From Indexing Data Structures to de Bruijn Graphs

Bastien Cazaux, Thierry Lecroq, Eric Rivals

From Indexing Data Structures to de Bruijn Graphs*

Bastien Cazaux[†], Thierry Lecroq[‡], Eric Rivals[†]
[†] L.I.R.M.M. & Institut Biologie Computationnelle
Université de Montpellier II, CNRS U.M.R. 5506
Montpellier, France
[‡] LITIS EA 4108 & UFR des Sciences et des Techniques,
Université de Rouen, France
cazaux@lirmm.fr, thierry.lecroq@univ-rouen.fr, rivals@lirmm.fr

5th March 2014

Abstract

New technologies have tremendously increased sequencing throughput compared to traditional techniques, thereby complicating DNA assembly. Hence, assembly programs resort to de Bruijn graphs (dBG) of k-mers of short reads to compute a set of long contigs, each being a putative segment of the sequenced molecule. Other types of DNA sequence analysis, as well as preprocessing of the reads for assembly, use classical data structures to index all substrings of the reads. It is thus interesting to exhibit algorithms that directly build a dBG of order k from a pre-existing index, and especially a contracted version of the dBG, where non branching paths are condensed into single nodes. Here, we formalise the relationship between suffix trees/arrays and dBGs, and exhibit linear time algorithms for constructing the full or contracted dBGs. Finally, we provide hints explaining why this bridge between indexes and dBGs enables to dynamically update the order k of the graph.

1 Introduction

The de Bruijn graph (dBG) of order k on an alphabet Σ with σ symbols has σ^k vertices corresponding to all the possible distinct strings of length k on the alphabet Σ and there is a directed edge from vertex u to vertex v if the suffix of u of length $k-1$ equals the prefix of v of length $k-1$. De Bruijn graphs have various properties and

*This work is supported by ANR Colib'read (ANR-12-BS02-0008) and Défi MASTODONS SePhHaDe from CNRS.

[†]

[‡]

are more commonly defined on all the k-mers of the strings of a finite set rather than on all the possible strings of length k on the alphabet. When a vertex u has only one outgoing edge to vertex v and when v has only one ingoing edge from vertex u then the two vertices can be merged. By applying this rule whenever possible, one gets a contracted dBG. dBGs occur in different contexts. In bioinformatics they are largely used in *de novo* assembly due to a result of Pevzner *et al* [14]. Indeed recent sequencing technologies allow to obtain hundreds of million of short sequencing reads (about 100 nucleotides long) from one DNA sample. Next step is to reconstruct the genome sequence using assembly algorithms. However, the volume of read data to process has forced the shift from the classical overlap graph approach, which requires too much memory, towards a de Bruijn Graph where vertices are k-mers of the reads. In this context, there exist compact exact data structures for storing dBGs [7, 3, 15, 5] and probabilistic data structures such as Bloom filters [12, 6, 5]. Onodera *et al* propose to add to the succinct dBG representation of [3] a bit vector marking the branching nodes, thereby enabling them to simulate efficiently a contracted dBG, where each simple path is reduced to one edge [11].

Suffix trees are well-known indexing data structures that enable to store and retrieve all the factors of a given string. They can be adapted to a finite set of strings and are then called generalised suffix trees (GSTs). They can be built in linear time and space. They have been widely studied and used in a large number of applications (see [1, 9]). In practice, they consume too much space and are often replaced by the more economical suffix arrays [10], which have the same properties.

Read analysis and assembly include preliminary steps like filtering and error correction. To speed up such steps, some algorithms index the substrings, or the k-mers of the reads. Hence, before the assembly starts, the read set has already been indexed and mined. For instance, the error correction software `hybrid-shrec` builds a GST of all reads [16]. It can thus be efficient to enable the construction of the dBG for the subsequent assembly, directly from the index rather than from scratch. For these reasons, we set out to find algorithms that transform usual indexes into a dBG or a contracted dBG. It is also of theoretical interest to build bridges between well studied indexes and this graph on words. Despite recent results [15, 11], formal methods for constructing dBG from suffix trees are an open question. Notably, the String Graph, which is also used for genome assembly, can be constructed from a FM-index [17].

In this article, given a finite collection S of strings and an integer k we formalise the relationship between GSTs and dBGs and show how to linearly build the dBG of order k for S. Next we show how to directly build the contracted dBG of order k for S in linear time and space, without building the dBG. We also show how to perform the same task using suffix arrays. Finally, we give some hints on how to dynamically adapt our dBG construction from order k to $k-1$ or from k to $k+1$.

2 Preliminaries

An *alphabet* Σ is a finite set of *letters*. A finite sequence of elements of Σ is called a *word* or a *string*. The set of all words over Σ is denoted by $\Sigma^\star$, and ε denotes the empty word. For a word x, $|x|$ denotes the *length* of x. Given two words x and y, we denote by

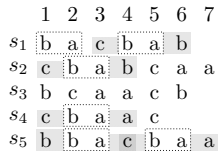


Figure 1: $Support_S(ba) = \{(1,1),(1,4),(2,2),(4,2),(5,2),(5,5)\}$, $RC_S(ba) = \{\varepsilon, c, cb, cba, cbab, b, bc, bca, bcaa, a, ac, cbaa\}$, $LC_S(ba) = \{\varepsilon, c, ac, bac, b, bbac\}$ and $d_S(ba) = 0$. One has $RC_S(ba) \cap \Sigma = \{a, b, c\}$. Thus, the word ba is not right extensible in S (see Def. 2.2).

xy the *concatenation* of x and y. For every $1 \leq i \leq j \leq |x|$, $x[i]$ denotes the i-th letter of x, and $x[i\,..\,j]$ denotes the *substring* or *factor* $x[i]x[i+1]\ldots x[j]$. Let k be a positive integer. If $|x| \geq k$, $first_k(x)$ is the *prefix* of length k of x and $last_k(x)$ is the *suffix* of length k of x. A substring of length k of x is called a k-mer of x. For i such that $1 \leq i \leq |x| - k + 1$, $(x)_{k,i}$ is the k-mer of x starting in position i, *i.e.* $(x)_{k,i} = x[i\,..\,i+k-1]$. Thus we have $first_k(x) = (x)_{k,1}$ and $last_k(x) = (x)_{k,|x|-k+1}$. We denote by $\sharp(\Lambda)$ the cardinality of any finite set Λ.

Let $S = \{s_1, \ldots, s_n\}$ be a finite set of words. Let us denote the sum of the lengths of the input strings by $\|S\| := \sum_{s_i \in S} |s_i|$. We denote by F_S the set of factors of words of S. For a word w of F_S,

- $Support_S(w)$ is the set of pairs (i, j), where w is the substring $(s_i)_{|w|,j}$. $Support_S(w)$ is called the support of w in S.
- $RC_S(w)$ (resp. $LC_S(w)$) is the set of *right context* (resp. *left context*) of the word w in S, *i.e.* the set of words w' such that $ww' \in F_S$ (resp. $w'w \in F_S$).
- $\lceil w \rceil_S$ is the word ww' where w' is the longest word of $RC_S(w)$ such that $Support_S(w) = Support_S(ww')$. In other words, such that w and ww' have exactly the same support in S.
- $\lfloor w \rfloor_S$ is the word w' where w' is the longest prefix of w such that $Support_S(w') \neq Support_S(w)$.
- $d_S(w) := |\lceil w \rceil_S| - |w|$.

In other words, $\lceil w \rceil_S$ is the longest extension of w having the same support than w in S, while $\lfloor w \rfloor_S$ is the shortest reduction of w with a support different from that of w in S. These definitions are illustrated in a running example, with $S := \{bacbab, cbabcaa, bcaacb, cbaac, bbacbaa\}$, presented in Fig. 1.

We give the definition of a de Bruijn graph for assembly (dBG for short), which differs from the original definition of a complete graph over all possible words of length k stated by de Bruijn [8].

Definition 2.1 *Let k be a positive integer and $S := \{s_1, \ldots, s_n\}$ be a set of n words. The* de Bruijn graph *of order k for S, denoted by DBG_k^+, is a directed graph, $DBG_k^+ :=$*

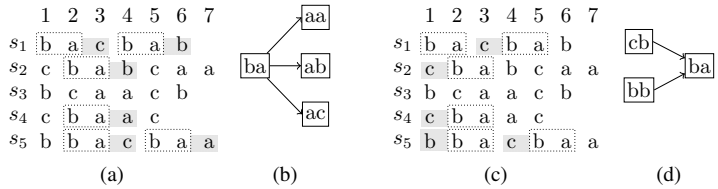


Figure 2: Examples of arcs from DBG_k^+. (a) letters in the right context of ba, and (b) the successors of node ba in DBG_2^+; one for each letter in $RC_S(w) \cap \Sigma$. (c) letters in the left context of ba, and (d) the predecessors of node ba in DBG_2^+.

(V^+, E^+), *whose vertices are the k-mers of words of S and where an arc links u to v if and only if u and v are two successive k-mers of a word of S,* i.e.*:* $V^+ := F_S \cap \Sigma^k$ *and* $E^+ := \{(u,v) \in V^{+2} \mid last_{k-1}(u) = first_{k-1}(v)$ *and* $v[k] \in RC_S(u)\}$.

Examples of arcs are displayed on Fig. 2.

Let us introduce now the notions of extensibility for a substring of S and that of a Contracted dBG (CdBG for short).

Definition 2.2 (Extensibility) *Let w be a word of* F_S*, w is* right extensible *in S if and only if* $\sharp(RC_S(w) \cap \Sigma) = 1$ *and w is* left extensible *in S if and only if* $\sharp(LC_S(w) \cap \Sigma) = 1$.

As S is clear from the context, we simply omit the "in S". Let w be a word of $\Sigma^\star$. The word w is said to be a *unique* k'-mer of S if and only if $k' \geq k$ and for all $i \in [1..k'-k+1]$, $(w)_{k,i} \in F_S$ and for all $j \in [1..k'-k]$, $(w)_{k,j}$ is right extensible and $(w)_{k,j+1}$ is left extensible.

Definition 2.3 *A* contracted de Bruijn graph *of order k, denoted by* $CDBG_k^+ = (V_c^+, E_c^+)$*, is a directed graph where:*
$V_c^+ = \{w \in \Sigma^\star \mid w$ *is a* k'*-mer unique maximal by substring and* $k' \geq k\}$ *and* $E_c^+ = \{(u,v) \in V_c^{+2} \mid last_{k-1}(u) = first_{k-1}(v)$ *and* $v[k] \in RC_S(last_k(u))\}$.

Note that in the previous definition, an element w in V_c^+ does not necessarily belong to F_S, since w may only exist as the substring of the agglomeration of two words of S. Thus, let w be a k'-mer unique maximal by substring with $k' \geq k$: $last_k(w)$ is not right extensible or $RC_S(last_k(w)) \cap \Sigma = \{a\}$ and $last_{k-1}(w) \cdot a$ is not left extensible, $first_k(w)$ is not left extensible or $LC_S(first_k(w)) \cap \Sigma = \{a\}$ and $a \cdot first_{k-1}(w)$ is not right extensible. With this argument, we have both following propositions.

Proposition 1 *Let* $(u,v) \in E_c^+$*;* $(last_k(u), first_k(v)) \in E^+$ *and there exists* $w \in V^+$ *such that* $(w, first_k(v)) \in E^+ \setminus \{(last_k(u), first_k(v))\}$ *or* $(last_k(u), w) \in E^+ \setminus \{(last_k(u), first_k(v))\}$.

Proposition 2 *Let* $(u,v) \in E^+$*. If u is right extensible and v is left extensible, then there exists* $w \in V_c^+$ *such that uv*$[k]$ *is a substring of w. Otherwise, there exists* $(u', v') \in E_c^+$ *such that* $u = last_k(u')$ *and* $v = first_k(v')$.

According to Prop. 1 and 2, CDBG_k^+ is the graph DBG_k^+ where the arcs (u,v) are contracted if and only if u is right extensible and v is left extensible.

3 Definition of de Bruijn Graphs with words

Let k be a positive integer. We define the following three subsets of F_S.

- $InitExact_{S,k} = \{w \in F_S \mid |w| = k \text{ and } d_S(w) = 0\}$
- $Init_{S,k} = \{w \in F_S \mid |w| \geq k \text{ and } d_S(first_k(w)) = |w| - k\}$
- $SubInit_{S,k} = InitExact_{S,k-1}$

A word of $InitExact_{S,k}$ is either only the suffix of some s_i or has at least two right extensions, while the first k-mer of a word in $Init_{S,k} \setminus InitExact_{S,k}$ has only one right extension.

Proposition 3 $InitExact_{S,k} = Init_{S,k} \cap \{w \in F_S \mid |w| = k\}$.

For w an element of $Init_{S,k}$, $first_k(w)$ is a k-mer of S. Given two words w_1 et w_2 of $Init_{S,k}$, $first_k(w_1)$ and $first_k(w_2)$ are distinct k-mers of S. Furthermore for each k-mer w' of S, there exists a word w of $Init_{S,k}$ such that $first_k(w) = w'$. From this, we get the following proposition.

Proposition 4 *There exists a bijection between $Init_{S,k}$ and the set of the k-mers of S.*

According to Def. 2.1 and Prop. 4, each vertex of DBG_k^+ can be assimilated to a unique element of $Init_{S,k}$. To define the arcs between the words of $Init_{S,k}$, which correspond to arcs of DBG_k^+, we need the following proposition, which states that each single letter that is a right extension of w gives rise to a single arc.

Proposition 5 *For $w \in InitExact_{S,k}$ and $a \in \Sigma \cap RC_S(w)$, there exists a unique $w' \in Init_{S,k}$ such that $last_{k-1}(w)a$ is a prefix of w'.*

The set $Init_{S,k}$ represents the nodes of DBG_k^+. Let us now build the set of arcs that is isomorphic to E^+. Let w be a word of $Init_{S,k}$ and $\mathrm{Succ}(w)$ denote the set of successors of $first_k(w)$: $\mathrm{Succ}(w) := \{x \in Init_{S,k} \mid (first_k(w), first_k(x)) \in E^+\}$. We know that for each letter a in $RC_S(w)$, there exists an arc from $first_k(w)$ to $first_k(last_{|w|-1}(w)a)$ in DBG_k^+. We consider two cases depending on the length of w:

Case 1 $|w| = k$. According to Prop. 3, $w \in InitExact_{S,k}$ and hence $last_{k-1}(w) \in SubInit_{S,k}$. Therefore, the outgoing arcs of w in DBG_k^+ are the arcs from w to w' satisfying the condition of Prop. 5. Then,
$\mathrm{Succ}(w) = \cup_{a \in \Sigma \cap RC_S(w)} \lceil last_{k-1}(w)a \rceil_S$.

Case 2 $|w| > k$. As w is longer than k, it contains the next k-mer; hence $first_k(last_{|w|-1}(w)a) = first_k(last_{|w|-1}(w))$, and there exists a unique outgoing arc of w: that from w to $\lceil w[2..k] \rceil_S$. Indeed, by definition of $Init_{S,k}$, $\lceil w[2..k] \rceil_S \in Init_{S,k}$, and thus $\mathrm{Succ}(w) = \{\lceil w[2..k] \rceil_S\}$.

Now, we can build integrally DBG_k^+ or more exactly an isomorphic graph of DBG_k^+. Thus for simplicity, from now on we confound the graph we build with DBG_k^+. To do the same with CDBG_k^+, we need to characterise the concepts of right and left extensibility in terms of word properties. By the construction of DBG_k^+, we have the following results.

Proposition 6 *Let w be a word of $Init_{S,k}$. $first_k(w)$ is right extensible if and only if $|w| > k$ or $\sharp(RC_S(w) \cap \Sigma) = 1$.*

Proposition 7 *Let w be a word of $Init_{S,k}$ such that $first_k(w)$ is right extensible. Let the letter a be the unique element of $RC_S(first_k(w)) \cap \Sigma$, then $last_{k-1}(first_k(w))a$ is left extensible if and only if $\sharp(Support_S(first_k(w))) = \sharp(Support_S(last_{k-1}(first_k(w))a) \setminus \{(i,1) \mid 1 \leq i \leq n\})$.*

We present a generic algorithm to build incrementally CDBG_k^+. In the following sections, we exhibit algorithms to compute DBG_k^+ and CDBG_k^+ for two important indexing structures.

4 Transition from the suffix tree to de Bruijn graphs

A *generalised* ST (GST) can index the substrings of a set of words. Generally for this sake, all words are concatenated and separated by a special symbol not occurring elsewhere. However, this trick is not compulsory, and an alternative is to keep the indication of a terminating node within each node.

4.1 The Suffix Tree and its properties

The *Generalised Suffix Tree* (GST) of a set of words S is the suffix tree of S, where each word of S does not finish necessarily by a letter of unique occurrence. Hence, for each node v of the GST of S, we keep in memory the set, denoted by $\text{Suff}_S(v)$, of pairs (i,j) such that the word represented by v is the suffix of s_i starting at position j. Let us denote by T the GST of S (from now on, we simply say the tree) and by V_T its set of nodes. For $v \in V_T$, $Children(v)$ denotes its set of children and $f(v)$ its parent.

Some nodes of T may have just one child. The size of the union of $\text{Suff}_S(v)$ for all node v of T equals the number of leaves in the GST when the words end with a terminating symbol. Hence, the space to store T and the sets $\text{Suff}_S(.)$ is linear in $\|S\|$. By simplicity, for a node v of T, the word represented by v is confused with v. For each node v of T, $v \in F_S$. As all elements of F_S are not necessarily represented by a node of T, we give the following proposition.

Proposition 8 *The set of nodes of T is exactly the set of words w of F_S such that $d_S(w) = 0$.*

We recall the notion of a suffix link (SL) for any node v of T (leaves included). Let $sl(v)$ denote the node targeted by the suffix link of v, *i.e.* $sl(v) = v[2..|v|]$. By definition of a suffix tree, for all $w \in F_S$, there exists a node v of T such that w is a prefix of v. Let v' the node of minimal length of T such that w is a prefix of v, then $|v'| = |w| + d_S(w)$, and therefore $\lceil w \rceil_S = v'$.

Proposition 9 *Let $w \in F_S$. Then $|\lceil w \rceil_S| \geq |w| > |f(\lceil w \rceil_S)|$, where $f(\lceil w \rceil_S)$ is the* parent *of $\lceil w \rceil_S$ in T.*

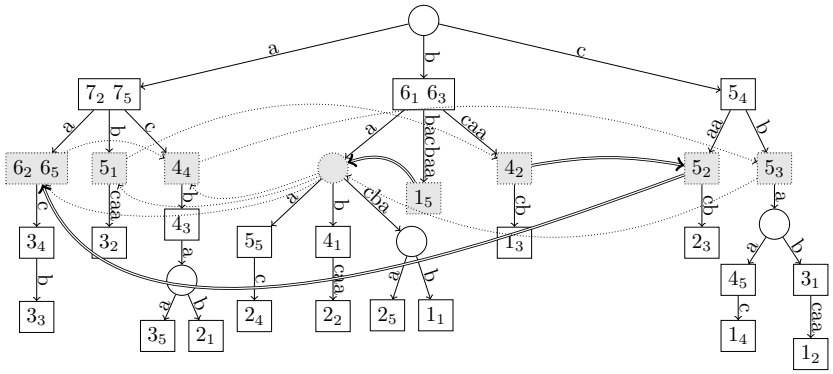


Figure 3: The GST for our running example and the constructed dBG for $k := 2$. Square nodes represent words that occur as a suffix of some s_i, circle nodes are the other nodes of T. Grey nodes represent the vertices of the dBG. Each square node contains i_j when it represents the suffix of s_j starting at position i. The curved arrows are the edges of the dBG; those in dotted lines correspond to Case 1 and those in doubled lines to Case 2 (see p. 7).

4.2 Construction of DBG_k^+

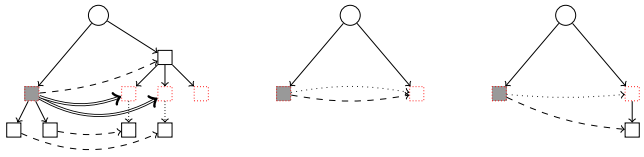
Let $[x_1..x_m]$ be the set of k-mers of S. According to the definition of $Init_{S,k}$ and to Prop. 4, $Init_{S,k} = [\lceil x_1 \rceil_S .. \lceil x_m \rceil_S]$. Thus, by Prop. 9, $Init_{S,k} = \{v \in V_T \mid |f(v)| < k \text{ and } |v| \geq k\}$. Similarly, $InitExact_{S,k} = \{v \in V_T \mid |v| = k\}$. Now, it appears clearly that $InitExact_{S,k}$ is a subset of $Init_{S,k}$, since for all $v \in V_T$, $|f(v)| < |v|$.

We consider the same two cases as for the construction of E^+ on p. 5, but in the case of a tree. Let $v \in Init_{S,k}$.

Case 1 $|v| = k$ (Fig. 4a). As $v \in InitExact_{S,k}$, $sl(v) \in SubInit_{S,k}$. Therefore, each child u of $sl(v)$ is an element of $Init_{S,k}$. Thus, the outgoing arcs of v in DBG_k^+ are the arcs from v to the child u of $sl(v)$ where the first letter of the label between $sl(v)$ and u is an element of the right context of v. As the set of the first letters of the label between v and children of v is exactly $RC_S(v) \cap \Sigma$, the number of outgoing arcs of v in DBG_k^+ is the number of children of v. To build the outgoing arcs of v in DBG_k^+, for each child u' of v, we associate v with the node of $Init_{S,k}$ between the root and $sl(u')$, *i.e.* $\lceil first_k(sl(u')) \rceil_S$.

Case 2 $|v| > k$ (Fig. 4b and 4c). We have that $sl(v)$ is a node of V_T. As $|v| > k$, $|sl(v)| \geq k$. Thus, there exists an element of $Init_{S,k}$ between the root and $sl(v)$. We associate v with this node, *i.e.* $\lceil first_k(sl(v)) \rceil_S$.

We illustrate these two cases in Fig. 3. For Case 1: v is $6_2\ 6_5$, $sl(v)$ is $7_2\ 7_5$, the unique child u' of v is 3_4, and $sl(u')$ is 4_4, which is in $Init_{S,k}$. For Case 2: v is 1_5, $sl(v)$ is 2_5, and $\lceil first_k(sl(v)) \rceil_S$ is .



(a) (b) (c)

Figure 4: Figures (a), (b) and (c) show Case 1 and Case 2 for the arcs of DBG_k^+. The grey node is v. The dashed arcs correspond to suffix links. The arcs of the dBG are in bold (a) for the Case 1 and in dotted lines (b) (c) for the Case 2.

In both cases, building the arcs of E^+ requires to follow the SL of some node. The node, say u, pointed at by a SL may not be initial. Hence, the initial node representing the associated first k-mer of u is the only ancestral initial node of u. We equip each such node u with a pointer $p(u)$ that points to the only initial node on its path from the root. In other words, for any $u \notin Init_{S,k}$ such that $|u| > k$, one has $p(u) := \lceil first_k(u) \rceil_S$.

The algorithm to build the DBG_k^+ is as follows. A first depth first traversal of T allows to collect the nodes of $Init_{S,k}$ and for each such node to set the pointer $p(.)$ of all its descendants in the tree. Finally to build E^+, one scans through $Init_{S,k}$ and for each node v one adds $\text{Succ}(v)$ to E^+ using the formula given above. Altogether this algorithm takes a time linear in the size of T. Moreover, the number of arcs in E^+ is linear in the total number of children of initial nodes. This gives us the following result.

Theorem 10 *For a set of words S, building the de Bruijn Graph of order k, DBG_k^+ takes linear time and space in $|T|$ or in $\|S\|$.*

4.3 Construction of CDBG_k^+

In Section 3, we have seen an algorithm that allows to compute directly CDBG_k^+ provided that one can determine if a node v is right extensible and if $\text{next}(v)$ is left extensible, where $\text{next}(v)$ denotes the only successor of v. Let us see how to compute the extensibility in the case of a Suffix Tree.

By applying Prop. 6 in the case of tree, for an element v of $Init_{S,k}$, $first_k(v)$ is right extensible if and only if $|v| > k$ or $\sharp(Children(v)) = 1$. Thus checking the right extensibility of a node takes constant time.

For the left extensibility of the single successor of a node, one only needs the size of support of some nodes (Prop. 7). Let us see first how to compute $\sharp(Support_S(.))$ on the tree, and then how to apply Prop. 7.

Proposition 11 *Let v be a word of F_S and $V_T(\lceil v \rceil_S)$ denotes the set of nodes of the subtree rooted in $\lceil v \rceil_S$. $Support_S(v) = \cup_{v' \in V_T(\lceil v \rceil_S)} Suff_S(v')$.*

Along a traversal of the tree, we compute and store $\sharp(Support_S(v))$ and $\sharp(Support_S(v) \cap \{(i,1) \mid 1 \leq i \leq n\})$ for each node v in linear time in $|T|$.

Let v be a word of $Init_{S,k}$ such that $first_k(v)$ is right extensible.

Case 1 If $|v| = k$ then $first_k(v) = v$ and $\sharp(Children(v)) = 1$. Let u be the only child of v. Thus, $|u| > k$, $\sharp(RC_S(v) \cap \Sigma) = \{u[k+1]\}$, and $last_{k-1}(v)u[k+1] = first_k(sl(u))$. Hence, $\sharp(Support_S(v)) = \sharp(Support_S(first_k(sl(u))) \setminus \{(i,1) \mid 1 \leq i \leq n\})$ and by Prop. 7, $first_k(sl(u))$ is left extensible.

Case 2 If $|v| > k$ then $\sharp(RC_S(first_k(v)) \cap \Sigma) = \{v[k+1]\}$ and $last_{k-1}(first_k(v)) \cdot v[k+1] = last_k(first_{k+1}(v)) = first_k(sl(v))$. By Prop. 7, $first_k(sl(v))$ is left extensible if and only if
$\sharp(Support_S(first_k(v))) = \sharp(Support_S(first_k(sl(v))) \setminus \{(i,1) \mid 1 \leq i \leq n\})$

As $\sharp(Support_S(first_k(v))) = \sharp(Support_S(\lceil first_k(v) \rceil_S))$ and $\sharp(Support_S(v) \setminus \{(i,1) \mid 1 \leq i \leq n\}) = \sharp(Support_S(v)) - \sharp(Support_S(v) \cap \{(i,1) \mid 1 \leq i \leq n\})$, determining if $\text{next}(v)$ is left extensible takes constant time. To conclude, as for any initial node v, we can compute in $O(1)$ its set of successors $\text{Succ}(v)$, its right extensibility, and the left extensibility of its single successor, we obtain a complexity that is linear in the size of DBG_k^+, since each successor is accessed only once. This yields Theorem 12.

Theorem 12 *For a set of words S, building the Contracted de Bruijn Graph of order k, $CDBG_k^+$ takes linear time and space in $|T|$ or in $\|S\|$.*

5 dBG and CdBG from Suffix Array

For lack of space the reader is referred to [4] for the full details.

Theorem 13 *The dBG of order k, $CDBG_k^+$, for a set of words S can be built in a time and space that are linear in $\|S\|$ using the generalised suffix array of S.*

6 Dynamically updating the order of $DBG_\cdot^+$

Genome assembly from short reads requires to test multiple values of k for the dBG. Indeed, the presence of genomic repeats, makes some order k appropriate to assemble non repetitive regions, and larger orders necessary to disentangle (at least some) repeated regions. Combining assemblies obtained from DBG_k^+ for successive values of k is the key of `IDBA` assembler, but the dBG is rebuilt for each value [13]. Other tools also exploit this idea [2]. It is thus interesting to dynamically change the order of the dBG. Here, we argue that starting the construction from an index instead of the raw sequences ease the update. On page 7, we mention which information are needed in general to build DBG_k^+. Assume the words are indexed in a suffix tree T (as in Section 4.2). Consider changing k to $k-1$. First, only the nodes of $Init_{S,k}$ whose parent represents a word of length $k-1$ are substituted by their parent in DBG_{k-1}^+, all other nodes remain unchanged. Thus, any arc of order k either stays as such or has some of its endpoints shifted toward the parent node in T. In any case, updating an arc depends only on the nature of its nodes in DBG_{k-1}^+ (whether they belong to $Init_{S,k-1}$ or $InitExact_{S,k-1}$), and can be computed in constant time.

The same situation arises when changing k to $k+1$. First, only nodes of $InitExact_{S,k}$ change in DBG_{k+1}^+: they are substituted by their children. Updating an arc also depends

on the nature of its nodes: it can create a fork towards the children of the destination node if the latter changes, or it can be multiplied and join each children of the source to one children of the destination if both nodes change. Then, the label of the children in T indicate which children to connect to. It can be seen that updating from DBG_k^+ to DBG_{k+1}^+ in either direction takes linear time in the size of T. Moreover, as updating the support of nodes in T is straightforward, we can readily apply the contraction algorithm to obtain CDBG_{k+1}^+ (see Section 4.3).

References

[1] A. Apostolico. The myriad virtues of suffix trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO Advanced Science Institutes, Series F*, pages 85–96. Springer, 1985.

[2] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, and et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012.

[3] A. Bowe, T. Onodera, K. Sadakane, and T. Shibuya. Succinct de Bruijn Graphs. In *WABI*, volume 7534 of *LNCS*, pages 225–235, 2012.

[4] B. Cazaux, T. Lecroq, and E. Rivals. From Indexing Data Structures to de Bruijn Graphs. Technical report, lirmm-00950983, Feb. 2014.

[5] R. Chikhi, A. Limasset, S. Jackman, J. Simpson, and P. Medvedev. On the representation of de Bruijn graphs. *ArXiv e-prints*, Jan. 2014.

[6] R. Chikhi and G. Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms for Molecular Biology*, 8:22, 2013.

[7] T. C. Conway and A. J. Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–486, 2011.

[8] N. de Bruijn. On bases for the set of integers. *Publ. Math. Debrecen*, 1:232–242, 1950.

[9] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, Cambridge, 1997.

[10] U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993.

[11] T. Onodera, K. Sadakane, and T. Shibuya. Detecting superbubbles in assembly graphs. In *WABI*, volume 8126 of *LNCS*, pages 338–348. 2013.

[12] J. Pell, A. Hintze, R. Canino-Koning, A. Howe, J. Tiedje, and C. Brown. Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. *Proc. Natl Acad. Sci. USA*, 109(33):13272–13277, 2012.

[13] Y. Peng, H. Leung, S. Yiu, and F. Chin. IDBA A Practical Iterative de Bruijn Graph De Novo Assembler. In *RECOMB*, volume 6044 of *LNCS*. 2010.

[14] P. Pevzner, H. Tang, and M. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc. Natl Acad. Sci. USA*, 98(17):9748–9753, 2001.

[15] E. A. Rødland. Compact representation of *k*-mer de Bruijn graphs for genome read assembly. *BMC Bioinformatics*, 14:313, 2013.

[16] L. Salmela. Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, 26(10):1284–1290, 2010.

[17] J. T. Simpson and R. Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, 26(12):i367–i373, 2010.