Superstring Graph: A New Approach for Genome Assembly

Bastien Cazaux, Gustavo Sacomoto, Eric Rivals

Superstring Graph: A New Approach for Genome Assembly

Bastien Cazaux[1,2], Gustavo Sacomoto[3], and Eric Rivals[1,2(✉)]

[1] LIRMM, Université de Montpellier, CNRS UMR 5506, Montpellier, France
{cazaux,rivals}@lirmm.fr
[2] Institut Biologie Computationnelle, Montpellier, France
[3] INRIA Rhône-Alpes and Université Lyon 1, CNRS,
UMR 5558, 69000 Lyon, France
sacomoto@gmail.com

Abstract. With the increasing impact of genomics in life sciences, the inference of high quality, reliable, and complete genome sequences is becoming critical. Genome assembly remains a major bottleneck in bioinformatics: indeed, high throughput sequencing apparatus yield millions of short sequencing reads that need to be merged based on their overlaps. Overlap graph based algorithms were used with the first generation of sequencers, while de Bruijn graph (DBG) based methods were preferred for the second generation. Because the sequencing coverage varies locally along the molecule, state-of-the-art assembly programs now follow an iterative process that requires the construction of de Bruijn graphs of distinct orders (i.e., sizes of the overlaps). The set of resulting sequences, termed unitigs, provide an important improvement compared to single DBG approaches. Here, we present a novel approach based on a digraph, the Superstring Graph, that captures all desired sizes of overlaps at once and allows to discard unreliable overlaps. With a simple algorithm, the Superstring Graph delivers sequences that includes all the unitigs obtained from multiple DBG as substrings. In linear time and space, it combines the efficiency of a greedy approach to the advantages of using a single graph. In summary, we present a first and formal comparison of the output of state-of-the-art genome assemblers.

1 Introduction

Ongoing improvements in DNA sequencing technologies have dramatically increased the throughput of sequencers, thereby authorising the launch of very large genome projects: the 1000 human genomes for studying natural variations [15], the 10 K vertebrate genomes for phylogenomics issues [11] or the 10,000 rice genomes, which aims at getting a genomic overview of all wild and cultivated rice varieties. If getting the collections of raw sequencing reads becomes easier and cheaper, assembling complex eukaryotic genomes remains one of the major practical and theoretical challenges in bioinformatics.

This work is supported by ANR Colib'read (ANR-12-BS02-0008), the Institut de Biologie Computationnelle (ANR-11-BINF-0002).

R. Dondi et al. (Eds.): AAIM 2016, LNCS 9778, pp. 39–52, 2016.
DOI: 10.1007/978-3-319-41168-2_4

With the advent of Next Generation sequencing technologies, most of the sequencing performed yields huge numbers of short reads. For that reason, the de Bruijn Graph (DBG) approach, also termed as Eulerian sequence assembly, has been preferred to the Overlap-Layout-Consensus approach, which resorts to an overlap graph and was used with traditional Sanger sequencing. The DBG encodes each k-mer of the read set as a node and contains an arc from one node to another if the $k+1$-mer obtained by merging them occurs in at least one read. A path in the DBG represents the sequence obtained by merging the k-mers along it. Many assemblers infer unitigs by traversing non branching paths in the DBG, or contigs if the path chooses some extension when it encounters a branching node. Unitigs, which are the parts of contigs comprised between two branching nodes, represent unambiguous regions of the target genome. However, the choice of the value of k is critical and difficult in practice. Indeed, the density of sequencing reads along the molecule depends on the amount of sequencing and fluctuates for technological and biological reasons. Some regions have low coverage, while others may collapse the expected number of reads several times because they contain genomic repeats.

1.1 Related Works

Recently, some papers have investigated the power of combining the assembly made successively by several DBG of different orders, i.e. varying the order k in a user defined range $[k_{min}, k_{max}]$. The goal is to enable the algorithm to find paths both in tangled and fragmented regions of the graph. More precisely, (1) using larger overlaps to find paths in tangled regions, where repeats in DNA create bubbles and branching nodes because shared k-mers are collapsed in the DBG, and (2) to find paths between connected components with shorter overlaps. The algorithms named IDBA and SPAdes do that by building several DBG with different values of k [1,13]. Their main problem is the necessity to build several DBG. Currently, state-of-the-art methods exploit multiple sizes of overlaps and also impose a constraint on the read coverage (i.e. the density of reads in the sequence region). IDBA [13] builds a DBG of order k_{min}, computes the set of unitigs, merges them with the reads, and iterates this procedure for all k until k_{max} (see Algorithm 2). SPAdes adopts a slightly different algorithm [1]. For all values of k between k_{max} and k_{min}, it computes in parallel the unitigs of each DBG_k, and makes their union. Finally, it builds $DBG_{k_{max}}$ on this set and outputs its unitigs. The result of the two approaches are similar. In practice, building that many DBGs is prohibitive, and hence both IDBA and SPAdes limit themselves to a few (i.e., ≈ 4) values of k.

Boucher et al. propose to extend the BOSS succinct data structure, which succinctly encodes a DBG, to enable the dynamic update from k to $k+1$ [2]. Their practical performance allows to navigate between different orders on bacterial and a Human dataset. However, the question of which size of overlap/order is needed in a given region remains. In other works, we have shown how to build in linear time a DBG of order k from either a Generalised Suffix Tree, a Suffix Array, or a Truncated Suffix Tree of the reads and exhibited an algorithm to update k also in linear time [4].

An alternative and interesting approach, called the manifold de Bruijn graph, which assigns words of arbitrary length to nodes in the graph, was presented in [9]. This perspective is different from the one we propose here.

Formally, the question of assembling strings is modelled as the Shortest Common Superstring, also termed the Shortest Linear Superstring. It requires finding a single superstring containing the input words as substrings. This well-studied problem is known to be NP-hard [7] and APX-hard [12]. Many approximation algorithms have been proposed, which solve a relaxed problem known as the Shortest Cyclic Cover of Strings (SCCS) – see Sect. 2 for a definition of a cyclic cover. SCCS is usually solved in polynomial time with the Hungarian algorithm; we have recently exhibited a linear time algorithm for SCCS and introduced the Superstring Graph for this sake [5]. To handle the fact that DNA is double stranded, we have extended this algorithm to the case where either the input word or its reverse complement (in the biological sense) must appear as a substring in the cyclic cover [3].

1.2 Summary of Our Contribution

Let P be a set of words on a finite alphabet. The well-known shortest superstring problems ask for a either cyclic or linear superstring of minimal length, and the Shortest Cyclic Cover problem asks for a collection of cyclic strings of minimal norm (cumulated length). Here, with the Shortest Mixed Cover of Strings or simply Shortest Mixed Cover, we relax the requirements and accept a solution made of a collection of strings that can be linear or cyclic. We introduce a graph that represents all the maximal overlaps between the input words in small space: the Truncated Hierarchical Overlap Graph (THOG). We show first that the Superstring Graph is embedded in the Truncated Hierarchical Overlap Graph of P; and second, that it captures the set of Mixed Covers built by a greedy algorithm that agglomerates words using their largest overlaps ranked in decreasing order.

As mentioned above, Generalised Suffix Tree can also serve to build the DBG of order k for P [4]. However, classical DBG are limited in the size of overlaps, which must be of length $k-1$. This is a strong limitation, and a natural remedy is to consider overlaps of different sizes, by extending the framework of DBG. Current state-of-the-art proposals successively build and explore several DBG to compute unitigs with different overlap sizes [1,13]. Our proposal is to capture multiple overlap sizes in a single graph and to explore its paths to compute unitigs.

Finally, we show that unitigs built with the IDBA approach are substrings of those found in our Superstring Graph. Moreover, we characterise when a unitig from the SG captures an overlap missed in a multiple DBG approach. It can be proven that IDBA solution is contained in the SG solution. A strong point of the SG algorithm is to retain the sensitivity of variable order DBG without building several graphs, which remains computationally prohibitive. Indeed, it is stated in [13] that exploring the whole range of orders $[k_{min}, k_{max}]$ is not feasible on large-scale data. In fact, each iteration in IDBA takes linear time in $||P||$, while the

SG algorithm takes overall linear time in $||P||$. Our contributions are theoretical, but our solution has a linear space complexity (Theorem 3). For simplicity, here we disregard the fact that one usually does not know from which DNA strand the input reads of an assembly problem come from. Hence, both the reads and their reverses complement are considered in assembly problem. However, the approach described in [3] shows that the results developed here can be extended to handle the case of missing information about the DNA strand. Due to space constraints, the proof of Theorems 1, 2, and 3 are omitted here.

1.3 Notation and Basic Definitions

About Strings. We consider two kinds of strings: linear and cyclic strings. For a string s, the length of s is $|s|$. For a linear string s and $i \leq j$ in $\{1, \ldots, |s|\}$, $s[i, j]$ is the linear substring of s beginning at the position i and ending at the position j, $s[i]$ is the substring $s[i, i]$, $s[1, j]$ is a prefix of s and $s[i, |s|]$ is a suffix of s. A prefix (or suffix) s' of s is proper if s' is different of s. For another linear string t, the *maximum overlap* from s to t, denoted by $\mathrm{ov}(s, t)$, is the longest substring that is a proper suffix of s and a proper prefix of t. The prefix from s to t, denoted by $\mathrm{pr}(s, t)$, is such that $s = \mathrm{pr}(s, t)\mathrm{ov}(s, t)$ and the suffix from s to t, denoted by $\mathrm{suf}(s, t)$, is such that $t = \mathrm{ov}(s, t)\mathrm{suf}(s, t)$. The merge of s with t using their maximal overlap is denoted $s \odot t$ and is equal to $\mathrm{pr}(s, t)\mathrm{ov}(s, t)\mathrm{suf}(s, t)$. Since we consider only maximal overlaps, we simply use the term overlap. For simplicity, we denote the concatenation of s with t simply by st.

We say that a linear string w is a substring of a cyclic string c if there exists w_c a linear permutation of c such that w is a substring of w_c^∞ (where $w_c^\infty = w_c w_c \ldots$). To ease distinction between linear and cyclic strings, we will denote a cyclic string c by $\langle c \rangle$.

For a set P of finite strings, we define and denote the *norm* of P by $||P|| := \sum_{w \in P} |w|$. For two strings x and y, we denote by $x \subset_{sub} y$ the fact that x is a substring of y. We denote the *set of factors* of P by $Fact(P) := \{w \mid \exists s_i \in P,\ w \subset_{sub} s_i\}$. Moreover, for k an integer, we denote by $Fact_k(P)$ the subset of $Fact(P)$ made of strings of length k.

About permutations. Let E be a finite set. A permutation on E is a bijection from E onto itself. Let σ be a permutation on E. The *partition* of E *due to* σ, which is denoted by $Part_\sigma$, is a partition $(E_1, \ldots, E_p)$ of E of maximal cardinality, and such that for any i in $[1, p]$ and for any x of E_i and for any integer k, one has $\sigma^k(x) \in E_i$. Then, one can define p permutations on E, $(\sigma_1, \ldots, \sigma_p)$, such that for any i in $[1, p]$, for any x in E one has $\sigma_i(x) := \sigma(x)$ if $x \in E_i$, and $\sigma_i(x) := x$ otherwise. Then $(\sigma_1, \ldots, \sigma_p)$ is called a *decomposition* of σ in circular permutations.

Throughout the article. let $P := \{s_1, \ldots, s_n\}$ be a set of input words, and $||P||$ denotes the norm of P. Without loss of generality, we always assume that P is factor-free, i.e. for any two strings of P, none is a substring of the other.

2 Permutations and Truncated Hierarchical Overlap Graph

Let $P = \{s_1, \ldots, s_n\}$ be a finite set of linear strings over a finite alphabet. We can define two types of covers:

- a *cyclic cover of strings* of P is a set $C = \{\langle c_1\rangle, \ldots, \langle c_p\rangle\}$ of cyclic strings such that each string s_i of P is a substring of a $\langle c_j\rangle$ of C, i.e., $s_i \subset_{sub} \langle c_j\rangle$.
- a *mixed cover of strings* of P is a set $C = \{\langle c_1\rangle, \ldots, \langle c_q\rangle, l_{q+1}, \ldots, l_p\}$ of cyclic and linear strings such that each string s_i of P is a substring of an element of C.

Obviously, one could consider also linear covers of strings. However, by concatenating the strings of a shortest linear cover one gets a shortest linear superstring. Thus, the problem of finding a shortest linear string cover is as hard and as difficult to approximate as the shortest linear superstring problem (NP-hard [7] and APX-hard [12]). Another reason explains our interest in mixed cover of strings: state-of-the-art assemblers like IDBA or SPAdes can yield linear and cyclic strings. Indeed, the de Bruijn Graph may contain an isolated cycle. Hence, their result is indeed a mixed cover of strings. Clearly, a cyclic cover is a mixed cover, and the norm of a shortest cyclic cover of P is at most that of a shortest mixed cover of P. To our knowledge the problem of finding a shortest mixed cover of strings has not yet been studied.

It is known that for each optimal solution and each greedy solution of $SCCS$, there exists a permutation such that this permutation induces this cyclic cover of strings [5]. Figure 1 shows how to build a cyclic cover of strings from a permutation. Indeed, let $P = \{s_1, \ldots, s_n\}$ be a set of strings which is factor-free and let σ be a permutation. We define

$$CC(P, \sigma) = \{circular(P_1, \sigma_1), \ldots, circular(P_m, \sigma_m)\}$$

where the decomposition in circular permutation of σ is $\sigma_1 \ldots \sigma_m$, $Part_\sigma = \{P_1, \ldots, P_m\}$ is such that for any i in $[1, m]$, P_i is the element of $Part_\sigma$ corresponding to σ_i, and for all i between 1 and m where $P_i = \{s^i_1, \ldots, s^i_{|P_i|}\}$:

$$circular(P_i, \sigma_i) := \langle pr(s^i_1, s^i_{\sigma_i(1)}).pr(s^i_{\sigma_i(1)}, s^i_{\sigma^2_i(1)}).\ldots.pr(s^i_{\sigma^{|P_i|-1}_i(1)}, s^i_1)\rangle.$$

We denote by $Overlap(CC(P, \sigma))$ the set of overlaps used by the cyclic cover of strings $CC(P, \sigma)$, i.e. $Overlap(CC(P, \sigma)) = \{ov(s_i, s_{\sigma(i)}) \mid \forall i \in \{1, \ldots, n\}\}$.

For any cyclic cover of strings w of P, we can map each word of P on w, and create the permutation σ_w defined so that on the mapping $s_{\sigma_w(i)}$ is just after the string s_i. Hence, we get that $|CC(P, \sigma_w)| \leq |w|$. Indeed, $CC(P, \sigma_w)$ always merges the input words using their maximal overlaps, while w can use any overlap. Thus, we can restrict the problem $SCCS$ to consider only cyclic covers induced by permutations.

Some assemblers consider that a subset of overlaps are unreliable, for example if these are too short [1,13]. In fact they forbid this subset of overlaps. We adapt our definitions to this case and introduce a set F representing the maximal

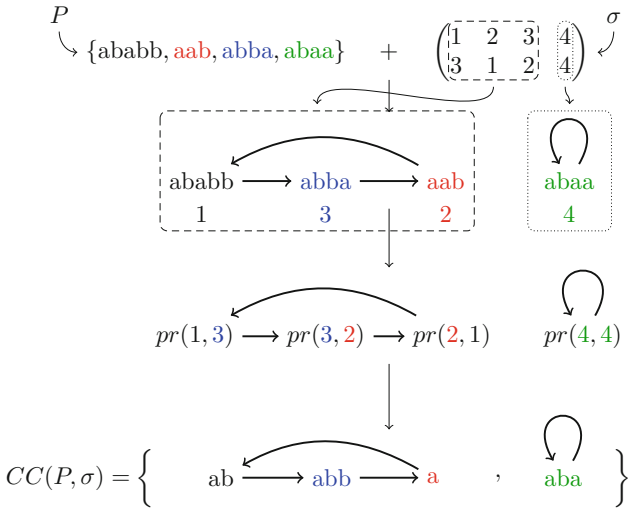

Fig. 1. From a permutation to a cyclic cover. Example with the input set $P := \{\text{ababb}, \text{aab}, \text{abba}, \text{abaa}\}$. Instance of a cyclic cover of P obtained with a permutation σ. We obtain the cyclic cover $CC(P, \sigma) = \{\langle\text{ababba}\rangle, \langle\text{aba}\rangle\}$. (Color figure online)

elements among all forbidden overlaps. All substrings of elements of F will be forbidden. We define variants of shortest cover problems that are constrained by the set of forbidden overlaps (see Definitions 1 and 2).

Definition 1 (Constrained Shortest Cyclic Cover of Strings (CSCCS)).

- ***Input****: Two sets of linear strings P and F.*
- ***Output****: A cyclic cover of strings C induced by a permutation of P such that $Overlap(C) \cap Fact(F) = \emptyset$, which minimises $||C||$.*

Note that if we assume two elements x, y of F such that y is a substring of x, $Overlap(C) \cap Fact(\{x\}) = \emptyset$ implies that $Overlap(C) \cap Fact(\{y\}) = \emptyset$. Hence from now on, F is assumed to be factor-free. Unfortunately, some instances of $CSCCS$ lack solutions, and for other instances, the greedy algorithm (Algorithm 1) does not find any solution (see Example 1).

So, we define the following problem which is a relaxed formulation of $CSCCS$.

Definition 2 (Constrained Shortest Mixed Cover of Strings (CSMCS)).

- ***Input****: Two sets of linear strings P and F,*
- ***Output****: A mixed cover of strings C induced by a permutation of P such that $Overlap(C) \cap Fact(F) = \emptyset$, which minimises $||C||$.*

We denote by **CMC(P,F)** the set of mixed covers C induced by a permutation of P such that $Overlap(C) \cap Fact(F) = \emptyset$. Let $OPT_{CMC}(P, F)$ be the set of optimal solutions of $CSMCS$ for (P, F).

This time, we can determine easily whether **CMC(P,F)** is empty or not (see Proposition 1 and Example 1). We get the same result as for the greedy solutions of CSMC(P,F) (see Theorem 1 and Example 1).

Proposition 1. ***CMC(P,F)*** *is empty if and only if* $Fact(F) \cap P$ *is not empty.*

Example 1. 1. Let $P = \{\text{abba}, \text{baab}, \text{bab}\}$ and $F = \{\text{b}\}$, Then the set of cyclic covers of P constrained by F is empty but $\{\langle \text{abba} \rangle, \text{bab}\} \in$ **CMC(P,F)**,
2. Let $P = \{\text{abec}, \text{bed}, \text{cfabe}, \text{dgab}\}$ and $F = \{\text{b}\}$, $OPT_{SCCS}(P) = \{\langle \text{ecfabedgab} \rangle\}$; however, the greedy algorithm for $CSCCS$ gives no solution but the greedy algorithm for $CSMCS$ gives $\{\langle \text{cfabe} \rangle, \text{bedgab}\}$ as a solution.

From now on, we assume that $F \subseteq Fact(P) \setminus P$ and F is factor-free.

Let $P := \{s_1, \ldots, s_n\}$ be a factor-free set of words, and $F \in Fact(P) \setminus P$. For any permutation σ of $\{1, \ldots, |P|\}$, we can obtain a cyclic cover of strings. We ask when such a cyclic cover satisfies the constraint of F, i.e., when it uses a forbidden overlap. For any circular permutation σ_c in a decomposition of σ in circular permutations, we define a set of violations, denoted $Violations(P, F, \sigma_c)$. If this set is empty, the induced cyclic cover is a solution of CSCCS and of CSMCS. If $Violations(P, F, \sigma_c)$ contains only one violation, say i, then the cyclic string can be transformed into a linear string satisfying the constraint of F. The transformation is as follows: the forbidden overlap occurs between s_i and $s_{\sigma_c(i)}$.

One builds the linear word by cutting the word $circular(P, \sigma_c)$ between the words s_i and $s_{\sigma_c(i)}$ to obtain:

$$linear(P, \sigma_c, i) := pr(s_{\sigma_c(i)}, s_{\sigma_c^2(i)})\ pr(s_{\sigma_c^2(i)}, s_{\sigma_c^3(i)})\ \ldots\ pr(s_{\sigma_c^{n-1}(i)}, s_i)\ s_i$$

Let $F \subseteq Fact(P)$ and σ_c be a circular permutation of $\{1, \ldots, n\}$. We set $Violations(P, F, \sigma_c) := \{i \in \{1, \ldots, n\} \mid \exists f \in F$ such that $ov(s_i, s_{\sigma_c(i)}) \subset_{sub} f\}$. Violations are the overlaps used in the cyclic cover (induced by σ) that are substrings of an element of F.

We say that a circular permutation σ_c is *coherent with* (P, F) if and only if $|Violations(P, F, \sigma_c)| \leq 1$. We say that a permutation σ is coherent with (P, F) if each circular permutation in a decomposition of σ in circular permutations is coherent with (P, F). For any circular permutation σ_c that is coherent with (P, F) we define the Mixed Cover (MC) induced by σ_c on (P, F) as

$$MC(P, F, \sigma_c) := \begin{cases} circular(P, \sigma_c) & \text{if } |Violations(P, F, \sigma_c)| = 0, \\ linear(P, \sigma_c, i) & \text{if } Violations(P, F, \sigma_c) = \{i\}. \end{cases}$$

and for any permutation σ coherent with (P, F)

$$MC(P, F, \sigma) := \{MC(P_1, F, \sigma_1), \ldots, MC(P_m, F, \sigma_m)\}$$

where $(\sigma_1, \ldots, \sigma_m)$ is a decomposition of σ in circular permutation, and $Part_\sigma = \{P_1, \ldots, P_m\}$ is such that for any i in $[1, m]$, P_i is the element of $Part_\sigma$ corresponding to σ_i. Let **PMC(P,F)** denote the subset of Mixed Covers induced by

a permutation coherent with (P, F). We obtain the following proposition, which means that (1) if there is a solution to CSMC, there also exists one solution induced by a coherent permutation, and (2) an optimal solution is induced by a coherent permutation.

Proposition 2. *Let P be a factor-free set of words and let $F \subseteq Fact(P)$. One has*

1. $\boldsymbol{PMC(P,F)} = \emptyset$ *if and only if* $\boldsymbol{SMC(P,F)} = \emptyset$.
2. $\boldsymbol{OPT(P,F)} \subseteq \boldsymbol{PMC(P,F)} \subseteq \boldsymbol{SMC(P,F)}$,

Let us introduce the Truncated Hierarchical Overlap Graph (THOG), which is a generalised version of the Hierarchical Overlap Graph defined in [3].

Let $Ov(P)$ be the set of maximum overlaps from a string of P to another string or the same string of P. Let $Ov^*(P, F)$ be $Ov(P)$ minus the set of all factors of forbidden overlaps; in other words, $Ov^*(P, F) := Ov(P) \setminus Fact(F)$. Now, we define the *Truncated Hierarchical Overlap Graph (THOG)* of (P, F), in which the nodes are either words of P or allowed overlaps between these words, and an arc links a string to the node representing its maximal suffix or the maximal prefix of a string with this string. Two examples of THOG are shown in Fig. 4a and c.

Definition 3. *The Truncated Hierarchical Overlap Graph of (P, F), denoted by $THOG(P, F)$, is the oriented graph $(P \cup Ov^*(P, F), R \cup B)$ where:*

$$R = \{(x, y) \in (P \cup Ov^*(P, F)) \times (P \cup Ov^*(P, F)) \mid y \text{ longest suffix of } x \text{ in } P\}$$
$$B = \{(y, x) \in (P \cup Ov^*(P, F)) \times (P \cup Ov^*(P, F)) \mid y \text{ longest prefix of } x \text{ in } P\}$$

R denotes the set of red arcs, and B the set of blue arcs.

It is known that overlaps between two strings are explicit nodes in the Generalised Suffix Tree of these words [8]. Hence, all nodes of THOG are explicit nodes of the Generalised Suffix Tree of P. Moreover, a blue arc is a contracted path of edges of the suffix tree, while a red arc is a contracted path of suffix links. Altogether, we can built THOG in linear time.

Proposition 3. *The graph $THOG(P, F)$ can be built in linear time in $||P||$.*

Let s_i and s_j be two words of P. We define the RB-path from s_i to s_j, denoted by $RB\text{-}path(s_i, s_j)$, as the path in $THOG(P, F)$ going from s_i to $ov(s_i, s_j)$ using only arcs from R, and then from $ov(s_i, s_j)$ to s_j using only arcs of B. Let σ be a permutation of P coherent with (P, F). Then, for any i between 1 and n, the RB-path from s_i to $s_{\sigma(i)}$ is well defined and exists in $THOG(P, F)$.

THOG construction algorithm We execute Gusfield's algorithm for finding maximal overlap nodes in the Generalised Suffix Tree (GST) of P [8] and along the way we mark the words of F. This gives an explicit list of the THOG nodes. We then perform a depth first traversal of the GST (using the suffix tree arcs) to set all blue arcs of the THOG. Finally, we perform the same using the tree of suffix links to the set of all red arcs. Altogether it takes linear time in the GST of P.

3 Greedy Algorithm and Superstring Graph

Here, we define the Superstring Graph and introduce the greedy algorithm for the problem *Shortest Mixed Cover of Strings* (SMC) of a set P of words. The difference between the norm of the input and the norm of a solution is the compression achieved by this solution. Finally, we show that a Eulerian multi-path of the SG and the associated set of words form a solution of SMC, which approximates the optimal compression by a factor $\frac{1}{2}$, as later shown in Theorem 1.

We define the greedy algorithm for CSMCS (see Algorithm 1).

Algorithm 1. The greedy algorithm for CSMCS

Input: P a set of linear words and $F \subseteq Fact(P)$ **Output**: $C' \in$ **PMC(P,F)**
$C := \emptyset$
while $Ov^*(P, F)$ *is not empty* **do**
  Select u and v of P which have the longest overlap (u can be equal to v)
  $P := P \setminus \{u, v\}$
  if $u = v$ *(i.e.* $u \odot v$ *is cyclic)* **then** $C := C \cup \{u \odot v\}$;
  else $P := P \cup \{u \odot v\}$;
return $C \cup P$

Let **Greedy(P,F)** denote the set of solutions of algorithm greedy for CSMCS (for simplicity, we say greedy solutions). One has the following theorem, whose third statement gives the $\frac{1}{2}$-approximation ratio of compression of the greedy algorithm. To prove this ratio, one can define a subset system for CSMCS, which turns out to be 2-extendible. The ratio of $\frac{1}{2}$ follows directly from this 2-extendibility [10] (see [6] for details). Note this greedy approximation ratio of $\frac{1}{2}$ for the compression is the same as for the well-studied Shortest Common Superstring problem [6,14]. These considerations support Theorem 1 (omitted proof).

Theorem 1. *Let P be a factor-free set of words and let $F \subseteq Fact(P)$. One has*

1. ***Greedy(P,F)*** $\subseteq$ ***PMC(P,F)*** $\cap \{\sigma$ *permutation coherent with* $(P, F)\}$,
2. ***Greedy(P,F)*** $= \emptyset$ *if and only if* ***SMC(P,F)*** $= \emptyset$.
3. *Let* $w_g \in$ ***Greedy(P,F)*** *and* $w_o \in$ ***OPT(P,F)***. *Then* $||P|| - w_g \geq \frac{1}{2}(||P|| - w_o)$.

The inclusions of set of solutions are illustrated in Fig. 2. Section 3 states how greedy solutions can be found in linear time.

Let σ be a permutation that is coherent with (P, F) and such that $MC(P, F, \sigma)$ is a greedy solution for CSMCS. Let us denote by $G(\sigma)$ the subgraph that consists of the set of RB-paths from s_i to $s_{\sigma(i)}$ for all i in $[1, n]$. As in [5], one can show that any two permutations that are coherent with (P, F) and correspond to a greedy solution for CSMCS, yield the same graph. We call this graph the *Superstring Graph (SG)* and define it as follows. Two examples of superstring graphs are shown in Fig. 4b and d.

Definition 4 (Superstring Graph). *The Superstring Graph of* (P, F)*, denoted SG(P,F), is the graph* $G(\sigma)$ *where* σ *is a permutation that is coherent with* (P, F) *and corresponds to a greedy solution for CSMCS.*

As the Superstring Graph of (P, F) is embedded in THOG of (P, F), it can clearly be built in a time linear in $||P||$.

Theorem 2. *The Superstring Graph of* (P, F) *can be built in time* $O(||P|| + ||F||)$.

4 Comparing the Superstring Graph with a Multiple Order DBG Approach

The IDBA assembler iteratively builds DBG basically as depicted in Algorithm 2. The only difference concerns the step for removing the so-called short *dead-ends* in the DBG at each iteration. As the name says, a dead-end is a simple path starting after a branching node and ending in a node having a single neighbour. IDBA removes dead-ends shorter than $2k$, which are likely due to nucleotidic errors [13]. Such a dead-end would make up a very short, biologically meaningless, unitig. However, for the simplicity of the proofs, we consider a simplified algorithm without short dead-end removal. As usual, we require that the input set P of words is factor-free.

Algorithm 2. Algorithm IDBA assembler where $DB_m(P, k)$ is the de Bruijn Graph of order k (i.e. , dBG_k^+) where we remove all nodes which represent a k-mer of coverage smaller than m.

Input: A set P of reads factor-free **Output**: A set $U_{k_{max}}$ of unitigs
for $k_{min} \leq k \leq k_{max}$ **do**
  $H_k = DB_m(P, k)$
  $U_k =$ Unitigs H_k
  $P = P \cup U_k$
return $U_{k_{max}}$

*About IDBA algorithm (*Algorithm 2*)*

Complexity. For each k between k_{min} and k_{max}, the algorithm needs to look at all the strings of the instance, i.e. $||P||$. At the end, the complexity of Algorithm 2 is at least linear in $(k_{max} - k_{min}) \times ||P||$.

Theoretical Solution. Nothing prevents a unitig of $DB_m(P, k)$ from being a cycle. Let w be a string. We denote the cover of w in P by $Cov_P(w) := \{(i, j) \mid \exists r_i \in P$ such that $w = r_i[j : j + |w|]\}$. Let $P(k, m)$ denote the set of substrings of P satisfying, for any $w \in P(k, m)$: $|w| \leq k$, and $Cov_P(w) \geq m$, and for all word w' such that w is a proper substring of w', $Cov_P(w') < m$ or $|w'| > k$. Hence, we have that $U_{k_{max}}$, which is a set of cyclic and linear strings, is in fact a mixed cover of string of $P(k_{max} + 1, m)$ with the set of $(k_{min} - 1)$-mers, i.e. $Fact_{k_{min}-1}(P)$, is taken as forbidden overlaps.

We are going to use the Superstring Graph on the Truncated Hierarchical Overlap Graph to build in linear time in $||P||$ an improved mixed cover for the same instance, that is for strings of $P(k_{max}+1, m)$ with the set of $(k_{min}-1)$-mers taken as forbidden overlaps. The mixed cover obtained from the superstring graph is smaller in terms of inclusion, of cardinality and of norm than $U_{k_{max}}$ (see Theorem 3).

Let $SG(P, k_{max}, k_{min}, m)$ be the Superstring Graph of $(P(k_{max}+1, m), Fact_{k_{min}-1}(P))$. A RB-route of a Superstring Graph is a sub-path of a sequence of RB-paths.

Proposition 4. *We can build $SG(P, k_{max}, k_{min}, m)$ in linear time in $||P||$.*

Proof. With the Generalised Suffix Tree of P, we can build $P(k_{max}+1, m)$ in linear time in the size of P. We can build the Superstring Graph of $(P(k_{max}+1, m), Fact_{k_{min}-1}(P))$ in linear time in $||P(k_{max}+1, m)||$, because in this case, we can determine the nodes of the tree corresponding to the elements of $Fact_{k_{min}-1}(P)$ during the construction of the GST of P without reading these strings. Hence, it improves on the complexity of Theorem 2, and show that one can build the SG in linear time in $||P||$.

Let U_{SG} be the set of labelled maximal RB-routes (u, v) of the $SG(P, k_{max}, k_{min}, m)$ such that $(d_R^{in}(u) = d_B^{in}(u) = 1$ or $d_R^{in}(u) + d_B^{in}(u) \leq 1)$ and $d_R^{out}(v) + d_B^{out}(v) \leq 1$. Here, $d_R^{in}(u)$ denotes the in-degree in number of red arcs of node u in the superstring graph, and $d_R^{out}(u)$ the out-degree of u in number of red arcs. The notation $d_B^{in}(u)$ and $d_B^{out}(u)$ are defined similarly for blue arcs.

Proposition 5. *For all $c \in U_{k_{max}}$, there exists $x \in U_{SG}$ such that $c \subset_{sub} x$.*

Proposition 6. *For all $x \in U_{SG}$, $\exists c_1, \ldots, c_q \in U_{k_{max}}$ such that $x = c_1 \odot \ldots \odot c_q$ and for all i between 1 and $q-1$, $|ov(c_i, c_{i+1})| \geq k_{min}$.*

Theorem 3. *We can build a mixed cover that includes a solution of Algorithm 2 in time in $O(||P||)$, and in linear space in the size of the de Bruijn Graph of order k_{max} of P.*

Now, we know that the words of U_{SG}, the solution provided by the SG, contains all unitigs of IDBA as substrings. Some words of U_{SG} are exactly equal to some unitigs of IDBA. However, the remaining words of U_{SG}, contain strictly more than one unitig of IDBA as substring. In other words, they elongate the unitigs of IDBA by capturing an overlap missed by IDBA. We formalise this result in the next proposition.

Proposition 7. *(Figure 3) For all $x \in U_{SG}$, $\exists c_1, \ldots, c_q \in U_{k_{max}}$ such that $x = c_1 \odot \ldots \odot c_q$ and for all i between 1 and $q-1$, there exists $y \in U_{SG}$ such that $\exists c'_1$ and $c'_2 \in U_{k_{max}}$ such that $c'_1 \odot c'_2 \subset_{sub} y$ and $ov(c_i, c_{i+1})$ is a strict prefix of $ov(c'_1, c'_2)$.*

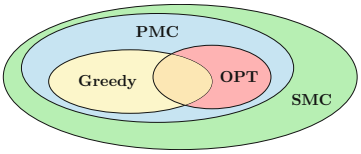

Fig. 2. Inclusions of sets of solutions of the greedy algorithm for CSMCS.

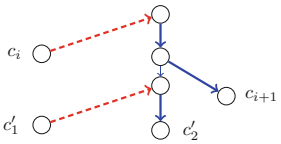

Fig. 3. Illustration of Proposition 7.

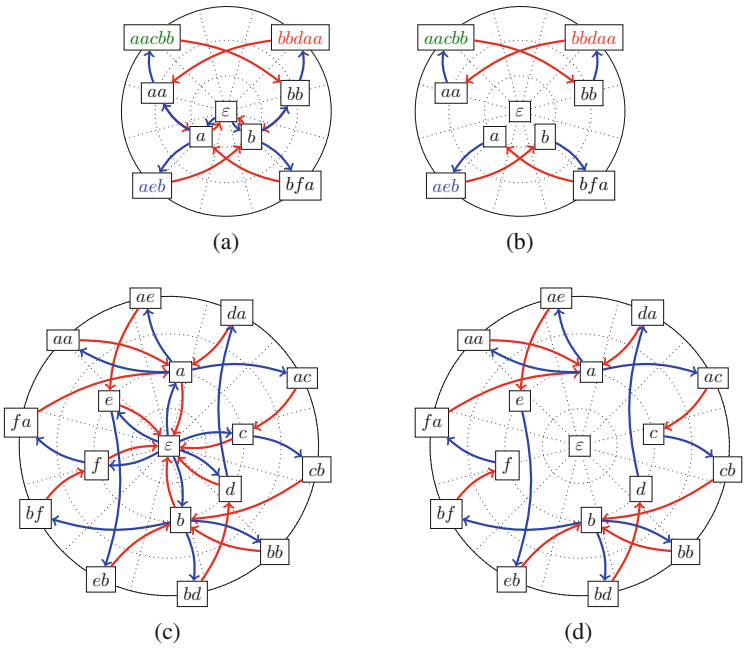

Fig. 4. Examples of truncated hierarchical overlap graphs ((a) and (c)) and of associated Superstring Graphs ((b) and (d)) for $P := \{\text{aacbb}, \text{bbdaa}, \text{aeb}, \text{bfa}\}$. (a) and (b) have instance (P, F) with $F := \emptyset$. (c) and (d) have instance $(P(2,1), F)$ with $F := Fact_1(P)$; F forbids any overlap of length 1 or 0. (Color figure online)

5 Conclusion

State-of-the-art genome assemblers, like IDBA or SPAdes, build multiple DBG with distinct values of k to improve the quality of assembled unitigs. In a formal manner, we compared the result of IDBA with the sequences obtained using the Superstring Graph of an input set P of reads. The SG is a recently introduced digraph with labels on its arcs, which is embedded in a Truncated Hierarchical Overlap Graph (THOG) of P. The SG yields solutions for *Constrained Mixed Cover* that greedily merge the input words using their maximal overlaps; hence, we get a $\frac{1}{2}$-approximation ratio for the compression. We show that the unitigs output by IDBA are always substrings of the sequences assembled with the

SG, and that the converse is false. Indeed, some assembled sequences from the SG extend IDBA unitigs by merging words with smaller overlaps that cannot be incorporated in IDBA. For the first time, a theoretical framework helps to understand and to characterise formally the output of real-world assembly software that adopts a multiple-order de Bruijn graph approach. It also provides a way to improve on their results. Moreover the Superstring Graph offers the possibility to dynamically extend the range of overlap lengths considered without recomputing the unitigs from scratch. It can be adapted to cope with reverse complement of the reads/k-mers using the approach of [3]. The main advantage of the SG, which is linear in the input size, over IDBA is to concentrate all overlaps needed to build a similar assembly in one single graph.

Acknowledgements. We thank the reviewers for their comments and suggestions.

References

1. Bankevich, A., Nurk, S., Antipov, D., Gurevich, A.A., Dvorkin, M., Kulikov, A.S., Lesin, V.M., Nikolenko, S.I., Pham, S., Prjibelski, A.D., Pyshkin, A.V., Sirotkin, A.V., Vyahhi, N., Tesler, G., Alekseyev, M.A., Pevzner, P.A.: SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. J. Comp. Biol. **19**(5), 455–477 (2012)
2. Boucher, C., Bowe, A., Gagie, T., Puglisi, S.J., Sadakane, K.: Variable-order de bruijn graphs CoRR abs/1411.2718 (2014)
3. Cazaux, B., Cánovas, R., Rivals, E.: Shortest DNA cyclic cover in compressed space. In: Data Compression Conference DCC, pp. 536–545. IEEE Computer Society Press (2016)
4. Cazaux, B., Lecroq, T., Rivals, E.: From indexing data structures to de bruijn graphs. In: Kulikov, A.S., Kuznetsov, S.O., Pevzner, P. (eds.) CPM 2014. LNCS, vol. 8486, pp. 89–99. Springer, Heidelberg (2014)
5. Cazaux, B., Rivals, E.: A linear time algorithm for shortest cyclic cover of strings. J. Discrete Algorithms (2016). doi:10.1016/j.jda.2016.05.001
6. Cazaux, B., Rivals, E.: The power of greedy algorithms for approximating Max-ATSP, cyclic cover, and superstrings. Discrete Appl. Math. (2015). doi:10.1016/j.dam.2015.06.003
7. Gallant, J., Maier, D., Storer, J.A.: On finding minimal length superstrings. J. Comput. Syst. Sci. **20**, 50–58 (1980)
8. Gusfield, D., Landau, G.M., Schieber, B.: An efficient algorithm for the all pairs suffix-prefix problem. Inf. Process. Lett. **41**(4), 181–185 (1992)
9. Lin, Y., Pevzner, P.A.: Manifold de bruijn graphs. In: Brown, D., Morgenstern, B. (eds.) WABI 2014. LNCS, vol. 8701, pp. 296–310. Springer, Heidelberg (2014)
10. Mestre, J.: Greedy in approximation algorithms. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 528–539. Springer, Heidelberg (2006)
11. G. K. C. of Scientists: Genome 10K a proposal to obtain whole-genome sequence for 10 000 vertebrate species. J. Hered. **100**(6), 659–674 (2009)
12. Ott, S.: Lower bounds for approximating shortest superstrings over an alphabet of size 2. In: Widmayer, P., Neyer, G., Eidenbenz, S. (eds.) WG 1999. LNCS, vol. 1665, pp. 55–64. Springer, Heidelberg (1999)

13. Peng, Y., Leung, H.C.M., Yiu, S.M., Chin, F.Y.L.: IDBA – a practical iterative de bruijn graph de novo assembler. In: Berger, B. (ed.) RECOMB 2010. LNCS, vol. 6044, pp. 426–440. Springer, Heidelberg (2010)
14. Tarhio, J., Ukkonen, E.: A greedy approximation algorithm for constructing shortest common superstrings. Theor. Comp. Sci. **57**, 131–145 (1988)
15. The 1000 Genomes Project Consortium: An integrated map of genetic variation from 1,092 human genomes. Nature **491**(7422), 56–65 (2012)