig. 5. For the human chromosome 19, on one hand, Greedy can only work if $L > \ell_{\text{repeat}} = 4092$. Not-So-Greedy, on the other hand, only requires $L > \ell_{\text{crit}} = 2248$, which is the information lower bound

3 Methods

The Not-So-Greedy algorithm comprises two stages. In the first one, we construct a read-overlap graph $\mathcal{G}$ where all nodes have in and out-degree at most two. While such a graph is already sparse in the sense that $|E| \le 2|V|$, it will still contain many spurious edges. Therefore, in order to further reduce it to a graph with no spurious edges, a pruning stage will follow with the goal of eliminating them. The resulting read-overlap graph, $\tilde{\mathcal{G}}$, will have the property that the true sequence corresponds to a cycle that traverses every edge at least once; i.e. a Chinese Postman cycle. Furthermore, the additional property that the cycle visits every node at most twice will allow us to further reduce the problem to an Eulerian cycle problem.

Formally, we will show that Not-So-Greedy correctly performs this reduction provided that the sequence is covered by $\mathcal{R}$ (i.e. each base is read by at least one read in $\mathcal{R}$), and certain bridging conditions are satisfied. As illustrated in Figure 6, a triple repeat A is said to be *all-bridged* if each of its copies is bridged by a read (i.e. there is a read that extends at least one base before and one base after the repeat segment). The Not-So-Greedy algorithm, which we describe in the remainder of this section, has the following guarantee.

Theorem 1

If $\mathcal{R}$ covers s and all triple repeats in s are all-bridged, Not-So-Greedy produces a read-overlap graph $\tilde{\mathcal{G}}$ containing a cycle c_s that traverses every edge and for which $\text{st}(c_s) = s$ up to cyclic shifts.

Why does this result allow us to answer question (1) in Section 1 in the affirmative? As shown by Bresler *et al.* (2013) (and summarized in supp. material G), the set

$$\mathcal{I}_s \triangleq \left\{ \mathcal{R} \left| \begin{array}{l} \mathcal{R} \text{ covers } s \\ \text{triple repeats in } s \text{ are all-bridged} \\ \text{interleaved repeats in } s \text{ are bridged} \end{array} \right. \right\} \quad (1)$$

nearly matches the set of information-feasible instances of the AP (for most genomes considered). A direct consequence of Theorem 1 is that, if $\mathcal{R} \in \mathcal{I}_s$, Not-So-Greedy builds a read-overlap graph $\tilde{\mathcal{G}}$ where s is a Chinese Postman cycle. Furthermore, in Supplementary Material F, we present a simple algorithm to modify the edge multiplicities of $\tilde{\mathcal{G}}$ producing a read-overlap (multi) graph $\hat{\mathcal{G}}$, with the following guarantee:

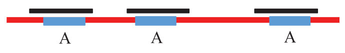

Fig. 6. A copy of a triple repeat A is bridged if there is a read that extends beyond the repeat at both ends. A triple repeat is said to be all-bridged if all of its copies is bridged

Corollary 1 *If $\mathcal{R} \in \mathcal{I}_s$, the read-overlap graph $\widehat{\mathcal{G}}$ produced by the* Not-So-Greedy *algorithm contains a unique Eulerian cycle c_s, and it satisfies* $\text{st}(c_s) = s$ *up to cyclic shifts.*

As we describe in the remainder of this section, all steps of Not-So-Greedy can be performed in $O(NL)$ time (i.e. linear in the input size). Since an Eulerian cycle can be found in linear time (if one exists), we conclude that for the instances of the AP in $\mathcal{I}_s$, one can efficiently construct a read-overlap graph and identify a cycle corresponding to s. This answers question (2) from Section 1 also in the affirmative.

3.1 Building a read-overlap graph with $\Delta(\mathcal{G})$=2

The main idea of the Not-So-Greedy construction of the read-overlap graph with $\Delta(\mathcal{G}) = 2$ is to include, for a given node u, the edge (u, v) that would be included by the Greedy algorithm and a second carefully chosen edge.

To describe this more precisely, consider the illustration in Figure 7, where the nodes v with which u has an overlap are ordered in decreasing order of $w(u, v)$, breaking ties arbitrarily. If we assume that $w(u, v_1) \geq w(u, v_2) \geq \ldots$, then the 'greedy edge' (u, v_1) is the first to be included in the graph. To choose the second out-edge of u, we keep going down on this list until we find a v_i that disagrees with the extension of v_1 beyond u; i.e. we pick the first node v_i for which $\text{st}(v_1)^{L-w(u,v_1)} \neq \text{st}(v_i)[w(u, v_i) + 1 : L + w(u, v_i) - w(u, v_1)]$. Note that this is equivalent to picking the first node v_i for which $w(v_1, v_i) < L - w(u, v_1) + w(u, v_i)$.

In the example in Figure 7, $\text{st}(v_3)$ disagrees with $\text{st}(v_1)$ on its extension beyond u, so we choose (u, v_3) as the second edge (or the Not-So-Greedy edge). We point out that this choice of the second edge can be understood in the context of the transitive reduction used to construct string graphs (Section 2.2). It is easy to see that all edges that are skipped when we look for the Not-So-Greedy edge

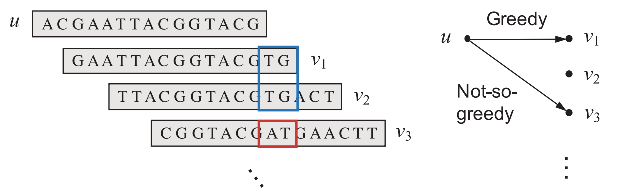

Fig. 7. For a given node u, we add two outgoing edges (u, v_1) and (u, v_i), where v_1 is the greedy match of a suffix of u, and v_i is the second longest match that does not agree with v_1

Algorithm 1. Not-So-Greedy construction of $\mathcal{G}_{\text{out}}$.

```
1: Input: $V, \text{st}, w$
2: $E \leftarrow \varnothing$
3: for $u \in V$ do
4:     % First we sort $V$ according to $w(u, v_i)$
5:     $V_{\text{sorted}} = \{v_1, \ldots, v_N\}$, where $w(u, v_1) \geq \ldots \geq w(u, v_N)$
6:     $E \leftarrow E \cup (u, v_1)$ % Choose $(u, v_1)$ as greedy edge
7:     % Go down $V_{\text{sorted}}$ to find Not-So-Greedy edge
8:     for $i = 2, \ldots, N$ do
9:         if $w(v_1, v_i) < L - w(u, v_1) + w(u, v_i)$ then
10:            $E \leftarrow E \cup (u, v_i)$
11:            break
12: Output: $\mathcal{G}_{\text{out}} = (V, E, \text{st}, w)$
```

would be transitive edges, and would have been eliminated by the transitive reduction algorithm of Myers (2005). However, by only looking for these two edges, we avoid the need to compute the entire string graph as prescribed by the standard string graph approach (Myers, 2005).

By repeating this procedure for every node, we construct a graph with the property that $\Delta_{\text{out}}(u) \leq 2$ for every node u, which we refer to as $\mathcal{G}_{\text{out}}$. This procedure is summarized in Algorithm 1, and has the following guarantee (proved in Supplementary Material B).

Lemma 1.

If $\mathcal{R}$ covers s and all triple repeats in s are all-bridged, Algorithm 1 produces a read-overlap graph $\mathcal{G}_{\text{out}}$ containing a generalized Hamiltonian cycle c_s such that $\text{st}(c_s) = s$.

Lemma 1 implies that $\mathcal{G}_{\text{out}}$ contains a cycle corresponding to the true sequence, and it is clear by construction that $\Delta_{\text{out}}(u) \leq 2$ for every node u. However, nodes in $\mathcal{G}_{\text{out}}$ may have in-degree larger than 2. In order to fix that, we let E_{out} be the set of edges produced by Algorithm 1, and we consider repeating Algorithm 1 modified in a straightforward way to produce a second read-overlap graph $\mathcal{G}_{\text{in}} = (V, E_{\text{in}}, \text{st}, w)$ with $\Delta_{\text{in}}(u) \leq 2$ for every $u \in V$ and for which Lemma 1 also holds by symmetry. Finally, we construct the read-overlap graph $\mathcal{G} = (V, E, \text{st}, w)$, by setting $E = E_{\text{out}} \cap E_{\text{in}}$. It is clear that $\Delta(\mathcal{G}) \leq 2$, and it is easy to check that Lemma 1 still holds after the intersection.

We point out that the construction procedure in Algorithm 1 is mainly for description purposes. In supp. material D, we introduce an approach to find overlaps efficiently based on rolling hashes (Karp and Rabin, 1987), which allows us to efficiently identify overlaps between reads in decreasing order of overlap size. Therefore, we do not need to perform a sorting of overlaps for every read as in line 5. This approach, which we refer to as decremental hashing (in analogy to the incremental hashing of Ben-Bassat and Chor (2014)), yields a $O(NL)$ construction (for error-free reads), summarized in supp. material D.

3.2 Pruning the read-overlap graph

Next, we describe the second part of the Not-So-Greedy algorithm, which attempts to prune $\mathcal{G}$ in order to reduce the number of spurious edges. The fact that this graph has $\Delta(\mathcal{G}) \leq 2$ will allow us to utilize the natural partition of the edge set into *zig-zag* components in order to identify edges that should be removed.

The pruning algorithm starts by preprocessing the read-overlap graph $\mathcal{G}$ for the edge pruning operations by computing an *effective overlap* function $\tilde{w} : E \to \{1, 2, \ldots\}$. The idea is to check when the string $\text{st}(u)$ can be extended to the right or to the left in an unambiguous way. For example, suppose $(u, v_1) \in E$ and $(u, v_2) \in E$, and the corresponding reads are as shown in Figure 8. Then, read $\text{st}(u)$ must be followed by A, G, G in s, and we can imagine that $\text{st}(u)$ is virtually extended to the right by three positions. The effective overlaps can then be defined as $\tilde{w}(u, v_1) = w(u, v_1) + 3$ and

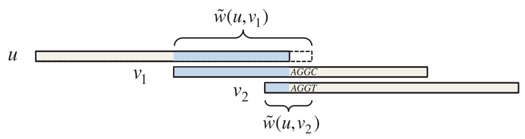

Fig. 8. In this example, $\text{st}(u)$ can be virtually extended to the right in an unambiguous manner by three symbols and we have the effective overlap values $\tilde{w}(u, v_i) = w(u, v_i) + 3$ for $i = 1, 2$

$\tilde{w}(u, v_2) = w(u, v_2) + 3$. Read st($u$) can be virtually extended to the left similarly, by considering the two incoming edges (z_1, u) and (z_2, u). We will set $\tilde{w}(u, v) = \infty$ whenever $\Delta_{\text{out}}(u) = 1$ or $\Delta_{\text{in}}(v) = 1$. This will prevent the algorithm from ever deleting edge (u, v). A more detailed description of the computation of the effective overlaps is presented in Supplementary Material E. Notice that we do not actually modify the mapping st($\cdot$), and the read extensions are only a way to visualize the computation of effective overlaps $\tilde{w}(\cdot,\cdot)$.

Once all effective overlaps are computed, we move to the pruning phase, described in Algorithm 2. In this part, the actual string associated with each read is no longer needed, and the algorithm works solely on the graph (V, E) and the effective overlaps $\tilde{w}(\cdot,\cdot)$. The algorithm will keep at all times a graph (V, E) with the property that $1 \leq \Delta_{\text{in}}(v), \Delta_{\text{out}}(v) \leq 2$ for every vertex $v \in V$. At each iteration, our goal is to reduce the size of the subset of edges

$$U = \{(u, v) : \Delta_{\text{out}}(u) = \Delta_{\text{in}}(v) = 2\}, \quad (2)$$

and the algorithm stops when $U = \emptyset$. The set U can be thought of as the set of edges in E whose presence in the cycle c_s corresponding to the true sequence s is unknown. More precisely, any edge $(u, v) \notin U$ must have $\Delta_{\text{out}}(u) = 1$ or $\Delta_{\text{in}}(v) = 1$ and if the read-overlap graph maintained by the algorithm contains a cycle c_s that visits every node, c_s must traverse (u, v). Notice also that any edge $(u, v) \notin U$ will have $\tilde{w}(u, v) = \infty$.

For an edge $(u_2, v_1) \in U$, let hd(u_2, v_1) be the (unique) edge sharing the same head vertex as (u_2, v_1), and tl(u_2, v_1) be the (unique) edge sharing the same tail vertex as (u_2, v_1), as shown in Figure 9a. Algorithm 2 starts by identifying the edges in U (i.e. those with $\tilde{w}(u, v) \neq \infty$) and then sorting them in increasing order of effective overlap value $\tilde{w}(\cdot,\cdot)$. If during the sorting two edges $(u_1, v_1), (u_2, v_2) \in U$ are found to have $\tilde{w}(u_1, v_1) = \tilde{w}(u_2, v_2)$, and we have $\tilde{w}(u_1, v_1) = \tilde{w}(\text{hd}(u_1, v_1)) = \tilde{w}(\text{tl}(u_1, v_1))$, then ($u_2$, v_2) is placed before (u_1, v_1) in the sorting. Ties are broken arbitrarily otherwise.

The algorithm operates by taking the first (u_2, v_1) from the sorted U and considering the 'zig-zag' structure formed by edges $(u_1, v_1) = \text{hd}(u_2, v_1)$, $(u_2, v_2) = \text{tl}(u_2, v_1)$, which must exist by the definition of U and be unique by the bounded-degree property. At this point, the algorithm considers two cases to decide which transformation to make to the graph. If $(u_1, v_2) \in E$, we have the 'hourglass' structure. If in addition all four edges $(u_1, v_1), (u_1, v_2), (u_2, v_1), (u_2, v_2)$ have the same effective overlap value $\tilde{w}(\cdot,\cdot)$, then the algorithm adds a virtual read that creates an X-node in the read-overlap graph, as illustrated in Figure 9. Otherwise, the algorithm deletes (u_2, v_1). The same pruning procedure is applied iteratively, until $U = \emptyset$.

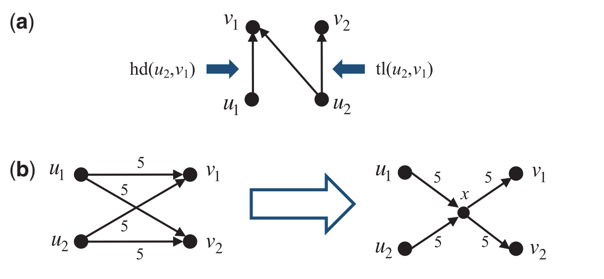

Fig. 9. (a) Zig-zag structure formed by an edge $(u_2, v_1) \in U$. (b) If an hourglass structure $(u_1, v_1), (u_1, v_2), (u_2, v_1), (u_2, v_2)$ with $\tilde{w}(u_1, v_1) = \tilde{w}(u_1, v_2) = \tilde{w}(u_2, v_1) = \tilde{w}(u_2, v_2)$ is found, we create a vertex x whose string st(x) corresponds to the effective overlap between u_i and v_j for $i, j \in \{1, 2\}$ (they are all the same), delete edges (u_i, v_j) for $i, j \in \{1, 2\}$ and add edges $(u_1, x), (u_2, x), (x, v_1), (x, v_2)$ to E

Algorithm 2. Not-So-Greedy pruning phase.

```
1: Input: $\mathcal{G} = (V, E, \text{st}, w)$ with $\Delta(\mathcal{G}) = 2$
2: Compute effective overlaps $\tilde{w}(\cdot,\cdot)$
3: $U \leftarrow \{(u, v) : \Delta_{\text{out}}(u) = \Delta_{\text{in}}(v) = 2\}$
4: Sort $U$ according to $\tilde{w}(u, v)$
5: while $U \neq \emptyset$ do
6:    $(u_2, v_1) \leftarrow$ first edge in $U$
7:    $(u_1, v_1) \leftarrow \text{hd}(u_2, v_1)$
8:    $(u_2, v_2) \leftarrow \text{tl}(u_2, v_1)$
9:    if $(u_1, v_2) \in U$ and $\tilde{w}(u_1, v_1) = \tilde{w}(u_2, v_1) = \tilde{w}(v_1, v_2) = \tilde{w}(u_1, v_2)$ then
10:        Create new node $x$
11:        Perform transformation in Figure 9b
12:    else
13:        Delete edge ($u_2$, $v_1$)
14:    Update set $U$ and weights $\tilde{w}(\cdot,\cdot)$
15: Output: $\tilde{\mathcal{G}} = (V, E, \text{st}, w)$
```

This iterative pruning procedure is summarized in Algorithm 2. As described in Supplementary Material E, computing the effective overlaps for each edge $(u, v) \in E$ has a running time of $O(|E|) = O(N)$, since $|E| \leq 2N$. Algorithm 2 first sorts the edges in U, which has a time complexity of $O(N \log N)$, and performs $O(|E|) = O(N)$ pruning iterations, each of which can be done in constant time. Finally, we notice that since the reads in $\mathcal{R}$ are assumed to be all distinct from each other (which can be achieved by a preprocessing step), we must have $L \geq \log_4 N$, and we conclude that the total running time for the Not-So-Greedy algorithm is $O(NL)$.

Theorem 1, which we formally prove in Supplementary Material C, states that provided that $\mathcal{R}$ covers the sequence s and triple repeats are all-bridged, the graph $\tilde{\mathcal{G}}$ produced by Not-So-Greedy contains a Chinese Postman cycle c_s corresponding to the true sequence s. The problem of finding the shortest such cycle, the Chinese Postman Problem, can be solved in polynomial time (with a running time of $O(|V|^2|E|) = O(N^3)$ (Edmonds and Johnson, 1973)). However, when triple repeats are all bridged, the cycle c_s has the additional property that it cannot visit any node (and, consequently, any edge) more than twice. As it turns out, this allows us to reduce the problem further to an instance of the Eulerian Cycle Problem, which can be solved in linear time. This reduction (which leads to the proof of Corollary 1) is described in Supplementary Material F.

4 Results

While the Not-So-Greedy algorithm presented in Section 3 is motivated by the goal of perfect assembly, it can be seen as a general method for constructing a sparse read-overlap graph. Notice that when the assumptions from Theorem 1 and Corollary 1 are not satisfied, extracting a single contig from the graph may not be possible. Hence, in a practical scenario where these assumptions are not known to be satisfied, once the graph $\tilde{\mathcal{G}}$ is constructed by Algorithm 2, instead of using the final Eulerian reduction step in Supplementary Material F, one should identify unambiguous paths and extract them as contigs. This approach allows us to evaluate Not-So-Greedy in a more general setting.

In this section, we take this approach and describe two sets of results. First, in Section 4.1 we consider directly applying the algorithm described in this paper to reads simulated from several real genomes. Then, in Section 4.2, we consider applying these ideas to a real long-read dataset.

4.1 Simulated reads from real genomes

In order to evaluate Not-So-Greedy, we implemented the algorithm and studied its performance under the sampling model described in Section 2. More precisely, we considered sampling error-free reads of a fixed length and uniformly at random from real genomes (available at www.ncbi.nlm.nih.gov). We then compared the performance of Not-So-Greedy and the performance of the standard string graph approach. This approach allows us to study the performance of Not-So-Greedy across many independent experiments and a wide variety of coverage depths, and emphasize the conceptual advantages of a sparse read-overlap graph. The implementation of Not-So-Greedy used in these experiments is available at github.com/samhykim/nsg, and the standard string graph construction was performed using SGA (Simpson and Durbin, 2010) and the Incremental Hashing assembler (Ben-Bassat and Chor, 2014). In order to handle the fact that reads can come from both strands of the genome, before running Not-So-Greedy, we preprocess the set of reads to include each read and its reverse complement.

In Figure 10, we analyze the read-overlap graphs produced by the Not-So-Greedy algorithm and by a string graph assembly algorithm (Ben-Bassat and Chor, 2014). The read-overlap graph is evaluated in terms of the number of missing edges and spurious edges. In order to compute these values, we utilized the reference genome to produce the 'perfect' read-overlap graph (where only reads that are adjacent in the genome have an edge between them) and compared it with the read-overlap produced by the different algorithms. As mentioned in Section 2.2, the minimum overlap θ controls a tradeoff between the number of missing and spurious edges in the string graph. We chose θ for the string graph assembler in a 'genie-aided' fashion so that the number of missing edges and spurious edges are the same.

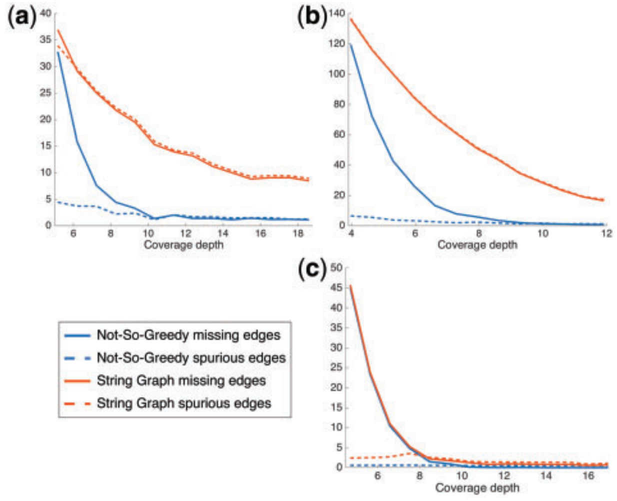

Fig. 10. Spurious edges and missing edges in the read-overlap graph produced by Not-So-Greedy and a string graph assembler (Ben-Bassat and Chor, 2014) as a function of the coverage depth $c = NL/G$ for (a) *S. aureus*, (b) *E. coli* K12, (c) *R. sphaeroides* (chromosome 1), with reads of length $L = 3000$. The number of spurious edges and missing edges were averaged over twenty realizations of the read set

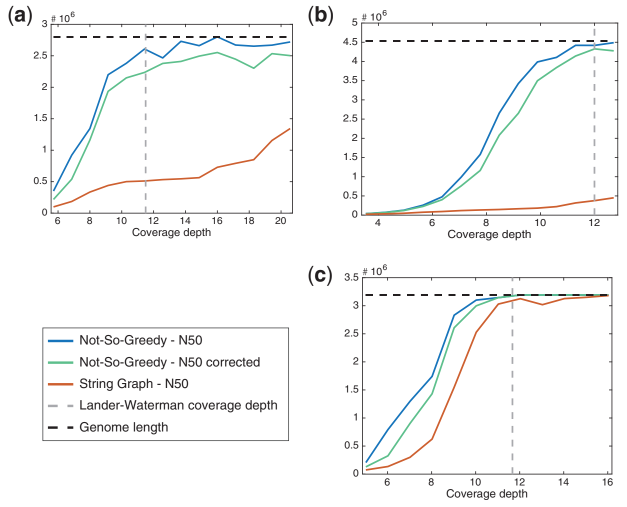

Fig. 11. N50 as a function of the coverage depth for (a) *S. aureus*, (b) *E. coli* K12, (c) *R. sphaeroides* (chromosome 1). We set $L = 3200$ so that L is at least 10% greater than ℓ_{crit} for all three genomes. N50 values were averaged over 40 realizations for each point

Intuitively speaking, the 'shotgun' reading process causes the overlap between pairs of reads that are consecutive on the genome to span a wide range of values. Because of that, trying to sparsify the graph via the global parameter θ is suboptimal. On the contrary, Not-So-Greedy chooses edges in a local fashion, giving priority to longer matches. As a result, we see in Figure 10 that the graph produced by Not-So-Greedy has fewer spurious edges and missing edges across all coverage depths and genomes shown. In fact, for the higher values of coverage depth, as predicted by Theorem 1, the graph produced by Not-So-Greedy often has zero missing edges and spurious edges (notice that the values shown in Figure 10 are averaged over 20 realizations of the read sets). We also notice that for the *Rhodobacter sphaeroides* genome, which is not very repetitive, the performance of both algorithms is very similar.

It is important to point out that the number of missing and spurious edges for both approaches are small in comparison with the total number of edges. For instance, at coverage depth $c = 10$ for *Staphylococcus aureus*, the 'perfect' read-overlap graph contains roughly $N = cG/L \approx 10\,000$ edges. This means that the spurious or the missing edges represent only about 0.015% of the total number of edges. However, even such a small number of incorrect edges on the graph can significantly reduce the quality of the assembly. This is demonstrated in Figure 11, where we show the N50 values obtained from the read-overlap graphs produced by different algorithms.

We see that the (average) N50 values achieved by Not-So-Greedy converge to the genome length as the coverage depth increases much quicker than the N50 values achieved by a string graph approach. We point out that for the experiment of Figure 11, in contrast to the experiment of Figure 10, we evaluated the performance of the string graph using SGA (Simpson and Durbin, 2010) instead of the assembler by Ben-Bassat and Chor (2014) (because computing N50 from the SGA output is easier), and we chose the minimum overlap threshold θ to maximize N50. In order to verify that the higher N50 values of Not-So-Greedy are not created by missassemblies, in Figure 11, we also display the value of N50 corrected.

4.2 PacBio *Escherichia coli* K12 dataset

In order to apply the algorithmic ideas designed under the idealized model from Section 2 on actual long-read sequencing datasets, important modifications are required. In particular, in order to deal with the read errors, we consider using DAligner (Myers, 2014) to identify approximate overlaps between reads. These overlaps can then be treated as 'correct' matches and used for the construction of a sparse read-overlap graph. In addition, the assumption in Section 2 of equal length reads can be relaxed to the case where all reads are maximal (i.e. no read is contained in another read) in a straightforward manner. This can be attained in practice by filtering contained reads.

Nonetheless, long-read datasets present additional undesirable features such as chimeric reads, and it is in general unknown whether the information requirements for perfect assembly have been met. Hence, a major focus of our ongoing research is on developing a practical assembler that is robust to these non-idealities. In this section, we present initial results (obtained in collaboration with Kamath *et al.*, 2016) to support the applicability of our techniques on real data. These results were obtained by using DAligner (Myers, 2014) to identify the overlaps between the reads in a PacBio *E. coli* K12 dataset (github.com/pacificbiosciences/devnet/wiki/datasets) and applying a modified algorithm NOT-SO-GREEDY′ to build a sparse read-overlap graph.

In Figure 12, we show the information feasibility region for the *E. coli* reference provided by PacBio. By considering the normalized coverage depth and average read length of the PacBio data, we see that this dataset contains enough information for perfect reconstruction, and in theory a single contig should be obtained by the NOT-SO-GREEDY algorithm. Indeed, the graph constructed by NOT-SO-GREEDY′ is as shown in Figure 13a. Since for each read we add two nodes to the graph, one for each orientation, the resulting graph contains two rings, each corresponding to a cyclic contig of length 4.5 Mb. To verify that there are no misassemblies, we mapped the reads onto the reference provided by PacBio and colored the nodes according to the mapping position. The smooth gradient along the two rings in the graph supports the correctness of the assembly.

Next we considered modifying this dataset so that it no longer lies within the feasibility region in Figure 12. To do so, we randomly broke each read into two parts (the first one normally distributed with mean 6 kb and standard deviation 1.5 kb). The resulting dataset has average read length 3.5 kb and does not bridge all triple repeats in the *E. coli* K12 genome. Hence, it is not possible to perfectly assemble the genome, and NOT-SO-GREEDY′ produces a sparse read-overlap graph which, although Eulerian, does not uniquely determine the entire genome, as shown in Figure 13b. Notice that the existence of an unbridged inverted repeat causes the two strands to merge. The colors of the nodes again indicate the location of the read on the genome and show that all contigs are correctly assembled.

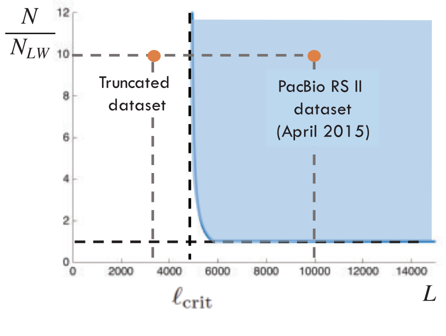

Fig. 12. The *E. coli* K12 dataset considered lies well within the information feasibility region, deeming the task of perfect assembly possible. By truncating the reads, we obtained a second dataset, from which perfect assembly is not possible

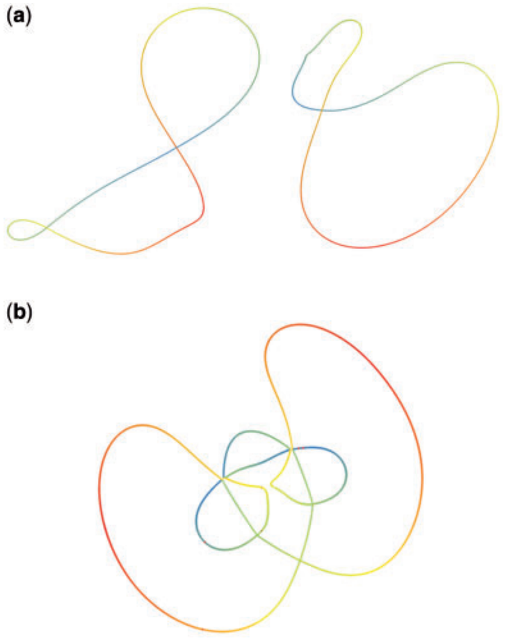

Fig. 13. (a) Assembly graph obtained from the PacBio *E. coli* K12 dataset (visualized with Gephi (Mathieu *et al.*, 2009)). Nodes are colored according to the relative position of the read in the genome, obtained by mapping the reads onto the *E. coli* K12 reference genome provided by PacBio. (b) Assembly graph obtained after truncating the reads in this dataset to an average length of 3.5 kb

Table 1. Comparison with Miniasm for the truncated *E. coli* K12 dataset. Notice that NOT-SO-GREEDY′ produces a double-stranded graph, while Miniasm produces a single-stranded (bidirected) graph

	Longest contig	# of contigs	N50
NOT-SO-GREEDY′	2.9 Mb	28 (two strands)	2.5 Mb
Miniasm	670 kb	32 (one strand)	450 kb

As a benchmark, we compared our performance to that of Miniasm (Li, 2015). For the original dataset, perfect assembly is also attained by Miniasm, while the results for the truncated dataset are shown in Table 1. We emphasize that these results are provided merely as proof-of-concept, and a thorough evaluation of our assembler on different datasets is part of our ongoing work (Kamath *et al.*, 2016).

5 Discussion

In this work, we introduced a new algorithm for the construction and subsequent pruning of read-overlap graphs. Unlike most other approaches, the NOT-SO-GREEDY algorithm takes inspiration from information-theoretic considerations about the AP. More precisely, the algorithm is tailored to succeed in instances of the problem where the set of reads $\mathcal{R}$ contains enough information for unambiguous reconstruction. As it turns out, in such instances of the problem, the read length and the coverage depth must be large enough so that the remaining ambiguity due to repeats can be resolved via a careful construction of a sparse read-overlap graph, which can be performed efficiently. This allows us to avoid the usual NP-hardness of optimization-based formulations of the AP (which is largely due to information limitations) and establish that most instances of the problem that are information-feasible are also efficiently solvable.

In addition to the implications on our theoretical understanding of the AP, the NOT-SO-GREEDY algorithm can be understood as providing a technique to build sparser read-overlap graphs than those produced by the standard string graph approach. As shown in Section 4, even when the bridging conditions assumed in Theorem 1 are not satisfied, most of the edges that are removed by the NOT-SO-GREEDY algorithm are indeed spurious edges. This can lead to assemblies that are more contiguous (and achieve higher N50 scores) than those obtained via a standard string graph approach. The natural direction for follow-up work, and our current research focus, is the translation of these ideas into a practical assembler. While the preliminary results in Section 4.2 are promising, significant work is required to produce an assembler that is robust to artifacts present in long-read data, and that can handle genomes with complex repeat patterns (such as telomeric regions).

The theoretical framework utilized in this work also suggests interesting directions for further investigation. In particular, understanding the fundamental limits of *partial* assembly (i.e. when reads do not have enough information for perfect assembly) is of significant practical importance. This is another focus of our current work (Shomorony *et al.* 2016). The development of a theoretical framework to understand the problem of *hybrid* assembly would also be of interest. In practice, hybrid datasets (i.e. with both short and long reads) are usually assembled in a decoupled fashion, by first using short reads to correct the long reads, and then assembling the cleaned-up long reads (Koren *et al.*, 2012). However, it is unclear whether such an approach is optimal from an information-theoretic standpoint.

Another direction for future work, from a more theoretical standpoint, is understanding whether, in information-feasible instances of the AP, the output of NOT-SO-GREEDY coincides with the solution of a combinatorial optimization problem. Notice that while Theorem 1 guarantees the reconstruction of the true sequence s, there is no guarantee that this sequence corresponds to the solution of an optimization-based formulation of the AP such as those considered by Nagarajan and Pop (2009) and Medvedev and Brudno (2009). As mentioned by Medvedev and Brudno (2009), parsimony-based formulations tend to encourage an over-collapsing of the repeats, and the optimal solution is in general different from the true underlying sequence. The maximum-likelihood formulation of the AP (Medvedev and Brudno, 2009), on the contrary, seems to be robust to these issues, and thus a good candidate for the 'correct' formulation. Understanding whether bridging conditions can be used to guarantee that the maximum-likelihood sequence is the true sequence is currently an open question.

Acknowledgements

The authors would like to thank the anonymous reviewers for helpful comments and suggestions. Part of this work was completed while the authors were visiting the Simons Institute for the Theory of Computing, UC, Berkeley.

Funding

This work was partially supported by the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370 and also by a Hellman Fellowship.

Conflict of Interest: none declared.

References

Bastian,M. *et al.* (2009) Gephi: an open source software for exploring and manipulating networks.

Berlin,K. *et al.* (2015) Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature Biotech*, **33**, 623–630

Ben-Bassat, I. and Chor, B. (2014) String graph construction using incremental hashing. *Bioinformatics*, **30**, 3515–3523.

Blum,A. *et al.* (1994) Linear approximation of shortest superstrings. *JACM*, **41**, 630–647.

Bresler,G. *et al.* (2013) Optimal assembly for high throughput shotgun sequencing. *BMC Bioinformatics*, **14**, S18,

Chin,C.-S. *et al.* (2013) Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nat. Methods*, **10**, 563–569.

Edmonds,J. and Johnson,E.L. (1973) Matching, euler tours and the Chinese postman. *Math. Prog.*, **5**, 88–124.

Kamath,G. *et al.* (2016) HINGE: Long-Read Assembly Achieves Optimal Repeat Resolution. Preprint: BioRxiv 062117.

Karp,R.M. and Rabin,M.O. (1987) Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, **31**, 249–260.

Koren,S. *et al.* (2012) Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nat. Biotechnol.*, **30**, 693–700.

Lander,E.S. and Waterman,M.S. (1988) Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, **2**, 231–239.

Li,M. (1990) Towards a DNA sequencing theory (learning a string). In: *Proceedings of the Foundations of Computer Science*. New York: IEEE, pp. 125–134.

Li,H. (2015) Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *arXiv Preprint arXiv*, 1512.01801

Lin,Y. *et al.* (2016) Assembly of long error-prone reads using de Bruijn graphs. *bioRxiv*, [Epub ahead of print, doi:10.1101/048413].

Medvedev,P. *et al.* (2007) Computability of models for sequence assembly. In *Algorithms in Bioinformatics*. New York: Springer, pp. 289–301.

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

Myers,E.W. (2014) Efficient local alignment discovery amongst noisy long reads. In: *Algorithms in Bioinformatics*. Berlin: Springer, pp. 52–67.

Myers,E.W. (2005) The fragment assembly string graph. *Bioinformatics*, **21**, ii79–85

Nagarajan,N. and Pop,M. (2009) Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *J. Comput. Biol.*, **16**, 897–908.

National Center for Biotechnology Information (2015) (www.ncbi.nlm.nih.gov).

Pacbio,E. *coli* K12 dataset (2015) (https://github.com/pacificbiosciences/devnet/wiki/e.-coli-bacterial-assembly).

Peng,Y. *et al.* (2010) Idba – a practical iterative de bruijn graph de novo assembler. *In Research in Computational Molecular Biology*. Berlin: Springer, pp. 426–440.

Pevzner,P. (1995) DNA physical mapping and alternating Eulerian cycles in colored graphs. *Algorithmica*, **13**, 77–105.

Pevzner,P.A. *et al.* (2001) An eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci.*, **98**, 9748–9753.

Shomorony,I. *et al.* (2016) Partial DNA assembly: a rate-distortion perspective. In: *Proceedings of the International Symposium on Information Theory*.

Simpson,J.T. and Durbin,R. (2010) Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, **26**, 367–373.

Tarhio,J. and Ukkonen,E. (1988) A greedy approximation algorithm for constructing shortest common superstrings. *Theor. Comput. Sci.*, **57**, 131–145.

Zerbino,D.R. and Birney,E. (2008) Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res.*, **18**, 821–829.