On Algorithmic Complexity of Biomolecular Sequence Assembly Problem

Giuseppe Narzisi[1], Bud Mishra[1,2], and Michael C Schatz[1]

[1]Simons Center for Quantitative Biology, One Bungtown Road, Cold Spring Harbor Laboratory, NY, USA, 11724
[2]Courant Institute of Mathematical Sciences, New York University, New York, NY, USA, 10012
{gnarzisi,mschatz}@cshl.edu, mishra@nyu.edu

Abstract. Because of its connection to the well-known $\mathcal{NP}$-complete shortest superstring combinatorial optimization problem, the Sequence Assembly Problem (SAP) has been formulated in simple and sometimes unrealistic string and graph-theoretic frameworks. This paper revisits this problem by re-examining the relationship between the most common formulations of the SAP and their computational tractability under different theoretical frameworks. For each formulation we show examples of logically-consistent candidate solutions which are nevertheless unfeasible in the context of the underlying biological problem. This material is hoped to be valuable to theoreticians as they develop new formulations of SAP as well as of guidance to developers of new pipelines and algorithms for sequence assembly and variant detection.

Keywords: Genome Assembly, Sequence Assembly Problem, Optimality, $\mathcal{NP}$-complete Problem.

1 Introduction

The ability to sequence a genome and reconstruct its DNA sequence is changing human genetics research [14]. Recent advances in DNA sequencing technology have driven the cost of sequencing a complete human genome to below $1000 US[1], and the potential applications to biology and medicine have rekindled enormous interest in several classical algorithmic problems at the core of genomics and computational biology, especially the DNA sequence assembly problem (SAP). Two decades back, in the context of the Human Genome Project, the problem had received unprecedented scientific prominence: its computational complexity and intractability were thought to have been well understood; various competitive heuristics, thoroughly explored and the necessary software, properly implemented and validated. However, recent studies on the experimental validation of de novo assemblers, have highlighted several limitations [19, 4, 2].

The process of reducing/relating the problem of reconstructing the genome sequence into a well-defined computer science problem is not straightforward:

[1] http://dx.doi.org/doi:10.1038/nature.2014.14530

for instance, limited or incomplete knowledge of the original biological problem, can lead to erroneous formulations. Consequently, a perfectly well-defined "optimal solution" in the computational setting may turn out to be irrelevant, infeasible or incorrect, when translated back to the original biological setting. The sequence assembly problem is in fact a *wicked*[2] problem: incomplete, contradictory, changing requirements (e.g., genome structure) lead to incomplete and biologically incorrect formulations.

This paper carefully examines the most popular formulations for SAP over the last 20 years. Each formulation is rigorously defined. Similarity and differences among paradigms are explained, demonstrating a strong connection between the different formalisms. More importantly, we present examples of logically consistent solutions in each of this formulations which are intractable or unfeasible in the context of biology.

2 The dovetail-path framework

The *dovetail-path framework* was first introduced by Myers in [16]. The output of a sequencing project consists of a set of reads $F = \{r_1, r_2, \ldots, r_N\}$, where each read r_i is a string over the alphabet $\Sigma = \{A, C, G, T\}$. Each read is associated a pair of integers $(s_i, e_i), i \in [1, |F|]$ where s_i and e_i are respectively the starting and ending points of the read r_i in the reconstructed string R (to be computed by the assembler), such that $1 \leq s_i, e_i \leq |R|$. The order of s_i and e_i encodes the orientation of the read (whether r_i was sampled from Watson or Crick strand of the DNA molecule).

The overlaps between pairs of reads capture where the suffix of the first matches the prefix of the second within some maximum error rate, and may be computed using the Smith-Waterman algorithm [24] with match, mismatch and gap penalty scores dependent on the error model of the sequencing technology. Thanks to the high throughput of next-generation sequencing technology, overlaps computed using exact-match are now adequately informative for short reads, although emerging third-generation long read sequencing requires in-exact matching algorithms [11, 23]. The complete description of an overlap π is given by specifying:

1. the substrings $\pi.A[\pi.s_A, \pi.e_A]$ and $\pi.B[\pi.s_B, \pi.e_B]$ of the two reads that are involved in the overlap;
2. the offsets from the left-most and right-most positions of the reads $\pi.A_{hang}$ and $\pi.B_{hang}$;
3. the relative directions of the two reads: Normal (N), Innie (I);
4. a binary predicate $\mathit{suffix}_\pi(r)$ on a read r such that:

$$\mathit{suffix}_\pi(r) = \begin{cases} true \quad \mathit{iff} \text{ suffix of } r \text{ participates in the overlap } \pi \\ false \quad \mathit{iff} \text{ prefix of } r \text{ participates in the overlap } \pi \end{cases} \tag{1}$$

[2] A problem is wicked, if from its original formulation, one is led to a "correct" solution that reveals the incorrectness, incompleteness or inconsistencies in the formulation of the problem [22].

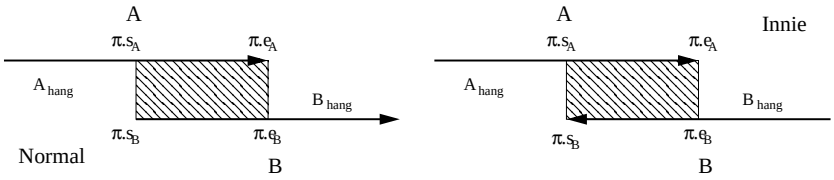

Fig. 1. Two possible overlaps (illustration): left overlap is *normal* (with both reads pointing to the same direction) right overlap is *innie* (with both reads pointing against each other); The suffix predicate for the left (normal) overlap is s.t. $\mathit{suffix}_\pi(A) = true$ and $\mathit{suffix}_\pi(B) = false$

Figure 1 illustrates two possible overlaps. Because of the double-stranded nature of the DNA molecule, each read can be sampled from either the Watson or Crick strands and they have different orientation.

Definition 1 (Layout). *The layout L associated to a set of reads F is defined as:*

$$L_F = r_1 \overset{\pi_1}{\rightleftharpoons} r_2 \overset{\pi_2}{\rightleftharpoons} r_3 \overset{\pi_3}{\rightleftharpoons} \ldots \overset{\pi_{N-1}}{\rightleftharpoons} r_N \tag{2}$$

Informally a layout is simply a sequence of reads with each neighboring read pair connected by overlap relations. The previous definition assumes that there are no *containments*[3]; without any loss of correctness or generality, contained reads can be initially removed (in a preprocessing step) and then reintroduced later after the layout has been created. Among all the possible layouts (possibly, exponential in the number of reads), it is imperative to efficiently identify the ones that are consistent according to the following definition:

Definition 2 (Consistency Property). *A layout L is* **consistent** *if the following property holds for $i = 2, \ldots, N-1$:*

$$\overset{\pi_{i-1}}{\rightleftharpoons} r_i \overset{\pi_i}{\rightleftharpoons} \text{iff } \mathit{suffix}_{\pi_{i-1}}(r_i) \neq \mathit{suffix}_{\pi_i}(r_i) \tag{3}$$

The consistency property imposes a directionality for traversing the sequence of reads in the layout. The directionality of each internal read in the layout must be preserved so that the left and right overlaps have opposite values for the *suffix* predicate. Figure 2 shows an example of layout arising from 7 overlapping reads.

Appealing to parsimony, we are typically interested in a layout whose length is minimal (although we will see that this assumption is biologically incorrect). The following theorem shows the correlation between the length of a layout and the sizes of its overlaps. Let us define the weight of a layout L to be the sum of the lengths of its overlaps, $\mathit{weight}(L) = \sum_{\pi \in L} \mathit{length}(\pi)$, then the following theorem holds [25, 26]:

Theorem 3 (Min-length reconstruction). *A layout of maximum weight results in a reconstruction of minimum length.*

[3] Reads that are proper subsequences of another read.

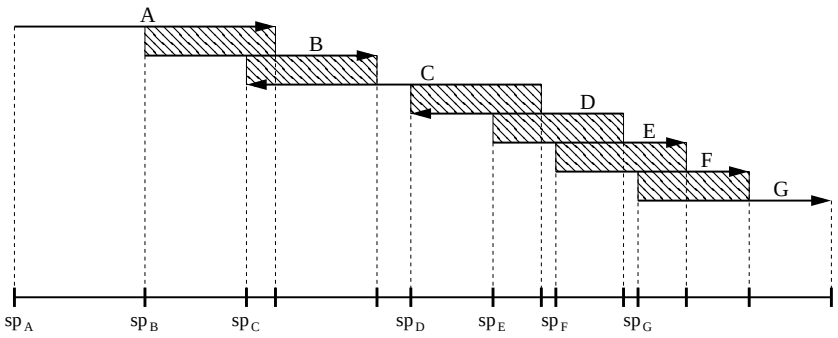

Fig. 2. Example of layout for a set of fragments $F = \{A, B, C, D, E, F, G\}$ with sequence of overlaps $\pi^N_{(A,B)}, \pi^I_{(B,C)}, \pi^N_{(C,D)}, \pi^I_{(D,E)}, \pi^N_{(E,F)}, \pi^N_{(F,G)}$

3 Shortest Superstring Problem (*SSP*)

Researchers first approximated the shotgun sequence assembly problem as one of finding the shortest common superstring of a set of sequences. This formulation was encouraged by the results of the previous theorem and the growing body of literature on efficient algorithms to solve the SSP.

Definition 4 (Shortest Superstring Problem). *Given a set of strings* $S = \{r_1, r_2, \ldots, r_n\}$ *find the shortest string* R *(reconstruction) such that* $\forall i$, r_i *is a substring of* R.

This formulation led to a simple theoretical abstraction, but by being oblivious to how biological sequences are organized by evolution, it often yielded biologically implausible and incorrect solutions. Its inability to correctly model the assembly problem is owed to a multitude of reasons, but primarily because:

1. the shortest-superstring formulation does not account for possible errors arising during the process of sequencing the fragments,
2. it does not model fragment orientation (the sequence source can be one of the two DNA strands), and
3. most importantly, it fails in the presence of *repeats*, as it encourages repeat-induced compressions.

Elaborating on the last point it is of interest to consider Richard Karp's statement in 2003 [9]: *The shortest superstring problem [is an] an elegant but flawed abstraction: [since it defines assembly problem as finding] a shortest string containing a set of given strings as substrings.* Figure 3 shows an example of the kind of errors that such formulation could lead to. Since strings contained inside a repeat regions cannot be disambiguated, multiple copies of a repeat are compressed into a single one.

Because of the theoretical computational intractability ($\mathcal{NP}$-completeness [5]) of the SSP, most of the approaches for genome sequence assembly have

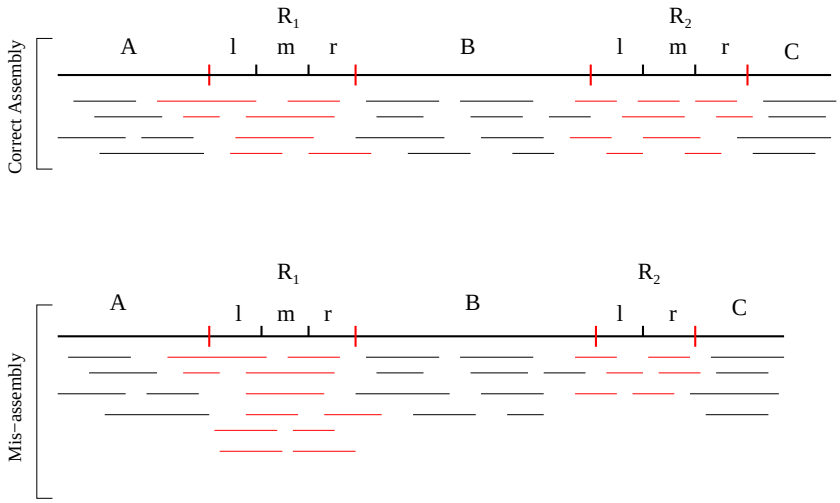

Fig. 3. Example of compression: the two copies of repeat (R_1 and R_2) are compressed into one leading to a shorter but misassembled sequence

resorted to greedy and heuristic methods that, by definition, restrict themselves to near-optimal solutions, where the "nearness" may be guaranteed within a multiplicative competitiveness factor. The best known greedy algorithm for the SSP has an approximation factor of $2\frac{2}{3}$ [1].

4 Graph-Theoretic formulation

Differently from string-based approaches, graph-theoretic formulations convert sequence assembly into solving specific problems for general graphs constructed using the overlap information of the input set of reads. This mapping has the advantage of allowing us to apply the large collection of algorithms and heuristics that have been developed in graph theory for many decades. However, this formulation still fails to completely cure the problems and limitations of the SSP model, since it can produce mis-assembly errors (as shown later). In this section we introduce the two most used graphical models for the sequence assembly problem: *string graph* and *De Bruijn graph*. But before formally specifying these graphs, we need to give a few basic definitions.

4.1 Strings, Overlaps and Overlap Graph

Let x and y be two strings over the alphabet Σ. Let us denote the length of x by $|x|$. The i^{th} character of x is denoted by $x[i]$. If $1 \leq i \leq j \leq |x|$, we use $x[i, j]$ to denote the substring of x starting at position i and ending at position j. Given two strings x and y over the alphabet Σ, we say that there is an *overlap* between

x and y, and we denote it with $x \rightleftharpoons y$, if there exists a suffix of x matching[4] a prefix of y. Let us denote with $o(x, y)$ the length of the longest such match.

Definition 5 (Overlap Graph). *Given a set of strings $S = \{r_1, r_2, \ldots, r_n\}$ and a minimum overlap threshold value k, the overlap-graph for S is a weighted bidirected graph $OG^k = (V, E)$ where:*

- $V = S = \{r_1, r_2, \ldots, r_n\}$;
- $E = \{(r_i, r_j) : (r_i \rightleftharpoons r_j) \wedge o(r_i, r_j) \geq k, r_i, r_j \in V\}$;
- *the weight of each edge (r_i, r_j) is $w(r_i, r_j) = |s_j| - o(r_i, r_j)$.*

The *overlap graph* [16] represents all the relationships that can be inferred between the strings in the set S. Note that $|r_j| - o(r_i, r_j)$ is the length of the overhang[5] for string r_j, Since each vertex/string r_i has an orientation, thus every edge has two orientations, one with respect to each of its endpoints. Because the graph is bidirectional, we need to describe how to explore the nodes of the graph to generate the set of *valid* paths.

Definition 6 (Path validity). *A path $P = \langle r_1 \overset{e_1}{\rightleftharpoons} r_2 \overset{e_2}{\rightleftharpoons} r_3 \overset{e_3}{\rightleftharpoons} \ldots \overset{e_{m-1}}{\rightleftharpoons} r_m \rangle$ in G is valid if $\forall i, 2 \leq i \leq m-1$, e_{i-1} and e_i have opposite directions at r_i.*

Note that this definition is equivalent to the *consistency property* for a layout. In order to traverse a node in the graph we need that the entry edge and the exit edge have opposite directions at the node. So we are allowed to enter a node x even if the edge e_i is pointing out of the node as long as we use an edge e_j with opposite direction to e_i when we exit the node (see figure 4 for an example of overlap graph).

Given any path P in the overlap graph, we associate a *path-string* to P that consist of the concatenation of the strings according to the order in the path, where only one copy of the overlap is kept. Clearly the weight of a path P is given by the sum of the weights of its edges:

$$w(P) = \sum_{(r_i, r_j) \in P} w(r_i, r_j) = \sum_{(r_i, r_j) \in P} (|r_j| - o(r_i, r_j)) \tag{4}$$

Note that because of the weight function associated to the edges of the graph, a path of minimum weight defines a path-string of minimum length.

4.2 String Graph

The size of the overlap graph can be dramatically reduced by a sequence of transformations whose goal is to eliminate edges that can be *transitively* inferred.

[4] The matching does not have to be perfect and it can be approximated allowing up to ϵ percent error on real data.

[5] A relaxation to an overlap, such that some small number of bases at the beginning or end of the read are excluded from the overlap region, typically because of a high error rate.

Definition 7 (transitively inferable edge). *If* $x \overset{e_1}{\rightleftharpoons} y \overset{e_2}{\rightleftharpoons} z$ *and* $x \overset{e_3}{\rightleftharpoons} z$ *are* **mutually consistent** *overlaps among nodes* x*,* y *and* z *then the edge* e_3 *is said to be transitively inferable from the sequence of edges* e_1 *and* e_2*.*

Informally the overlap between strings x and z is implied by the composition of the overlaps between x, y and z. It is important to note the edges must be mutually consistent: entry edge and the exit edge must have opposite directions. The *string graph* is a particular graph where all the contained string and transitively inferable edges are removed [17].

Definition 8 (String Graph). *Given a set of strings* $S = \{r_1, r_2, \ldots, r_n\}$ *and a minimum overlap threshold value* k*, the string graph* SG^k *for* S *is obtained from the overlap graph* OG^k *by removing contained strings (strings that are substrings of other strings) and transitively inferable edges [17].*

Such transformation can be computed in polynomial time using the algorithm proposed by Myers in [17]. In order to correctly apply the transitivity reduction step to the graph, it is important to first mark all transitively inferable edges and then remove all marked edges in a distinct phase. This is because this process is not Church-Rosser [3] and any arbitrary strategy would fail to remove some of the transitively inferable edges. Equipped with the notion of string graph, the sequence assembly problem can be formulated as follows:

Definition 9 (Sequence Assembly Problem). *Given a set of fragment or reads* $S = \{r_1, r_2, \ldots, r_n\}$ *and a minimum overlap threshold* k*, the Sequence Assembly Problem (SAP) is the problem of finding an Hamiltonian Path in the string graph* SG^k *for* S *such that its weight is minimum.*

The problem is clearly a special case of the Traveling Salesman Problem (TSP) with the following two differences: (1) instead a looking for a Hamiltonian cycle we look for an Hamiltonian path; (2) we work with bi-directed graphs instead of undirected or directed graphs. However, for circular genomes (such as plasmids and bacterial genomes), the first difference does not apply anymore as we need to find an Hamiltonian cycle as well.

Note that this formulation differs from the one presented in [18]. Specifically Nagarajan and Pop define the sequence assembly problem as one of finding a generalized Hamiltonian path (every node is visited at least once) of minimum weight in the string graph of the reads. This is in accordance to the solution proposed in [17] where they seek a cyclic tour. In such model each edge has assigned a selection constraint c that says how many times the edge should appear in the target solution: *exact* edge ($c = 1$), *required* edge ($c \geq 1$) and *optional* edge ($c \geq 0$). Note that, even if we allow a read to be potentially used more than once, the appeal to parsimony (min weight) could compromise the correctness of the layout.

Before discussing the complexity of this problem it is important to observe that this graph-theoretical formulation suffers from the same kind of problems of the shortest superstring approach. Figure 4 show an example of string graph

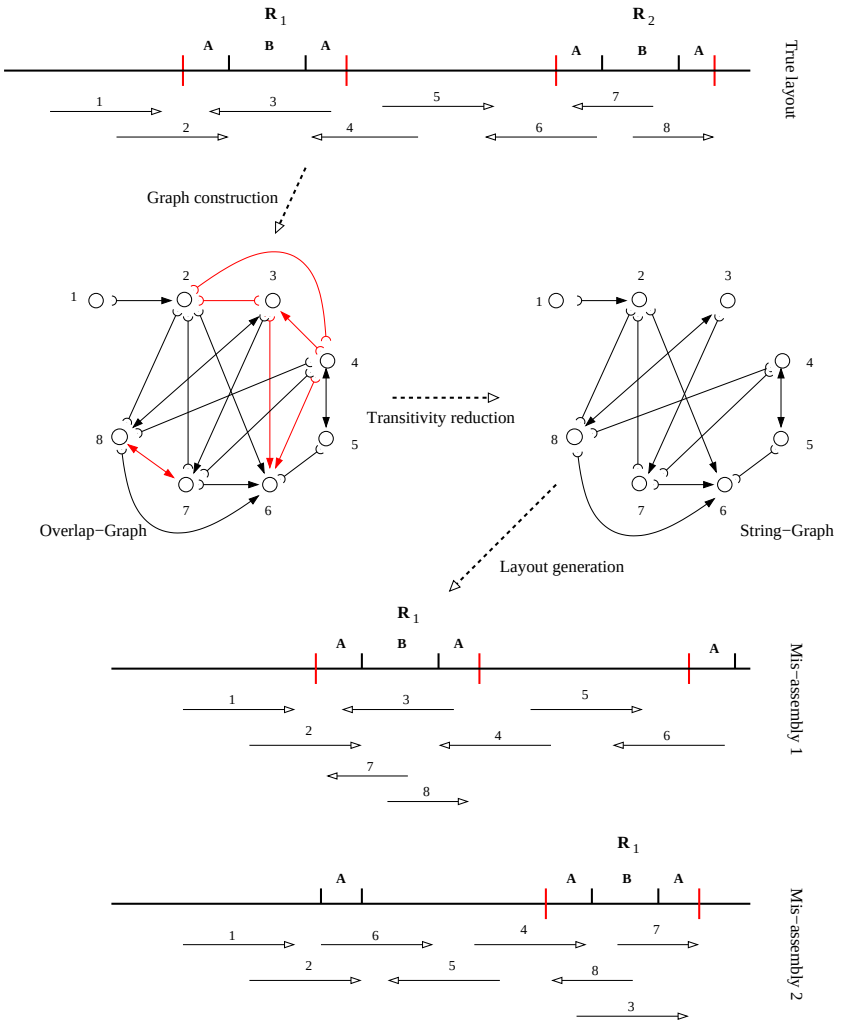

Fig. 4. Example of mis-assembly using a string graph: the removal of the transitively inferable edges (in red) produces a string graph where every (Hamiltonian) paths through all nodes creates mis-assemblies. The layouts for two of these paths are shown at the bottom: the first one with compression and the second one with both compression and inversion

where all the possible Hamiltonian paths create mis-assembly error due to the presence of a repeat. The compression error is due to the fact that repeats can induce *false positive transitively inferable edges*. For example consider the reads 3, 7 and 8 in figure 4, we have that $7 \rightleftharpoons 3$, $3 \rightleftharpoons 8$ and $7 \rightleftharpoons 8$, so the edge $7 \rightleftharpoons 8$ is removed with the negative effect to merging together reads that belong to two different copies of the repeat R_2. In particular, after removal of the transitively inferable edges, there is more than one path that traverses all the nodes and it always produces mis-assembled layouts. Note that edge $2 \rightleftharpoons 6$ cannot be removed because, although there are edges $2 \rightleftharpoons 7$ and $7 \rightleftharpoons 6$, the directions at node 7 do no match and so it cannot be traversed (the edges are not *mutually consistent*).

This example also shows another problem associated to this framework. Even if it would be possible to efficiently compute the Hamiltonian path, the string graph might have many different Hamiltonian paths (as in this example) of minimum length and all these paths represent a possible reconstruction of the genome. Additional information, such as mate-pairs, can sometimes be used to help resolve this ambiguity, although since mate-pairs are generally at most 10 to 20kbp long, in general they do not fully resolve the ambiguity except for the smallest genomes lacking any large repeats. The problem of finding a minimum weight Hamiltonian path in a directed or undirected graph is known to be $\mathcal{NP}$-complete. Since directed graphs are special types of bidirected graphs, we have:

Theorem 10. *The Sequence Assembly Problem is $\mathcal{NP}$-complete.*

4.3 De Bruijn graph

In a de Bruijn graph the notions of nodes and edges are somehow inverted compared to the overlap graph. A de Bruijn graph is formally defined as follows.

Definition 11 (De Bruijn Graph). *Given a set of strings $S = \{r_1, r_2, \ldots, r_n\}$ and a minimum overlap threshold value k, the de Bruijn graph for S is a directed graph $BG^k = (V, E)$ where:*

- $V = \{d \in \Sigma^k \mid \exists i \text{ s.t. } d \text{ is a substring of } r_i \in S\}$;
- $E = \{(d_i, d_j) : \text{if the prefix of length } k-1 \text{ of } d_i \text{ is a suffix of } d_j\}$;

Informally the set of vertices of BG^k is the set of k-mers for the set of input strings S (the *spectrum* L), and the edges correspond to their perfect $k-1$ overlap. Clearly every read $r_i \in S$ is translated into a path composed of $(|r_i| - k)$ nodes. Let us call such a path a *walk* and define it $w(r_i)$. Also note that there is no weight associated to the edges (the overlap weight is $k-1$ for all the edges and it can be omitted). Specifically, we create one node for each k-mer in the set L and a directed edge from node x_1 to node x_2 if the $k-1$ suffix of x_1 is a prefix of x_2 and we label the edge with the remaining rightmost string in x_2. Hence, in this graph each edge corresponds to one of the k-mers and so the general problem consists of finding a path that visits all the edges exactly once, namely, an *Eulerian path*. The string S corresponding to a path in this graph can be

reconstructed by concatenating k-mer sequence of the first node, in order, with all the labels of the edges in the path.

The de Bruijn Graph framework is currently the most popular approach for assembling the shorter reads coming from next-generation sequencing technologies such as Illumina [6, 12]. Moreover, it is now becoming more and more important to model the haplotyipic structure of DNA, specifically in the context of detecting DNA mutations such as short insertions and deletions of bases (INDELs). Recent works [21, 13, 8] demonstrate how sequence assembly approaches are the most promising methods for this task. However, repetitive structures, in particular near-perfect repeats, within genomes can produce artifacts in the assembly graphs that mislead such methods to make false-positive calls. Figure 5 shows an example of a near-perfect repeat that can be misinterpreted as a large deletion. The key observation is that the beginning of this sequence is a nearly perfect 69bp repeat. There is just 1bp difference between the two copies that are 15bp apart. The sequence is segmented as 19-C-49-A-14-19-T-49-G-21 where 19 and 49 are 19bp and 49bp perfect repeats, separated by a 15bp unique sequence (A, C, T, G are the regular bases). Since the longest exact repeat is 49bp long, one would expect that using k-mer=55 should be large enough to correctly assemble reads sampled from this sequence. However, if the sequencing data also contains reads with sequence 19-C-49-G, it can be wrongly interpreted as a long 84bp deletion of the A-14-19-T-49 segment when instead it is just a single base change. Since the de Bruijn graph is constructed using perfect matches of length $k-1 = 54$ (no mismatches allowed), the only way to connect all the 55-mers from these two sequences is to construct a false bubble jumping form the first copy of the near-perfect repeat to the second copy. When aligned to the reference, the sequence associated to the branch will show a false-positive deletion.

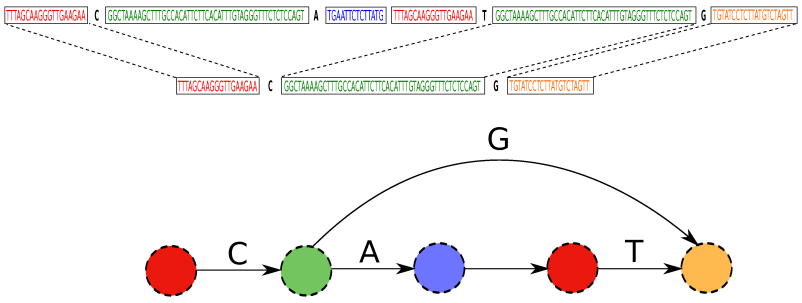

Fig. 5. Example of false bubble in a De Bruijn graph

Finally, it is important to note that in the de Bruijn framework, similarly to the String Graph framework, the graph can have more then one Eulerian path and choosing the correct one is not trivial. Indeed the number of valid paths may be extremely large, and bounded only by the product of the factorial of the degrees of the nodes times the number of potential spanning trees of the graph

[10]. Although an Eulerian path can be computed in polynomial time (using the Hierholzer's algorithm [7]), it might not represent a correct assembly of the input reads (the path may not be read-coherent [17]). However, as mentioned before, each read correspond to a particular walk in the de Bruijn graph, and any walk that contains all the reads as subwalks (a *superwalk*) represents a possible assembly of the reads. In this framework a parsimonious solution corresponds to a superwalk of minimum length:

Definition 12 (Superwalk Problem). *Given a set of reads* $S = \{r_1, \ldots, r_n\}$ *find a minimum length superwalk in the De Bruijn graph* BG^k *of* S.

It can be shown that this problem is also $\mathcal{NP}$-complete by reduction from the Shortest Superstring Problem [15]:

Theorem 13. *The Superwalk Problem is* $\mathcal{NP}$*-complete.*

5 Discussion

The process of abstracting a problem from its biological interpretation is a powerful tool to better investigate a biological problem. However, as demonstrated in this paper for the sequence assembly problem, it is very important to develop (biologically) correct formulations. The shortest superstring formulation was an elegant theoretical abstraction, but it was clearly oblivious to what biology needs to make a correct interpretation of genomic data. The subsequent graph-theoretical formulations, although more powerful than the simpler SSP model, still suffer from similar problems when dealing with repeat structures. We have presented examples of many popularly accepted formulations that can lead to miss-assembly errors. Although all the SAP formations presented in this paper lead to computationally intractable problems ($\mathcal{NP}$-complete), approximated solutions can be efficiently computed using graph search methods (BFS vs DFS) often in combination with branch-and-bound method [20, 21]. A better understanding and modeling of the sequence composition (e.g., repeats) contained within genomes has been one of the key factors to improve accuracy in computational genomics, but much work needs to be done to achieve the goal of an *error-free* reconstruction. Finally there is now the urgency to model the haplotypic structure of the human genome which introduces another level of complexity for example in the algorithms seeking to discover genetic mutations.

References

1. Armen, C., Stein, C.: A 2 2/3-approximation algorithm for the shortest superstring problem. In: CPM. pp. 87–101 (1996)
2. Bradnam, K., et al.: Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. GigaScience 2(1), 10 (2013)
3. Church, A., Rosser, J.B.: Some properties of conversion. Transactions of the American Mathematical Society 39(3), 472–482 (1936)

4. Earl, D.A., et al.: Assemblathon 1: A competitive assessment of de novo short read assembly methods. Genome Research (2011)
5. Gallant, J., Maier, D., Astorer, J.: On finding minimal length superstrings. Journal of Computer and System Sciences 20(1), 50 – 58 (1980)
6. Gnerre, S., et al.: High-quality draft assemblies of mammalian genomes from massively parallel sequence data. Proceedings of the National Academy of Sciences 108(4), 1513–1518 (2011)
7. Hierholzer, C., Wiener, C.: Ueber die mglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. Mathematische Annalen 6(1), 30–32 (1873)
8. Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., McVean, G.: De novo assembly and genotyping of variants using colored de bruijn graphs. Nature Genetics 44(2), 226–232 (2012)
9. Karp, R.M.: The role of algorithmic research in computational genomics. Computational Systems Bioinformatics Conf., IEEE Computer Society 0, 10 (2003)
10. Kingsford, C., Schatz, M., Pop, M.: Assembly complexity of prokaryotic genomes using short reads. BMC Bioinformatics 11(1), 21 (2010)
11. Koren, S., et al.: Hybrid error correction and de novo assembly of single-molecule sequencing reads. Nature Biotechnology 30(7), 693–700 (Jul 2012)
12. Li, R., et al.: De novo assembly of human genomes with massively parallel short read sequencing. Genome Research 20(2), 265–272 (2010)
13. Li, S., Li, R., Li, H., Lu, J., Li, Y., Bolund, L., Schierup, M., Wang, J.: Soapindel: Efficient identification of indels from short paired reads. Genome Research (2012)
14. Mardis, E.R.: The impact of next-generation sequencing technology on genetics. Trends in Genetics 24(3), 133 – 141 (2008)
15. Medvedev, P., Georgiou, K., Myers, G., Brudno, M.: Computability of models for sequence assembly. In: Giancarlo, R., Hannenhalli, S. (eds.) Algorithms in Bioinformatics, LNCS, vol. 4645, pp. 289–301. Springer Berlin Heidelberg (2007)
16. Myers, E.W.: Toward simplifying and accurately formulating fragment assembly. Journal of Computational Biology 2, 275–290 (1995)
17. Myers, E.W.: The fragment assembly string graph. Bioinformatics 21(suppl_2), ii79–85 (2005)
18. Nagarajan, N., Pop, M.: Parametric complexity of sequence assembly: theory and applications to next generation sequencing. Journal of computational biology 16(7), 897–908 (July 2009)
19. Narzisi, G., Mishra, B.: Comparing de novo genome assembly: The long and short of it. PLoS ONE 6(4), e19175 (04 2011)
20. Narzisi, G., Mishra, B.: Scoring-and-unfolding trimmed tree assembler: concepts, constructs and comparisons. Bioinformatics 27(2), 153–160 (2011)
21. Narzisi, G., O'Rawe, J.A., Iossifov, I., ha Lee, Y., Wang, Z., Wu, Y., Lyon, G.J., Wigler, M., Schatz, M.C.: Accurate detection of de novo and transmitted indels within exome-capture data using micro-assembly. bioRxiv (2013)
22. Rittel, H.W.J., Webber, M.M.: Dilemmas in a general theory of planning. Policy Sciences 4, 155 – 169 (1973)
23. Roberts, R., Carneiro, M., Schatz, M.: The advantages of smrt sequencing. Genome Biology 14(7), 405 (2013)
24. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology 147(1), 195–197 (March 1981)
25. Tarhio, J., Ukkonen, E.: A greedy approximation algorithm for constructing shortest common superstrings. Theor. Comput. Sci. 57(1), 131–145 (1988)
26. Turner, J.S.: Approximation algorithms for the shortest common superstring problem. Inf. Comput. 83(1), 1–20 (1989)