ON THE RELATIONSHIP BETWEEN DE BRUIJN AND VARIATIONS GRAPHS

Advanced Algorithms report

Santi Enrico
University of Udine
Academic year 2023/24

Contents

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 linking particular classes of the two families of graphs. The context on which we are focusin 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.

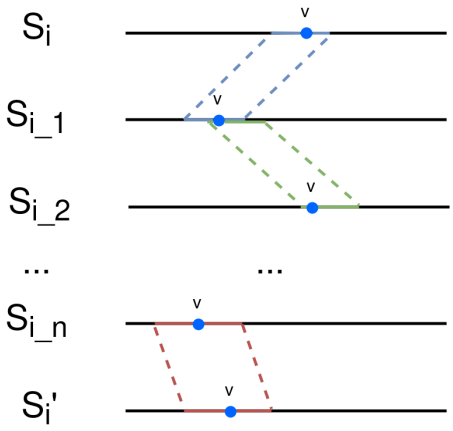

Figure 1: k-extendability for $\langle i, j\rangle$ $\langle i', j'\rangle$, occurrences of v.

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 $\hat{l}(p) = l(v_0) \cdot l(v_1)[k] \cdot ... \cdot l(v_n)[k]$.

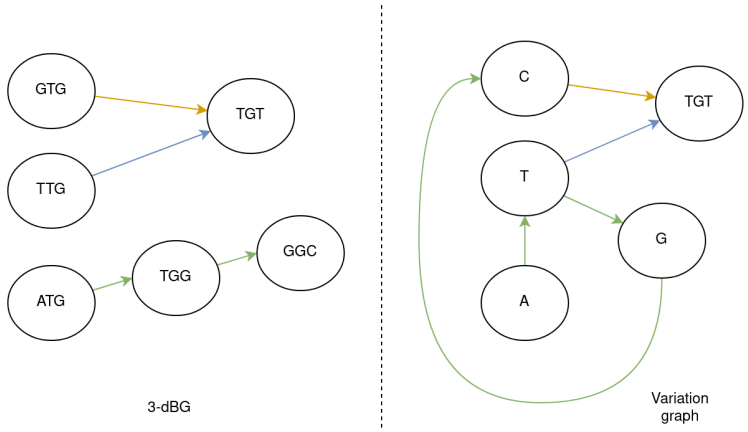

Figure 2: A comparison between *the* 3-dBG graph (3-complete and 3-faithfull) and *a* 3-complete varaition graph for $S = \{GTGT, TTGT, ATGGC\}$

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.

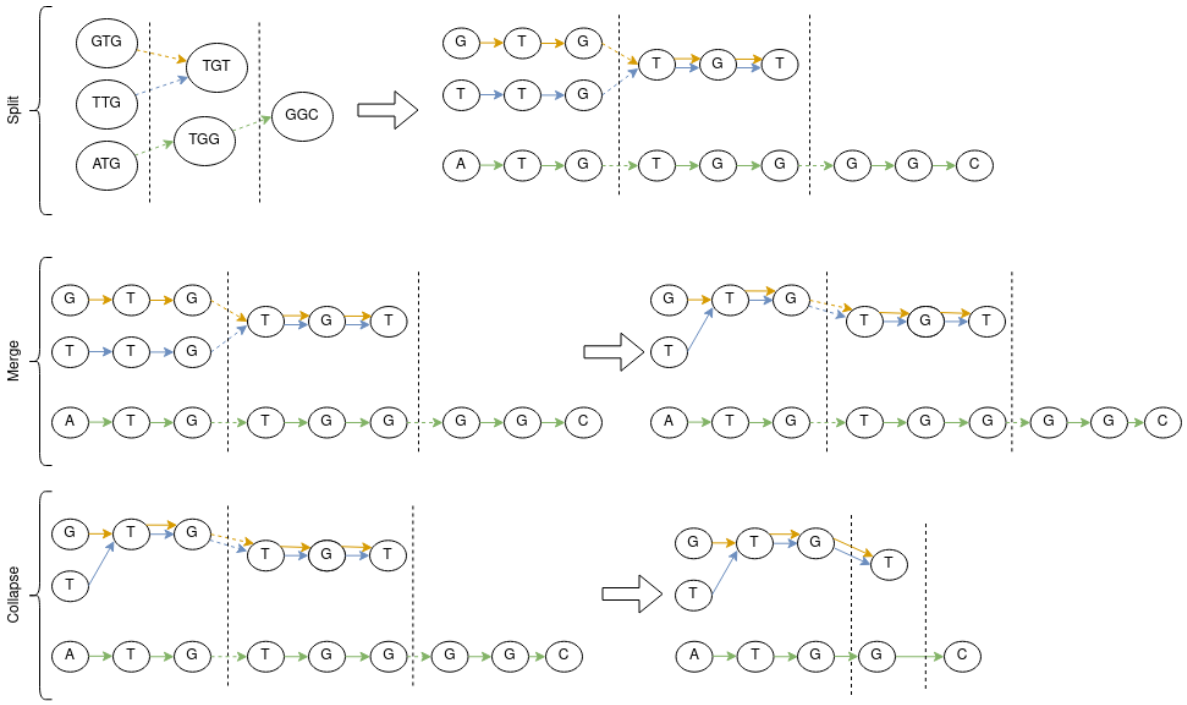

Figure 3: The results of the application of each procedure (in sequence) to the 3-dBG of Figure 2. On the left of each arrow the input structure for the corresponding procedure is found, while on the right the output is shown.

3 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.

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*). 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**: $\langle G' = \langle V', E'_B, E'_V, l' \rangle, \pi' \rangle \longrightarrow \langle G'' = \langle V'', E''_B, E''_V, l' \rangle, \pi'' \rangle$[5]. 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 node (which has an ingoing B-edges in both subgraphs), are made of single nodes and they present 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**: $\langle G'' = \langle V'', E'_B, E'_V, l' \rangle, \pi' \rangle\rangle \longrightarrow \langle G_{Var}, \pi_{Var} \rangle$, represents the last step of the algorithm and for this reason the output of this procedure is 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 $\langle A[i], B[i] \rangle$[6] 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 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 $k-1$.

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

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

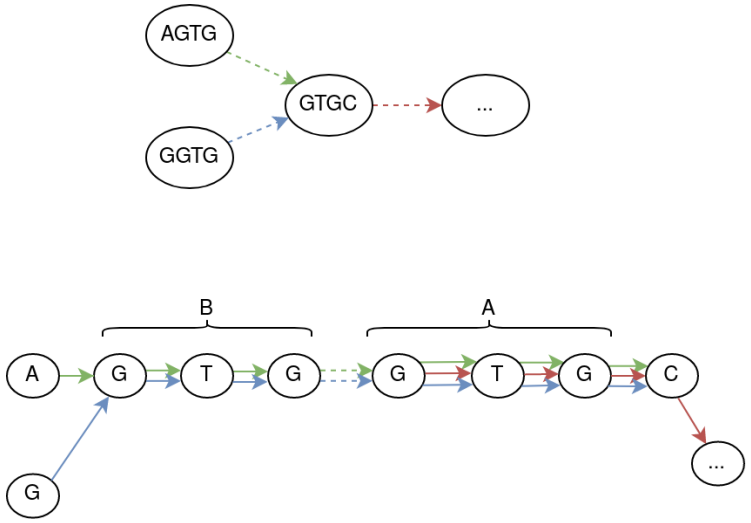

Figure 4: How the 4-dBG on top is transformed (after the Merge operation, before the Collapse) into a transition graph, highlighting the linear graphs A and B before and after the B-edge.

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. This procedure works by cases:

- **If G'' doesn't containing 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 pi' 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 [7].
- **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 subprocedure 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].

4 (Partial) 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.

[7] 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

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.

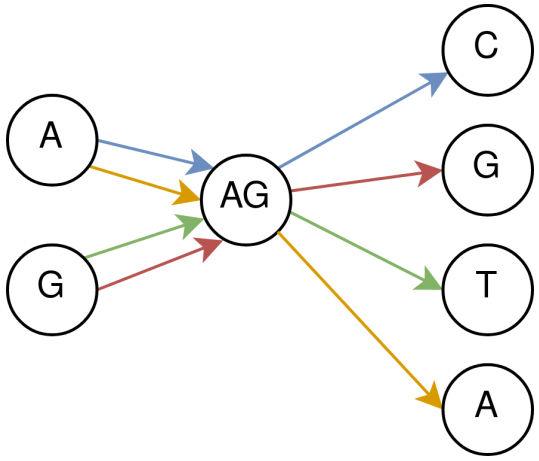

Figure 5: A 3-complete but not 3-faithful representation of S

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.

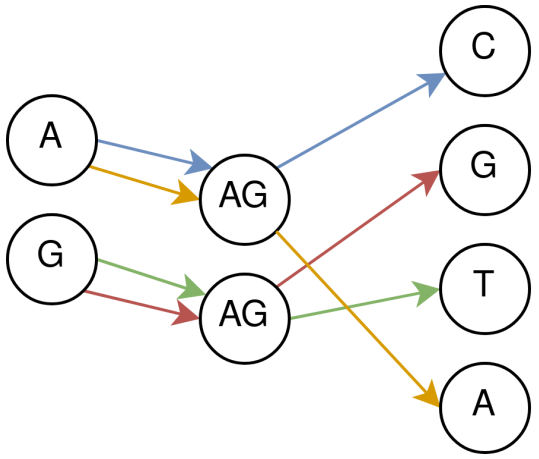

Figure 6: A 3-complete and 3-faithful representation of S

The idea of a graph representation being k-faithful and k-complete is thus a balance between trying to repeat less nodes[8] possible and not merging all the repeated substrings

[8]Nodes with the same label.

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

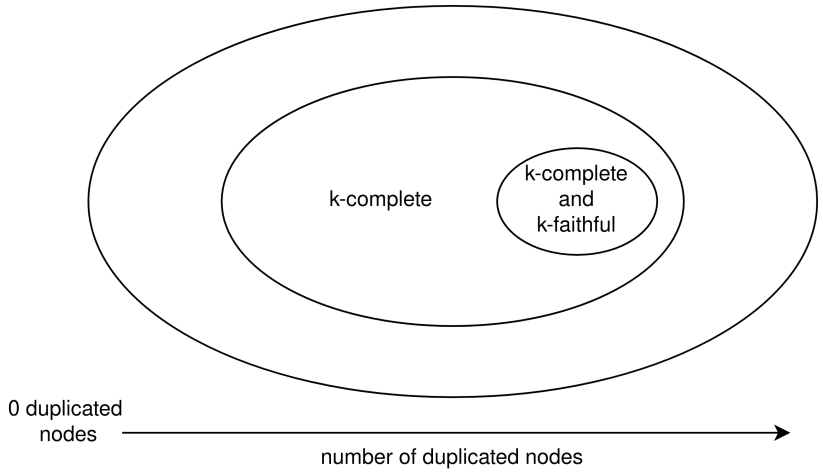

Figure 7: An overview of only the k-complete graphs (fixed k) for S. There's a boundary reachable by adding the correct duplicated nodes above which the representations become k-faithful. Note that if too much or the wrong nodes are added the graph representation could fall outside of the k-complete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

A.1 An alternative

Trying to play around with the notion of k-extendability it could seem to be equivalent to the following definition:
$\langle i, j\rangle$ $\langle i', j'\rangle$ *are weakly k-extendable if 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.* 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.

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