ON THE RELATIONSHIP BETWEEN DE BRUIJN AND VARIATIONS GRAPHS

Advanced Algorithms

Santi Enrico
University of Udine
Academic year 2023/24

Contents

1 Introduction **3**

2 Problem presentation **3**
2.1 String graphs . . . 3
2.1.1 Subpath-comaptible extension of l . . . 3
2.1.2 How to represent string sets . . . 4
2.2 De Bruijn graphs . . . 5
2.3 Variation graphs . . . 6

3 The transformation procedure **7**
3.1 The algorithm . . . 7

4 Conclusions **10**

Appendices **11**

A Examples and wrong alternatives **11**
A.1 An alternative . . . 12

B From singular variation graphs to Pdor's algorithm **14**
B.1 A single string . . . 14
4.2 To multiple strings and beyond . . . 16

1 Introduction

Throughout the years several data structures have been described to effectively represent collections of strings allowing specific operations on such strings to be performed efficiently[1]. The objective of this work is to provide an introduction to de Brujin and variation graphs, give an overview of the work presented in [1] and try to extend such work by giving more insights on the desired properties of such graphs and possibly how we can derive graphs with those properties. The context on which we are focusing is the one of *graph pangenome*[2] [2].

2 Problem presentation

Consider the problem of transforming a data structure into a different one while preserving different characteristics of the data. Such problem can be important for several reasons, for example because a data structure can be more efficiently used to perform certain operations, while another one may allow for an easier and more clear representation of the information. For this reason we may be interested in switching between the two representations. The contribution of [1] was to provide a procedure for transforming a de Bruijn graph into a variation one (while also axiomatizing the description of particular class of variation graphs).

2.1 String graphs

As in [1] we start by introducing the general idea used to represent a set of strings via a (string) multi-graph, such graphs will encode the commonalities and differences among a collection of genomes [2].

Fixed an alphabet Σ, a string multi-graph is a triple $G = \langle V, E, l \rangle$ where V and E are respectively a set of nodes[3] and directed edges, while $l : V \longrightarrow \Sigma^+$ is a labelling function which given a node returns a non empty string. Such function is used to associate strings to nodes, though strings represented by a single node are usually just a small substring of a string $s_i \in S$, (where $S = s_1..s_n$ is the of strings we want to depict). For this reason the function l must be *extendable* in order to capture strings represented by paths[4] in the graph.

2.1.1 Subpath-comaptible extension of l

Before jumping into the extension of l, namely $\hat{l}$ some basic notions on paths must be recalled.

Given a path $p = \langle v_1, ..., v_n \rangle$ its length describes the number of edges traversed by the path (i.e. $n - 1$ for a path with n nodes) [3]. The set of intervals of a given path p, are all the couples of indexes denoting two nodes in the path $Int(p) = \{\langle i, j \rangle | 1 \leq i \leq j \leq n\}$. The last notion needed is the one of subpath of p, a subpath defined over $\langle i_1, i_2 \rangle$ is a path contained in p which starts at node v_{i_1} and ends at v_{i_2}, it's denoted by $p[i_1...i_2] = \langle v_{i_1}, ..., v_{i_2} \rangle$.

[1] An example is the suffix tree which allows to represent in linear space all the suffixes of a string, allowing to perform an efficient search for the LCS (longest common substring).

[2] A graph-based representation of multiple genomes.

[3] Or Vertices, though we will always refer to them as nodes from now on.

[4] An ordered sequence of adjacent nodes.

We can now define $\hat{l}$ (parameterized by p) as the function that given a subpath of p returns the string denoted by that subpath, such fucntion satisfies the property (*subpath-compatibility*):

$$\hat{l}(p[i_1..i_2]) = \hat{l}(p)[i'_1..i'_2]$$

2.1.2 How to represent string sets

Given a set of strings $S = \{s_1, ..., s_n\}$ what characteristics a string graph $G = \langle V, E, l\rangle$ has to have in order to represent S?
There must exists a function $\pi : S \longrightarrow \mathcal{P}(G)$ (where $\mathcal{P}(G)$ is the set of all paths in G) such that:

- $\hat{l}(\pi(s_i)) = s_i \quad \forall i \in \{1, ..., n\}$, which means that the string we read by traversing the path describing s_i is s_i itself. Note that this property is central in describing S. If this property wasn't required then a string graph would need to indicate also a set of paths denoting the genomes $\{s_1..s_n\}$ we want to describe in the graph, since the graph could present paths describing genomes not in $\{s_1..s_n\}$.
- $\forall v \in V$, $\exists i : v \in \pi(s_i)$, meaning that we don't want G to have nodes which aren't present in any useful path describing a string.
- A characteristic similar to the previous one but for the edges. For each edge in G it must join two consecutive nodes in some $\pi(s_i)$. Again notice that this requirement alone doesn't imply that G depicts only the strings in S.

The three properties mentioned above are necessary for representing a collection of strings, indeed $\langle G, \pi\rangle$ represents $S \iff$ the mentioned properties hold.
On the other hand there are also additional properties that deal with k-mers (strings of length k) we'd like our graph representation to satisfy:

k-completeness: *$\langle G, \pi\rangle$ is k-complete if every k-mer in S is represented by the same path in G.*
More formally, given $s_i, s_{i'} \in S$ which share a substring of length k (k-mer), i.e. $s_i[j..(j+k-1)] = s_{i'}[j'..(j'+k-1)]$, $\pi(s_i)$ and $\pi(s_{i'})$ share a part of the path of the same length which depicts the k-mer. Thus we must have that $\pi(s_i)[p..(p+q)] = \pi(s_{i'})[p'..(p'+q)]$ and $\hat{l}(\pi(s_i)[p..(p+q)]) = s_i[j..(j+k-1)]$. This has to be verified for all k-mers.

k-faithfulness: A formal description requires to introduce the notion of k-extendability. Consider two strings s_i,$s_{i'}$ and two nodes indexes (which refer to the same node v) in their paths, namely j in $\pi(s_i)$ and j' in $\pi(s_{i'})$, such that $\pi(s_{i'})[j'] = \pi(s_i)[j] = v$. We say that the pair $\langle i, j\rangle$ $\langle i', j'\rangle$ is *directly k-extendable* iff $\pi(s_i)[(j-m)..(j+m')] = \pi(s'_i)[(j'-m)..(j'+m')]$ for $m, m' \geq 0$ with the length of the string associated to that subpath is greater or equal than k.
The meaning of direct k-extendability is that two strings share a common node in their path iff they share a substring longer than k, otherwise, the node is duplicated.
There is then the notion of *k-extendability* which generalizes the previous one, namely

$\langle i,j \rangle$ $\langle i',j' \rangle$ if it exists a sequence of occurrences of v, from $\langle i,j \rangle$ $\langle i',j' \rangle$ such that each consecutive occurrence in the sequence is k-extendable.

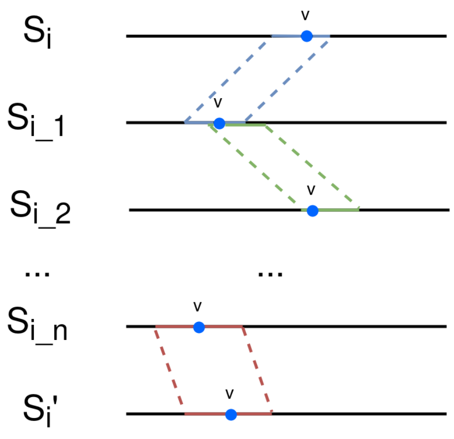

Figure 1: k-extendability for $\langle i,j \rangle$ $\langle i',j' \rangle$, occurrences of v.

The idea behind the k-extendability is that given two nodes depicting the same string in two paths describing strings, that two nodes must be the same node only if all strings share a common substring around that node of length longer than k (see Appendix A).

$\langle G,\pi \rangle$ is k-faithful for S if all pairs $\langle i,j \rangle$ $\langle i',j' \rangle$ of the same nodes are k-extendable. The meaning of k-faithfulness is then to limit when nodes depicting the same substring can be merged into a single one.

We will now focus on two concrete types of string graphs, namely de Bruijn and variation graphs.

2.2 De Bruijn graphs

Fixed an integer k, we define a de Bruijn graph of length k as a string graph with the following conditions:

- $|l(v)| = k \; \forall v \in V$, the length of the string described by each node is a k-mer.
- $l(v) = l(w) \implies v = w \;\; \forall v, w \in V$, in a de Bruijn graph there are no repeated nodes. This condition, implies that if all strings in S are longer than k the representation is k-complete (and k-faithful).
- $l(v)[2..k] = l(w)[1..(k-1)] \;\; \forall (v,w) \in E$, which indicates that v has an outgoing edge to w only if the last $k-1$ characters describing v are equal to the prefix of length $k-1$ of the string describing w.

We now need to describe $\hat{l}$ which, given a path $p = \langle v_1, .., v_n \rangle$ is defined as the concatenation of the k-mer described by v_1 and the last character of the k-mer describing each node v_i, namely $\hat{l}(p) = l(v_0) \cdot l(v_1)[k] \cdot ... \cdot l(v_n)[k]$.

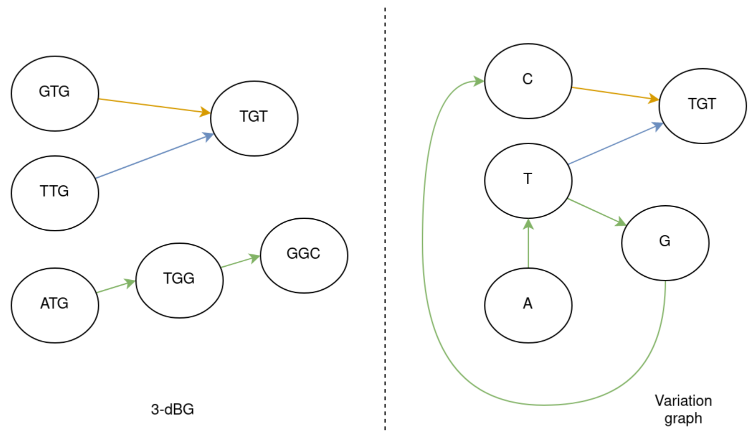

Figure 2: A comparison between *the* 3-dBG graph (3-complete and 3-faithfull) and *a* 3-complete varaition graph for $S = \{GTGT, TTGT, ATGGC\}$

One important choice when dealing with dBGs is the choice of the parameter k, since this can effectively impact the size of G. This choice can also influence the efficiency of the operations performed on G, for example, if k-mer lengths fit in a machine word, bitwise operations and integer arithmetic can be performed on strings [4].
We conclude this section by recalling a Theorem from [1] which states the *uniqueness (up to isomorphism) of a k-de Bruijn graph for a set S* if each string is longer than k.

2.3 Variation graphs

A different way to concretize the notion of a string graph is presented by the variation graphs. A variation graph is a triple $G = \langle V, E, l \rangle$, but in contrast to dBGs $|l(v)|$ doesn't need to be constant and the generalization of l, $\hat{l}$ is the concatenation of the labels of each node in the path.
The subpath-compatibility of $\hat{l}$ still holds, consider a path $p = \langle v_1, ..., v_n \rangle$ and suppose we are interested in the label spelled out by the nodes in between node v_{i_1} and v_{i_2}, namely $\hat{l}(p[i_1..i_2])$. Since we have defined $\hat{l}$ to be the concatenation of the labels, we observe that the string read by the subpath $p[i_1..i_2]$ is the same as the one read by considering the string on the whole path and then skipping the first characters of the corresponding to the $0..i_1 - 1$ nodes, and early stopping after having read the characters up to the ones of v_{i_2}:

$$\hat{l}(p[i_1..i_2]) = \hat{l}(p)[\sum_{i=1}^{i_1-1} |l(v_i)| + 1 \,..\, \sum_{i=1}^{i_2} |l(v_i)|]$$

We conclude this brief subparagraph by recalling that two variation graphs representations on a set S are equivalent if they present the same k-mers, $\forall k \geq 1$. In addition if given S two variation graphs representations $\langle G, \pi \rangle$ $\langle G', \pi' \rangle$ are both k-complete and k-faithful, then they are equivalent.

3 The transformation procedure

Having introduced dBGs and variation graphs, we now describe a procedure that given a dBGs representation $\langle G, \pi \rangle$ for S returns a corresponding variation graph representation. Such algorithm adopts three subprocedures and uses an *intermediate graph representation*, called transition graphs, to store the partial results of the subprocedures.

In a nutshell, fixed an alphabet Σ (which is related to the set S of strings we want to describe) a transition graph is a triple $G = \langle V, E, l \rangle$ where the nodes now are used to denote single characters in Σ (i.e. $l : V \longrightarrow \Sigma$) and $E := E_V \cup E_B$. E_V and E_B are disjoint sets, the latter is the set of *de Brujin edges* (B-edges) while the former is the set of *variation edges* (V-edges).

Before diving into the details of the three procedures we present a simple overview of the algorithm itself in Figure 3.

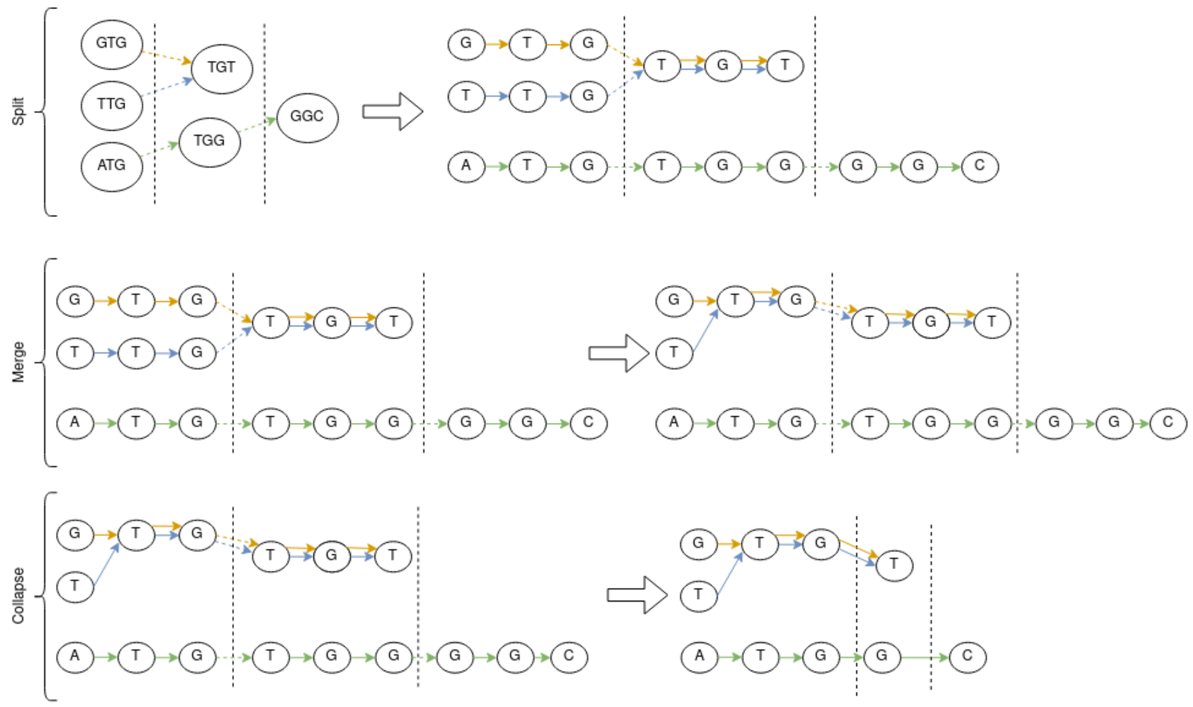

Figure 3: The results of the application of each procedure (in sequence) to the 3-dBG of Figure 2. On the left of each arrow the input structure for the corresponding procedure is found, while on the right the output is shown.

3.1 The algorithm

Having in mind the overview of the algorithm depicted in Figure 3 let's now dive into the procedures. Given in input the dBG representation the following procedures are invoked:

- **Split:** $\langle G, \pi \rangle \longrightarrow \langle G' = \langle V', E'_B, E'_V, l' \rangle, \pi' \rangle$, is the procedure that transforms the dBG in input into a transition graph. The idea is to split all the nodes representing k-mers (which we will call *k-nodes*) into linear subgraphs in which each node represents a single character (*single nodes*)[5]. These subgraphs are connected by B-edges, while the nodes composing them are connected by V-edges. As reported in [1] we have that $\langle G', \pi' \rangle$ represents S k-completely, k-faithfully and the consistency is maintained (i.e. the same strings are represented), the proof is immediate. More formally G' is defined as:

  - $V' = \bigcup_{v \in V} \{v_1, ..., v_k\}$, namely the new (single) nodes are all the nodes we obtain by splitting each k-node v into it's k components.
  - $E_V = \bigcup_{v \in V} \{\langle v_1, v_2 \rangle, ..., \langle v_{k-1}, v_k \rangle\}$, the set of the V-edges (which previously was the empty set since we had a dBG) consists in the union of all the edges connecting the k single nodes we obtained by splitting the original k-nodes.
  - $E_B = \{\langle v_k, w_1 \rangle | \langle v, w \rangle \ \in E\}$, the set of the B-edges is restricted to those edges which connect single nodes that were obtained as the first single node and last single node as a result of the split procedure on two connected k-nodes.
  - $l'(v_i) = l(v)[i] \quad \forall v \in V, j = 1..k$, namely the string (character) the new label function associates to a single node obtained as the $i - th$ node by the splitting of a k-node v, is the character in position i of the string obtained by the labelling function l on v.

- **Merge**: $\langle G' = \langle V', E'_B, E'_V, l' \rangle, \pi' \rangle \longrightarrow \langle G'' = \langle V'', E''_B, E''_V, l' \rangle, \pi'' \rangle$[6]. The idea of this procedure is to merge (thus eliminating) redundant nodes introduced by the previous procedure. Redundant in this context means that given two linear subgraphs $\langle V = \{v_1, .., v_n\}, E = \{\langle v_1, v_2 \rangle, .., \langle v_{n-1}, v_n \rangle\} \rangle$ and $\langle V = \{v'_1, .., v'_n\}, E = \{\langle v'_1, v'_2 \rangle, .., \langle v'_{n-1}, v'_n \rangle\} \rangle$, sharing a parent[7] node and presenting single nodes with the same label (character) for each node. The function π is modified such that whenever a string s_i was mapped into a path p_i containing one merged node v_j, now s_i is mapped into p'_i in which the occurrences of v_j are replaced by those of the node resulting from the merging.

  Also this transformation preserves the consistency, k-completeness and k-faithfulness.

- **Collapse**: $\langle G'' = \langle V'', E'_B, E'_V, l' \rangle, \pi' \rangle \longrightarrow \langle G_{Var}, \pi_{Var} \rangle$, represents the last step of the algorithm and for this reason the output of this procedure must be a variation graph, not a transition graph. Having understood this, it's clear that during this phase all the B-edges will be removed. Consider a B-edge (let's call it E since it marks the *End* of a former k-node), by the consistency we have that the considered B-edge is followed and preceded by a linear subgraph describing a path of length k-1, see Figure 4.

  The key is to notice that these two linear subgraphs, which we will call B (for the subgraph coming *Before* the edge considered) and A (for the one coming *After* the B-edge) contain a sequence of nodes in which each pair $\langle A[i], B[i] \rangle$[8] for $i = 1..k-1$ satisfies $l'(A[i]) = l'(B[i])$. This means that we have two linear graphs whose nodes (ordered) are mapped to the same letter by l. The reason for this is that A and B are

[5]In [1] graphs in which all nodes are labelled with only a single lablel are referred as singular graphs.

[6]Note that l' isn't modified.

[7]A node which has an ingoing B-edges in both subgraphs

[8]We depict linear subgraphs as arrays, so $A[i]$ indicates the $i - th$ node of A.

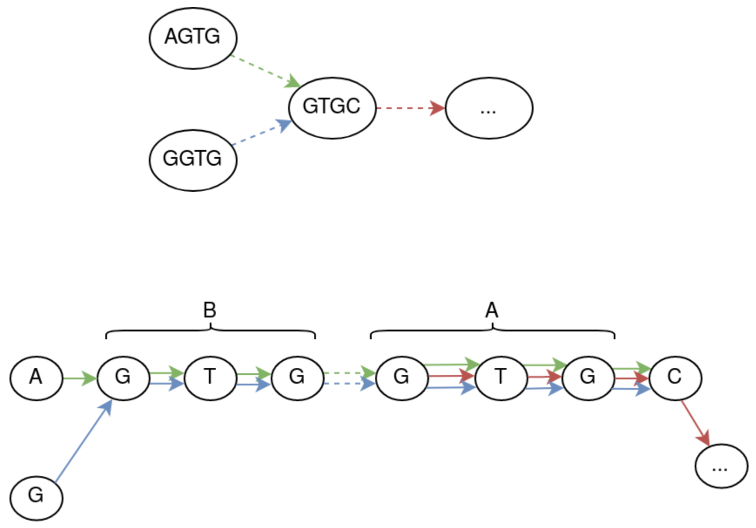

Figure 4: How the 4-dBG on top is transformed (after the Merge operation, before the Collapse) into a transition graph, highlighting the linear graphs A and B before and after the B-edge.

linear graphs resulting from the split of two adjacent k-dDB nodes. By definition of a k-dDB, we have that two adjacent nodes share a prefix and suffix of length $k-1$. By realizing this, it follows that the resulting variation graph shouldn't present the first $k-1$ nodes of A since the string represented by them has already been kept into account by reading the last $k-1$ nodes of B. The collapse procedure works by cases:

– **If G'' doesn't contains a cycle including a B-edge in which the subgraphs A and B overlap** then the actions to perform are the following: merge each pair $\langle A[i], B[i]\rangle$ into a node $C[i]$ and redirect all the incident edges of the pair on $C[i]$ after removing the B-edge (this avoids the creation of a cycle of length two involving E).

  The last step is to modify π' and $\hat{l}$ accordingly, i.e. consider each string $s \in S$ such that s was described by a path containing B and A, replace that part (i.e. B and A) by C [9].

– **Otherwise** it means that there exists a value $t \in 1..k-1$ for which $B[i] = A[t+i]\ \forall i \in 1..k-1$. In such case in addition to the operations performed in the previous case we merge the nodes $C[n]...C[2t+n]$ for $n = 1..t-1$. Also in this case π' should be slightly modified accordingly.

For what concerns the termination of the algorithm, it's based on the fact that each sub-procedure terminates (since Split and Merge only iterate over all nodes once, while Collapse either removes a B-edge at each iteration, or stops). The correctness is also guaranteed [1].

[9]The fact that a path that traversing B also traverses A comes from the idea that these used to represent the common $k-1$ characters of two adjacent k-nodes

4 Conclusions

The algorithm presented provides a relationship between k-complete and k-faithful variation graphs and k-dBGs, allowing among other things to transfer data between pangenome models based on different representations. In Appendix A the original part of the work done is presented, insights and equivalent formulations of the notions found in Section 2 are given.

Appendices

A Examples and wrong alternatives

Consider the string graph in Figure 7, by defining $\hat{l}$ as the concatenation of $l(v)$ on a given path, we have that the graph depicts the set $S = \{s_1, s_2, s_3, s_4\} = \{AAGC, AAGG, GAGT, GAGA\}$. Such string graph is 3-complete but not 3-faithful.

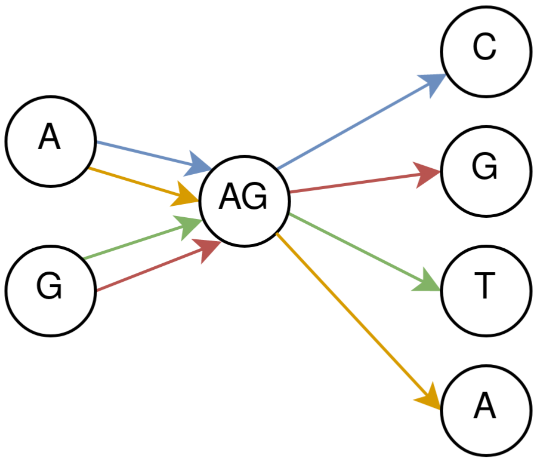

Figure 5: A 3-complete but not 3-faithful representation of S

The reason for which it's not 3-faithful is that the node "AG" shared by all four paths has four occurrencies: $\langle 1,2\rangle$, $\langle 2,2\rangle$, $\langle 3,2\rangle$,$\langle 4,2\rangle$, but not all pair of occurrencies share a 3-mer containing "AG" (e.g. $\langle 1,2\rangle$, $\langle 3,2\rangle$). In order to achieve the 3-faithfulness we need thus to duplicate the node "AG" as shown in Figure 6.

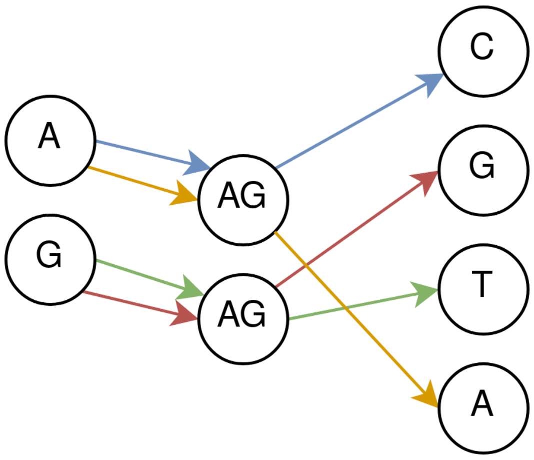

Figure 6: A 3-complete and 3-faithful representation of S

The idea of a graph representation being k-faithful and k-complete is thus a balance between trying to repeat less nodes[10] possible and not merging all the repeated substrings

[10]Nodes with the same label.

in S if the strings considered don't share at least a k-mer.

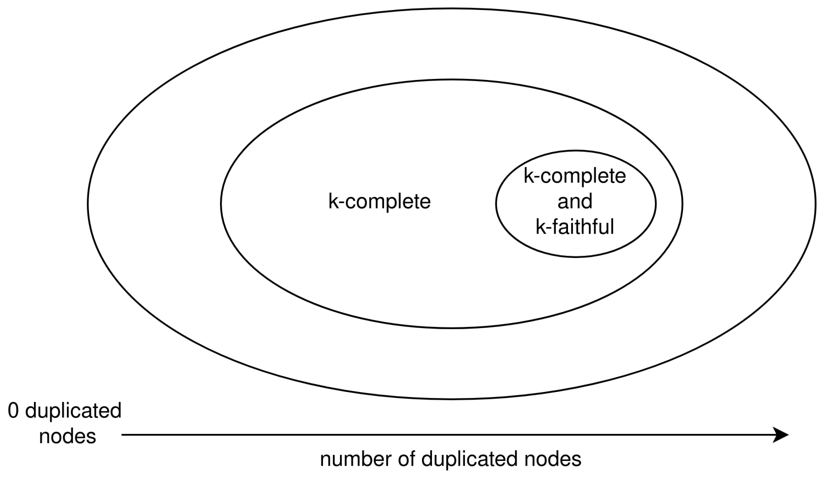

Figure 7: An overview of only the k-complete graphs (fixed k) for S. There's a boundary reachable by adding the correct duplicated nodes above which the representations become k-faithful. Note that if too much or the wrong nodes are added the graph representation could fall outside of the k-complete graphs.

A.1 An alternative

Playing around with the notion of k-extendability it could seem to be equivalent to the following definition.

Definition 1. *$\langle i, j\rangle$ $\langle i', j'\rangle$ are weakly k-extendable iff in the set of occurrences of v for each string s_l there's another string s'_l such that they are directly k-extendable.*

Thoguh, while being introduced to try to simplify the notion of *k-extendability* it turned out to be weaker, thus non equivalent. Indeed the notion of *weakly k-extendability* indicates that a node can be shared by multiple string paths (or by the same string in different positions) if at least two share a k-mer around that node.

Definition 2. *$\langle G, \pi\rangle$ represents S weakly k-faithully if every pair of a vertex is weakly k-extendable.*

Lemma A.1. *Let $\langle G, \pi\rangle$ be k-faithful then:*

- *$\langle G, \pi\rangle$ is k-complete.*
- *$\langle G, \pi\rangle$ is weakly k-faithful.*
- *$\langle G^-, \pi'\rangle$ is not k-faithful, where $G^- = G \setminus \{u : \exists u_1..u_r \in V \wedge l(u) = l(u_1) = .. = l(u_m)\}$ and π' redirects each in going edge on u to $u_1..u_r$ (not necessary they are all mapped on the same u_i), adding also the respective outgoing edges in order to describe the same set of strings.*

Proof. Given $\langle G, \pi\rangle$ k-faithful we first need to prove its also k-complete. Given two strings s_i and s'_i sharing a k-mer (since $\hat{l}(\pi(s_i)[j..j+m_1]) = \hat{l}(\pi(s'_i)[j'..j'+m_2])$) by k-extendability we have that $\pi(s_i)[j..j+m] = \pi(s'_i)[j'..j'+m]$, thus we have that the k-mer is described by

the same path ($m_1 = m_2$ and we substitute it by m). The same reasoning can be extended to more than two strings.
The proof of the second implication is immediate since given two or more vertices occurrences if they are k-extendable there they are also weakly k-extendable.
We now need to prove the third point. We know that in $\langle G, \pi \rangle$ G has at least a repeated node $(u, u_1, ..u_r)$, and suppose wlog that only the path related to the string s_i passes through u, namely $pi(s_i) = u_1, u_2..u_i..u_h$ (wlog assume it's not the first or last node). This means that the other paths of strings $s_{i_1}, .., s_{i_n}$ passing through $u_{r_1}..u_{r_u}$ don't share a k-mer containing $l(u)$ with s_i. We know that in $\langle G, \pi \rangle$ G has at least a repeated node $(u, u_1, ..u_r)$, and suppose wlog that only the path related to the string s_i passes through u, namely $pi(s_i) = u_1, u_2..u_i..u_h$ (wlog assume it's not the first or last node). This means that the other paths of strings $s_{i_1}, .., s_{i_n}$ passing through $u_{r_1}..u_{r_u}$ don't share a k-mer containing $l(u)$ with s_i. From this last statement and the completeness of G we can derive that $l(u) < k$ otherwise s_i and $s_{i_1}, .., s_{i_n}$ would share a k-mer. Removing u and replacing its occurrence in $\pi(s_i)$ by u_{r_1} i.e. creating $\pi'(s_i) = u_1, u_2..u_{r_1}..u_h$ makes the graph G^- not k-faithful since the strings in $s_{i_1}..s_{i_n}$ whose path passes through u_r don't share a k-mer around $l(u_r)$. □

Note that $\langle G, \pi \rangle$ being *k-faithful* doesn't imply that $\langle G^-, \pi' \rangle$ is not complete, (again see Figure 7 and 6). Let's now see a lemma stating the opposite implication (vice versa).

Lemma A.2. *Let $\langle G, \pi \rangle$ be k-complete. If $\langle G^-, \pi' \rangle$ is not k-faithful, where $G^- = G \setminus \{u : \exists u_1..u_m \in V \wedge l(u) = l(u_1) = .. = l(u_m)\}$ and π' redirects each in going edge on u to $u_1..u_r$ (not necessary they are all mapped on the same u_i), adding also the respective outgoing edges in order to describe the same set of strings, then it's NOT guaranteed for $\langle G, \pi \rangle$ to be k-faithful.*

Proof. Consider the following counterexample:

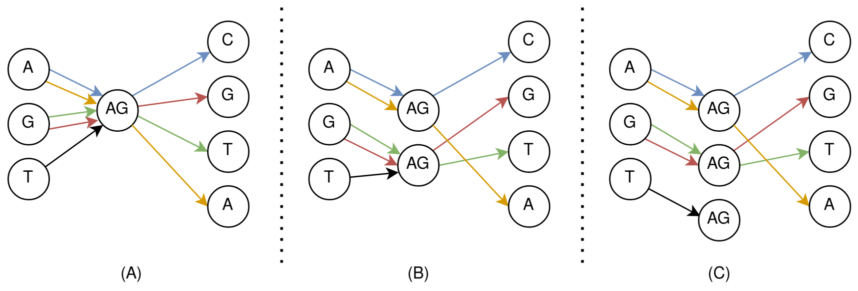

Figure 8: Duplicating the node "AG" twice to get a 3-faithful representation.

Consider the (B) string graph in Figure 8, it's 3-complete, and (A) is not 3-faithful, but (B) is not 3-faithful (the string "TAG" doesn't share a 3-mer with "GAGT" or "GAGA", though it shouldn't share a node with them). □

Lemma A.3. $\langle G, \pi \rangle$ *is weakly k-faithful* $\implies$ $\langle G, \pi \rangle$ *is k-complete*

Proof. Given two strings s_i and s'_i sharing a k-mer (since $\hat{l}(\pi(s_i)[j..j+m_1]) = \hat{l}(\pi(s'_i)[j'..j'+m_2])$) by directlty k-extendability we have that $\pi(s_i)[j..j+m] = \pi(s'_i)[j'..j'+m]$, thus we have that the k-mer is described by the same path ($m_1 = m_2$ and call it m). The same reasoning can be extended to more than two strings. □

Lemma A.4. *$\langle G = \langle V = \{v_1..\}, E\rangle, \pi\rangle$ representing S is weakly k-faithful iff:*

- *Each path $\pi(s_i)$ that shares a subpath $v_1..v_m$ with another path $\pi(s_j)$ implies s_i shares with s_j a string containing $\hat{l}(v_{m_1}..v_{m_l})$ longer or equal than k and vice versa.*

Proof. ($\implies$) Given $\langle G, \pi\rangle$ weakly k-faithful we know by lemma A.3 that $\langle G, \pi\rangle$ is k-complete. From k-completeness we have that each k-mer is reflected by a common subpath, now we need to prove the double implication (iff):

- ($\implies$) Consider a path $\pi(s_i) = v_{i_1}, v_{i_2}..v_{i_p}, v_{m_1}..v_{m_l}..$ sharing $v_{m_1}..v_{m_l}$ in position i_p+1 to $i_{p'}$ with $\pi(s_j) = v_{j_1}, v_{j_2}..v_{j_q}, v_m..v_{m'}..$, in position j_q+1 to $j_{q'}$. Since $\langle G, \pi\rangle$ is k-faithful, by k-extendability we have that $\pi(s_i)[i_p+1..i_{p'}] = \pi(s_j)[j_q+1..j_{q'}] \implies |\hat{l}(pi(s_i)[i_p+1..i_{p'}]) \geq k|$. The fact that $\hat{l}(pi(s_i)[i_p+1..i_{p'}])$ contains $\hat{l}(v_{m_1}..v_{m_l})$ comes from the fact that the positions i_p+1 to $i_{p'}$ are related to the nodes $v_{m_1}..v_{m_l}$.
- ($\impliedby$) Consider two strings, s_i and s_j sharing a string $sk = \hat{l}(v_{m_1}..v_{m_l})$ of length longer or equal than k. Since $\langle G, \pi\rangle$ is complete and $|sk| \geq k$ we have that sk is reflected by a common subpath, namely $\pi(s_i)[q..q+p] = \pi(s_j)[q'..q'+p]$. We now need to show $v_{m_1}..v_{m_l} = \pi(s_i)[q..q+p]$, so, for sake of contraddiction assume $v_{m_1}..v_{m_l} \neq \pi(s_i)[q..q+p]$ thus it must exists a different path $v_{i_1}..v_{i_t}$ shared by s_i and s_j describing $\hat{l}(v_{m_1}..v_{m_l})$. By k-completeness though (since $\hat{l}(v_{m_1}..v_{m_l}) \geq k$) we have that a $v_{m_1}..v_{m_l} = v_{i_1}..v_{i_t}$.

($\impliedby$) Now given a string graph representation $\langle G, \pi\rangle$ for S in which each path $\pi(s_i)$ sharing a subpath $v_{m_1}..v_{m_l}$ with another path $\pi(s_j)$ implies s_i shares with s_j a string longer or equal than k containing $\hat{l}(v_{m_1}..v_{m_l})$ and vice versa we want to show that G is weakly k-faithful. Consider a generic node v'_m in $v_{m_1}..v_{m_l}$ there will be at least two $\pi - occurrencies$ of such node, by the assumptions made, such occurrences are directly k-extendible. The reasoning can be generalized to three or more strings s_i, s_j sharing a subpath $v_{m_1}..v_{m'}..v_{m_l}$ and s_j s_h sharing a subpath $v_{l_1}, v_{l_2}..v_{m'}..v_{l_t}$. □

B From singular variation graphs to Pdor's algorithm

We now propose an alternative way[11] to derive, given $S = \{s_1..s_n\}$ a k-complete and k-faithful (singular) variation graph representation of S. Such singular graph could subsequently be compacted into a non singular variation graph. The algorithm we are going to present now is based on a suffix tree (slightly modified) and works in linear time with respect to $n = \sum_{s_i \in S} |s_i|$ if the size of k is a small enough constant (e.g. up to 100 characters) though no assumption is made on the alphabet size is $|\Sigma|$.

B.1 A single string

Let's first consider the case of $S = \{s_1\}$. We can very easily construct the singular (linear) variation graph by creating for each character in s_1 a singular node and concatenating them.

Now construct the suffix tree of $s_1\$$. The edges though don't contain just one starting and ending index of the string depicted by them, but all the starting and ending index in

[11]Respect with the one presented in [1]

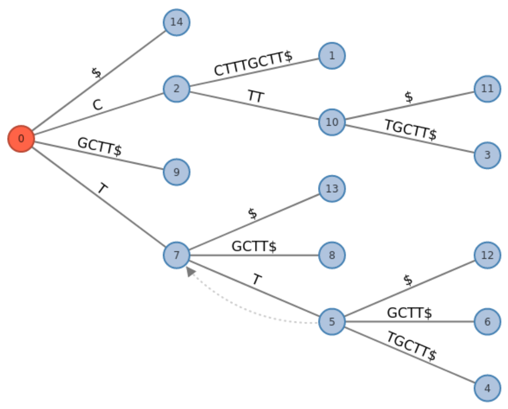

Figure 9: The suffix tree of $s_1\$$

which such occurrence is found[12].
To see which nodes merge we need to consider the graph nodes whose label depicts paths from a root to an internal node (with depth $\geq k$ considering the length of the strings on the edges) from which two or more leaves are reachable. Lastly merge such nodes, then the variation graph obtained is singular but $k-complete$ and $k-faithful$.

Consider as an example $s_1 = CCTTTGCTT$, in figure 9 its suffix tree is shown [13]. Suppose we want to derive the $3-complete$ and $3-faithful$ version of the singular variation graph. From such tree we see that the only labels in a path longer than three which leads to more leaves are C and TT. Such labels correspond to three nodes (C,T,T) repeated twice in the original variation graph (since the leaves are two). Having actually stored the ALL the indexes and not the strings on the edges on the tree[14] (the image depicts the strings for simplicity) we now know that we need to merge the two nodes C in position 1 and 6. The same is done for the nodes T, the edge TT indeed will be labelled by $[2,3],[7,8]$, which means that the nodes T in position 2 and 7 will be merged, and the nodes in position 3 and 8 will do the same.

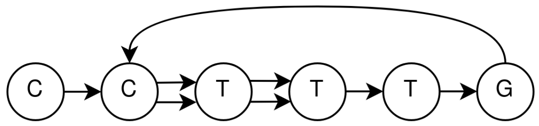

Figure 10: The resulting variation graph

It's immediate to see that the variation graph obtained by this procedure is $k-complete$: paths in the three depicting multiple paths in the graph depicting strings whose prefix longer or equal than k are merged together. It's also $k-faithful$ since no single nodes not sharing a $k-mer$ are merged. To this regard figure 10 may not be very immediate, but consider the equivalent graph obtained by merging the three nodes CTT into a single non singular

[12]To calculate this suffix tree Ukkonen's algorithm will still be used with a minimal modification that won't impact its runtime.

[13]Tool used: `https://brenden.github.io/ukkonen-animation/`

[14]I.e. instead of the label C we will have $[1,1],[6,6]$ assuming the indexes start from 0.

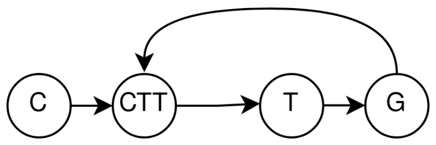

Figure 11: The resulting variation graph

node, figure 11, it's now immediate to see the $3 - faithful$.

4.2 To multiple strings and beyond

References

[1] A. Cicherski and N. Dojer, "From de bruijn graphs to variation graphs – relationships between pangenome models," in *String Processing and Information Retrieval*, F. M. Nardini, N. Pisanti, and R. Venturini, Eds. Cham: Springer Nature Switzerland, 2023, pp. 114–128.

[2] E. Garrison, "Computational graph pangenomics: a tutorial on data structures and their applications," 2022. [Online]. Available: https://link.springer.com/article/10.1007/s11047-022-09882-6

[3] M. Axenovich, "Lecture notes graph theory," 2014.

[4] E. Garrison, "Graphical pangenomics," 2019. [Online]. Available: https://www.repository.cam.ac.uk/handle/1810/294516