



**HAL**  
open science

## Sneak peek at the -tig sequences: useful sequences built from nucleic acid data

Camille Marchet

► **To cite this version:**

Camille Marchet. Sneak peek at the -tig sequences: useful sequences built from nucleic acid data. 2022. hal-03768446v1

**HAL Id: hal-03768446**

**<https://hal.science/hal-03768446v1>**

Preprint submitted on 3 Sep 2022 (v1), last revised 13 Sep 2022 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Sneak peek at the -tig sequences: useful sequences built from nucleic acid data.

Camille Marchet<sup>1\*</sup>

<sup>1</sup>Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

\*To whom correspondence should be addressed;

E-mail: camille.marchet@univ-lille.fr.

## Abstract

This manuscript is a tutorial on -tig sequences that emerged after the name "contig", and are of diverse purposes in sequence bioinformatics. We review these different sequences (unitigs, simplitigs, monotigs, ommitigs to cite a few), give intuitions of their construction and interest, and provide some examples of applications.

## 1 Introduction

The first appearance of a -tig sequence seems to be from Staden [1980], which quotes: "In order to make it easier to talk about data gained by the shotgun method of sequencing, we have invented the word 'contig'[...]" . Contig then denotes contiguous sequences, a term which will be used later in sequence assembly papers. Indeed, any -tig denomination has been created to describe some properties of a nucleotidic sequence (although some of the sequences presented below do not have the -tig suffix, but still fall in the scope). These properties are twofold, since -tigs can be of interest on the computer science side and on the biological side. -Tigs are helpful for computational sequence analysis since many of them offer graph properties or help with compression. They can also represent relevant biological units providing more genomic context than the smallest used substrings in indexing and assembly ( $k$ -mers). This manuscript aims at gathering and presenting a collection of concepts and ideas related to -tig sequences, that are currently used and under active research in sequence bioinformatics.

In the following we formalize our framework and give some important definitions. We work on finite strings (see Box 1.c) such as reads, genomic or transcriptomic sequences, genomes, over the genomic alphabet  $\Sigma = \{A, C, G, T\}$  (if needed we replace  $U$  by  $T$  for convenience).

**Definition 1  $k$ -mer.** A  $k$ -mer is a substring (see Box 1.c) of length  $k$  from a given string of length  $s \geq k$ , i.e.,  $k$  consecutive nucleotides extracted from a position  $p$  such that  $0 \leq p \leq s - k + 1$ .

### Box 1. Graph & string notions

**a. Graph.** A graph is a couple of two sets  $(\mathcal{V}, \mathcal{E})$ ,  $\mathcal{V}$  is a set of *nodes* and a  $\mathcal{E}$  is a set of *edges*. An edge connects a sink node to a source node. A subgraph is a selection of edges and nodes from a given graph. A *directed graph* has orientation in its edges, represented by an arrow.

**b. Paths, Walks.** We will use the term *path* to denote a finite sequence of edges that joins a sequence of distinct nodes. We will call a *walk* a similar sequence in which a node can be traversed several times.

**c. Strings.** A *string* is an ordered sequence of characters over a given alphabet. For a string  $S$  of size  $l$ , we will call  $T$  a *substring* of  $S$  when  $T$  is a string made of all  $S$ 's characters within two indices  $0 \leq i \leq j < l$  kept in the same order than in  $S$ .

Another important concept related to  $k$ -mer sets is the de Bruijn graph (see Box 1 for fundamental graph notions). It is a well known object in the assembly field, whose task is to reconstruct a genome from sequencing data. The de Bruijn graph serves as a fundamental structure for the assembly of second generation sequencing data thanks to its efficiency in representing of  $k$ -mer sets.

The rest of the presentation will be quite graph-oriented, but it is interesting to notice that ideas presented in the manuscript have twin concepts in stringology. We will mention a few of them.

Although there is no standard definition of de Bruijn graphs in bioinformatics, here we propose one which will be useful to introduce other concepts of the paper.

**Definition 2 Node-centric de Bruijn graph (dBG) in bioinformatics.** Given an input multiset of nucleotidic sequences  $S$  on  $\Sigma^*$ , the de Bruijn graph is a directed graph  $G_k(S) = (\mathcal{V}, \mathcal{E})$  where  $\mathcal{V}$  is a set of nodes and  $\mathcal{E}$  a set of directed edges.  $\mathcal{V}$  is the set of  $k$ -mers of  $S$ . For  $x, y \in \mathcal{V}$ , an edge  $(x, y) \in E$  if and only if  $x[2, k] = y[1, k-1]$ . In the following, we will simply call such structure a de Bruijn graph.

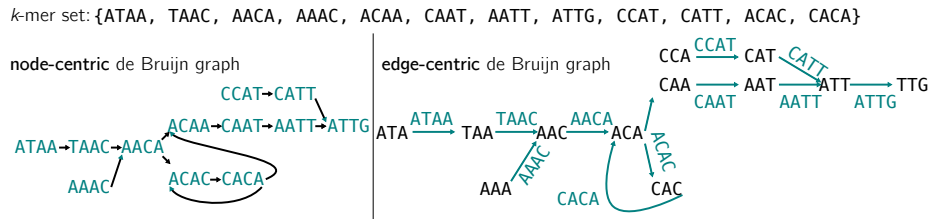


Figure 1: A  $k$ -mer set (top), the corresponding de Bruijn graphs ( $k=4$ ,  $k$ -mers shown in blue). Nodes contain strings, oriented edges are represented using arrows that can be labeled. We show a node-centric representation (left,  $k$ -mers are nodes and overlaps are implicit in edges) and an edge-centric representation (right,  $k$ -mers are labels of edges).

In our definition, a de Bruijn graph built upon a list of sequences  $S$  contains all distinct  $k$ -mers of  $S$  as nodes (called node-centric definition after Chikhi et al. [2021]),

see Figure 1). Importantly, node-centric means that the set of edges is implicit as can be inferred from the nodes. Indeed, nodes are connected if the  $k - 1$  suffix of a source node  $x$  matches exactly the  $k - 1$  prefix of a sink node  $y$ . Some works on de Bruijn graph use the edge-centric definition where  $k$ -mers are the labels of edges (see Figure 1 for an example).

The de Bruijn graph in bioinformatics is a subgraph of the de Bruijn graph as generally defined in computer science. Indeed, instead of showing all possible words on the given alphabet, it presents only  $k$ -mers from the original data. Therefore in the node-centric definition, it is subgraph built from a selection on the nodes of the general de Bruijn graph.

In practice, for instance in the case of  $k$ -mers extracted from reads, we do not know the original strand of the  $k$ -mer (forward or reverse). De Bruijn graphs can account for this information by being bidirected, a property that allows to encode all possible encountered overlaps (forward-forward, forward-reverse, reverse-forward, reverse-reverse) between nodes. In this manuscript, for the sake of simplicity, we avoid on purpose to present the full case which takes into account forward and reverse sequences. In practice some application keep only *canonical  $k$ -mers*, i.e. the smallest lexicographic version between a  $k$ -mer and its reverse complement (for instance, for AGGT and its reverse complement ACCT, the canonical  $k$ -mer will be ACCT).

## 2 Presentation of -tig sequences

-Tig sequences are built upon  $k$ -mers and aggregate them in different fashion. An early assembly paper (Myers et al. [2000]) introduces the first -tig of interest for us, unitigs, as “[c]ollections of fragments whose arrangement is uncontested by overlaps from other fragments”. In other words, unambiguous fragments of an assembly graph. In the following, we will see that -tig sequences are indeed very related to sequence assembly graphs, notably to the de Bruijn graph presented earlier.

### 2.1 Introduction to spectrum-preserving string sets with unitigs

#### 2.1.1 Unitigs

Unitigs are one of the most used and well known tigs. They are built from a de Bruijn graph. In assembly, unitigs are usually considered as ”safe” sequences, because it seemed that one can assemble their  $k$ -mers without ambiguity (we will see later in section 2.3.2 that this notion has been nuanced). Put another way, there is only one way to complete a sequence using  $k$ -mers from inside a unitig. When an ambiguity (a bifurcation in the graph, see examples in Figure 2) happens, the unitig is stopped and other ones start. These sequences are often output during the inner steps of an assembler, before being further elongated into contigs, which usually involve more heuristic choices in their construction (however, in other assembly schemes, contigs can also arise from different sequences than unitigs).

A path (see Box 1.a for a reminder on paths) in a de Bruijn graph is a sequence of nodes joined by edges, such that the edges in the sequence are all directed the same way, and each node appears only once. Simply put, **unitigs** are maximal simple paths in the de Bruijn graph, i.e., the longest possible sequence of nodes that can be traversed by following edges without having to choose between several sources/sinks (bifurcations).

**Definition 3 Compaction.** A compaction is an operation on  $k$ -mers whose resulting nucleotides are written in a single, larger or equal to  $k$ , string.

Let  $\mathcal{G}$  be a de Bruijn graph, and  $G$  be a compacted graph. A compaction creates of a new temporary node  $v$  from a path of nodes from the de Bruijn graph. The new node  $v$  copies the source node's nucleotides, and adds consecutively to this list all nucleotides which do not belong to an overlap between consecutive nodes of the path. When  $v$  has reached its maximal length, it is added to  $G$ 's set of nodes.

This definition of compaction is general enough to handle compactions in the next sections (2.2.1). A unitig graph is also a graph built out of compactions.

**Definition 4 Unitig graph or compacted de Bruijn graph.** Let  $G$  be a compacted graph built from a de Bruijn graph as in Definition 3. If  $G$  is a unitig graph, the maximal length for a node  $v$  is reached when a bifurcation is met in  $\mathcal{G}$ . Edges of  $G$  are edges from  $\mathcal{G}$  connecting  $k$ -mers at the extremity of unitigs.

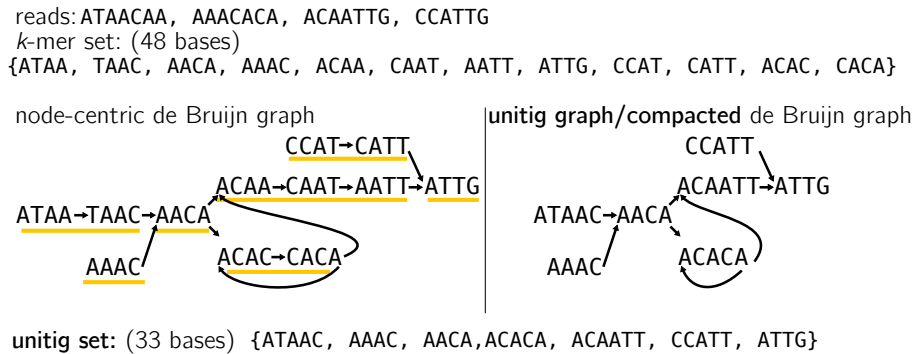


Figure 2: A read set and the extracted  $k$ -mer set (top), the corresponding de Bruijn graph ( $k=4$ , left) similar to Figure 1, paths leading to unitigs are shown in yellow in the graph, the unitig graph (right) and the unitig set (bottom). Each unitig starts or ends either at graph dead ends or when a bifurcation occurs, i.e. a node has several sinks or sources. See how the graph creates connections between reads (for instance the end of the first one becomes connected to the beginning of the third one). Note that for unitig ACACA, a second equivalent compaction is possible: CACAC, starting by  $k$ -mer CACA then compacting using the final "C" of ACAC.

The  $k$ -mers sequence of maximal simple paths of de Bruijn graphs are then *compacted* into unitigs. An example of compaction in Figure 2 starts from  $k$ -mer ATAA, and adding "C" and "A" from the two next  $k$ -mers of the unitig to obtain ATAACA (as seen in the definition, we use the term unitigs for maximal unitigs). The de Bruijn graph can be converted to a graph of unitigs and the inverse operation is possible as well. Constructing a graph of unitigs from a de Bruijn graph can have multiple solutions (unitig sets, see an example in Figure 2). Sometimes, the term compacted de Bruijn graph is encountered in place of the unitig graph (to our knowledge, the first

formalization of such a graph in bioinformatics is mentioned in Cazaux et al. [2014] and the first implementation appears in Chikhi et al. [2016]).

### 2.1.2 Efficient $k$ -mer set representation

**Definition 5 Spectrum-preserving string set (SPSS).** *Given an input string set containing  $k$ -mers  $S$ , a spectrum preserving string set of  $S$  is a plain text representation (e.g. a set of strings) that has the same set of  $k$ -mers as  $S$ .*

The SPSS notion has been introduced recently Rahman and Medvedev [2020] and has gained traction since then, as some research field have been looking into representing more efficiently sets of  $k$ -mers. Most SPSS do not handle multiplicity. Thus, they preserve the set of  $k$ -mers from a list of nucleotidic sequences, but not the  $k$ -mer multiset. There are exceptions that are presented in the following.

The most obvious SPSS is the  $k$ -mer set itself. Then we can notice that a set of unitigs from a de Bruijn graph is a SPSS as well. At worst (in a very fragmented graph), they use as much nucleotides to represent the  $k$ -mer set as the  $k$ -mer set itself, but usually, they represent it in a more compacted and efficient way (in Figure 2 we used 33 nucleotides in comparison to the 48 of the  $k$ -mer set). The representation is not optimal since nucleotides from the overlaps are represented several times. For the sake of simplicity, Figure 2 presents very small  $k$ -mers, but the burden of redundancy increases with real-life-sized  $k$ -mers (usually in the 21-51 nucleotides range for NGS reads).

In order to discuss the next -tig, notice the red substrings in Figure 2, that show some redundancy that remains in the representation. Such redundancy occurs because unitigs still share a  $k-1$  overlap on their extremities.

## 2.2 -Tig sequences for $k$ -mer set representation and management

### 2.2.1 Simplitigs and UST: nearly optimal SPSS

Keeping up with the idea of SPSS, and of representing the  $k$ -mer set, different works showed that there are better objects than unitigs to minimize the number of total number of nucleotides to be encoded. Two papers, for **simplitigs** (Brinda et al. [2021]) and for **UST** (after Unitig-STitch, Rahman and Medvedev [2020]) described a solution simultaneously, though independently. UST and simplitigs are therefore used for space footprint reduction when storing  $k$ -mer sets, and provide source code<sup>1</sup>.

Strings larger than  $k$ -mers can be found by following paths in the graph. Then, finding a SPSS is equivalent to finding a set of paths such that it covers all nodes of the de Bruijn graph (called a *path cover*), and then realize a compaction of the nodes in each path. With the constraint of each  $k$ -mer appearing only once in the final string set, this means that a  $k$ -mer should be used only once in the whole path cover (therefore called *distinct; path cover*). A simple, non minimal solution is that each path starts and ends in a single node. This would correspond to the  $k$ -mer set itself being a SPSS. The unitig set is

<sup>1</sup>UST: <https://github.com/medvedevgroup/UST/blob/master/README.md>, simplitigs: <https://github.com/prophyle/prophasm>

another distinct path cover. The UST/simplitigs intuition is that unitigs can themselves be compacted to obtain longer sequences and reduce the number of  $k-1$  redundancies.

Both papers propose a greedy algorithm to achieve that compaction. It means that for a given node, the algorithm look for an edge leading to a sink node, explores this node and uses it for compaction if it has not been explored before. The compaction is continue until there is no more node to be explored, and each node is visited only once. Interestingly these works also show that the greedy method is close to the lower bound (the lower bound being the theoretical minimum number of compacted strings). Figure 3 presents an example of these sequences compared to unitigs.

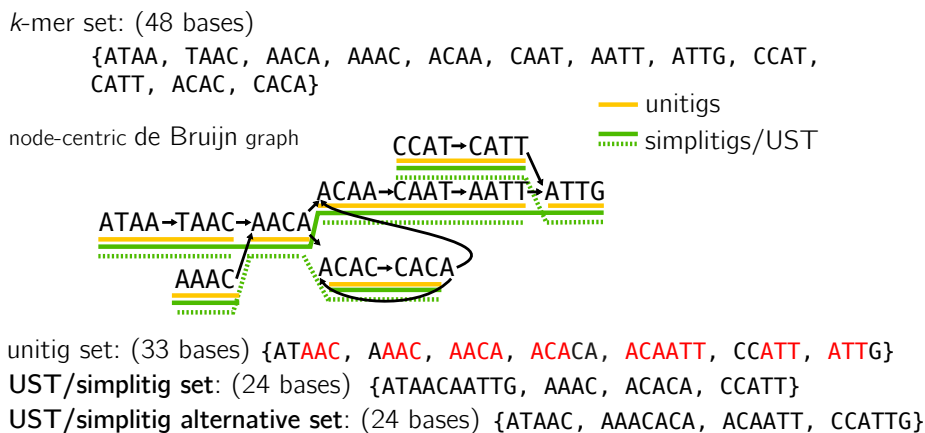


Figure 3: A  $k$ -mer set (top), and the simplitigs/UST built from the corresponding de Bruijn graph ( $k=4$ , similar to Figures 1 and 2). Two possible compaction schemes are shown (green/dotted green). Unitigs are shown for the comparison (yellow). Red parts of the unitig set correspond to redundant nucleotides that belong to  $k-1$  overlaps between unitigs. Redundant parts are reduced in simplitigs/UST.

### 2.2.2 Eulertigs and matchigs: minimal SPSS and other formulation of the problem

This section is more arduous than the previous. In a first paragraph, we intend to provide a simple intuition and let the reader decide whether they want to follow up to the technicalities.

Works following simplitigs/UST aim at minimizing the number of characters to encode the set of  $k$ -mers, with SPSS properties. To minimize the SPSS, we need the distinct path cover to be have the smallest cardinality, i.e. to include as few paths as possible. Intuitively, fewer paths lead to more  $k$ -mer compaction, hence less redundancy. Eulertigs give boundaries and a linear algorithm (see Box 2) for the minimal SPSS problem, but for now remains a more theoretical work than a practical one. UST/simplitigs are a bit faster but above all rely on a simpler graph representation. An implementation exists<sup>2</sup> and results show that all these methods compute SPSS in less than 10 minutes on the  $k$ -mers of thousands of *E. coli* genomes (Schmidt and

<sup>2</sup><https://github.com/algbio/matchtigs>

Alanko [2022]).

Here we give an informal intuition of the minimal distinct path cover problem solving. In order to find such a set, some graph traversal can guarantee each needed property, i.e. having a cover on the distinct  $k$ -mers, and yielding the least possible strings. One solution is to modify the graph by adding necessary special edges so that we can find a cycle that goes through all nodes of the graph once (see Figure 4). That would be called a hamiltonian cycle. Some constraints would be necessary, as the special edges should be added only to nodes which have not yet reached a balance between in-going and out-going edges. Then, by removing the special edges of the cycle we obtain a set of paths which is necessary minimal (see bottom right of Figure 4). Luckily, in the special case of a de Bruijn graph defined in bioinformatics on the nucleotide alphabet, this problem can be solved linearly. In the general case, finding hamiltonian cycles on a graph is NP-complete which means that it is very unlikely to find an algorithm solving this problem in a reasonable amount of time.

In practice, solving the problem means switching from the node-centric de Bruijn graph to an edge-centric de Bruijn graph (from a given node-centric dBG built from  $(k + 1)$ -mers of the data, build a dBG where all possible  $k$ -mers are in edges), and solving the same problem on edges (eulerian cycle). **Eulertigs** paper (Schmidt and Alanko [2022]) takes advantage of this property to propose an algorithm to find a minimal SPSS on a  $k$ -mer set time-linear in the size of the SPSS.

This correspondence between eulerian and hamiltonian graphs in the special case of de Bruijn graphs of  $k$ -mers has been a source of confusion on how assembly works (Medvedev and Pop [2021]). The difference here is that the goal is not to assemble genomes but indeed to find a good way to represent successive  $k$ -mers, regardless of the biological meaning of the output string.

Interestingly, two earlier works had produced all necessary material to solve this minimal SPSS problem. Crochemore et al. [2010] defined an equivalent stringology problem to the problem of finding a distinct path cover of smallest cardinality: finding *shortest common superstrings on a set of words*. Then Golovnev et al. [2013] shown how to solve this problem on de Bruijn graphs. Eulertigs are somehow their intersection although these two works were not pointed out in the paper.

A previous work (**matchtigs** in Schmidt et al. [2021]) proposed to relax the distinct path cover property, by finding a path cover which allows a  $k$ -mer to be re-used in different paths of the set (the  $k$ -mer should still appear once in a given path). The intuition is that some  $k$ -mers act as bridges that reduce the redundancy.

### 2.2.3 -Tig sequences for parallel computation: super- $k$ -mers

For performance purposes, one can need to perform operations on  $k$ -mers in a parallel way. In this case, it is interesting to have an efficient method to dispatch  $k$ -mers in balanced buckets and then to easily retrieve each  $k$ -mer's bucket. **Super- $k$ -mers of unitigs** are substring from unitigs built for this purpose. Super- $k$ -mers are built by compacting all consecutive  $k$ -mers of a unitig that share a similar *minimizer*. Super- $k$ -mers of unitigs are often less efficient than unitigs in terms of



## Box 2. Asymptotic notations.

We say an algorithm or a method solves a problem in **linear time** (or  $\mathcal{O}(n)$ ) in the size of an object  $n$  when in the worst case of running the algorithm, the time cost function of the algorithm grows more slowly than a linear function in infinity, up to a multiplicative constant.

Informally, we can say that the time needed is proportional to the size of  $n$  at worst.  $n$  is a parameter, it can be the size of the input or something else. Typically, reading a vector until a given element is found or the end of the vector is reached takes a time proportional to  $n$  the size of the vector at worst.

There are other asymptotic behaviors. Practically in bioinformatics, we aim at asymptotic behaviours close to  $\mathcal{O}(n)$  or below (e.g.  $\mathcal{O}(\log(n))$  or running in constant time) at least for large objects, for performance matters.

nucleotide minimization to represent the set of  $k$ -mers, since it is not their main goal. Historically, the first super- $k$ -mers to be introduced are the super- $k$ -mers from reads (Li et al. [2015]). They differ from the super- $k$ -mers of unitigs since they are built from the read sequences. Observe that super- $k$ -mers from unitigs are also a SPSS. In order to associate these SPSS to one and only one partition or bucket, *minimizers* are used.

**Definition 6 Minimizer.** *Given  $k$ ,  $m \leq k$  and a function  $h$  defining an order, a minimizer is the smallest  $m$ -mer with respect to the order given by  $h$ , that appears within a  $k$ -mer when screening positions  $0 \dots k - m + 1$ .*

Minimizers have been introduced in two independent contributions (Roberts et al. [2004], Schleimer et al. [2003]). We must stress that there exist other and more general definitions and use cases for minimizers than the one presented here, notably where minimizers are used to sample  $k$ -mers (for instance in sequence comparison and mapping methods). In the examples of the manuscript,  $h$  defines the lexicographical order. However, in practice,  $h$  is often a random hash function, which has been shown to have better properties for minimizers (Schleimer et al. [2003]) (therefore,  $m$ -mers are mapped to integers and the smallest integer is selected).

Thus, with a wisely chosen minimizer scheme, one can dispatch  $k$ -mers in balanced buckets per minimizer. An example of super- $k$ -mers is provided in Figure 6.

## 2.3 -Tig sequences in biological sequence analysis

De Bruijn graphs have been introduced in bioinformatics in the context of short read assembly, and therefore unitigs are tightly linked to assembly as well. **Contigs** are more loosely defined as the set of strings in the final output of an assembler. **Scaffolds**, which are contigs aggregates forming longer sequences, have been called occasionally **supercontigs** (see for instance Jaffe et al. [2003]). We will present other -tigs that were introduced more recently, such as monotigs and omnitigs.

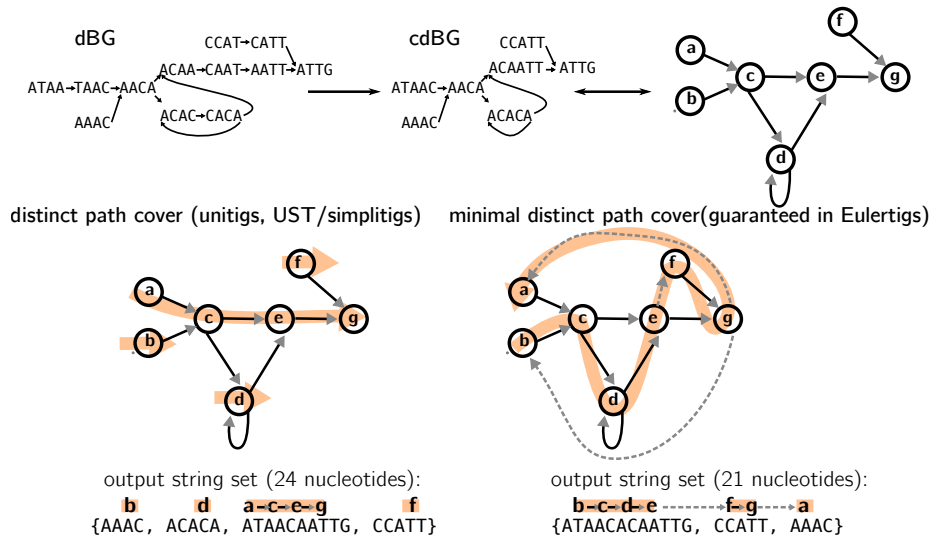
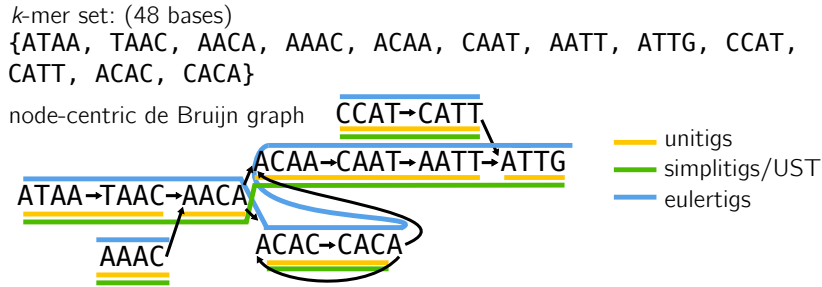


Figure 4: At the top, we show the correspondence between the graph notation we will use and the initial de Bruijn graph (similar to Figures 1,2 and 3). The de Bruijn graph is represented as an unitig graph (middle), and each unitig is given a letter to simplify the representation. At the bottom, we compare a distinct path cover that could have been computed with simplitig/UST solution for instance (same than in Figure 5, top right), with a minimal distinct path cover given by Eulertigs. Keep in mind that for the sake of simplicity we show an example on a node-centric representation, but for Eulertigs the graph has actually to be converted to an edge-centric de Bruijn graph. The Eulertig solution adds special edges (dotted lines) to balance the graph: each node has as many in-going edges than out-going edges (the paper shows it is always possible to draw all needed edges). A cycle in orange can go through any edge, special or not. Then, the cycle is broken into a set of paths by removing special edges. The result is a minimal distinct path cover.

### 2.3.1 -Tig sequences with additional information: monotigs

We now consider more information and assume that  $k$ -mers can come from different samples that are all pooled in a graph. We can consider the  $k$ -mer sample presence/absence profiles in the graph, i.e., whether a  $k$ -mer is present in a given dataset or not. The datasets can be ordered and listed, and this information can be encoded using bit vectors. For instance, a  $k$ -mer presence/absence profile can be encoded as 0110, which would mean there are 4 datasets, and this  $k$ -mer is absent from the first and fourth, and present in the second and third. **Monotigs** were introduced in Marchet et al. [2020] in order to create a SPSS that also guarantees that all  $k$ -mers in a string of the SPSS have the same bit vector profile. A side effect is that they can record multisets or sets of sets of  $k$ -mers.

Monotigs are showed in Figure 7. We notice that unitigs built from several datasets can contain  $k$ -mers that have different presence/absence profiles, for instance yielded by chimeric sequences in the assembly graph. For instance in Figure 7 the leftmost unitig ATAACA contains  $k$ -mers present in all three datasets and  $k$ -mers not present in the square dataset. This phenomenon happens as  $k$ -mers from separate samples



unitig set: (33 bases) {ATAAC, AAAC, AACA, ACACA, ACAATT, CCATT, ATTG}  
 UST/simplitig set: (24 bases) {ATAACAATTG, AAAC, ACACA, CCATT}  
 eulertigs: (21 bases) {ATAACACAATTG, CCATT, AAAC}

Figure 5: Comparison between Eulertigs (blue), simplitigs/UST (green) and unitigs (yellow) on a de Bruijn graph (graph is similar to Figures 1,2 and 3).

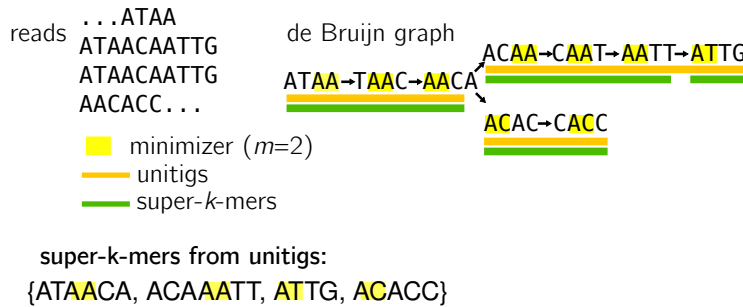


Figure 6: From a read set,  $k$ -mers are extracted and a de Bruijn graph can be built. Super- $k$ -mers from unitigs are shown at the bottom and in green paths. Minimizers ( $m=2$ ) are highlighted in yellow, we use the lexicographical order for this example. By following the green paths, we can see that super- $k$ -mers break for two reasons: end of a unitig or switch of minimizer. The yielded super- $k$ -mers of this example could be stored in three buckets labeled "AA" (ATAACA and ACAATT), "AT" (ATTG) and "AC" (ACACC).

share  $k-1$  overlaps. Monotigs fix this discrepancy by storing two different information for these  $k$ -mers.

### 2.3.2 Omnitigs and macrotigs: assembly in de Bruijn graphs

We leave the SPSS realm in this section, but we continue to review the -tig sequences. Here we make the choice to spend more time on omnitigs as they are defined on an already defined object, de Bruijn graphs. -Tigs on other assembly graphs are briefly described in section 2.3.3.

These -tigs' motivation is to represent a "safe" set of sequences, i.e., that will be found in any assembly solution from a de Bruijn graph. Indeed, assemblers add different heuristics on top on their formal definitions of assembly based on sequence graphs

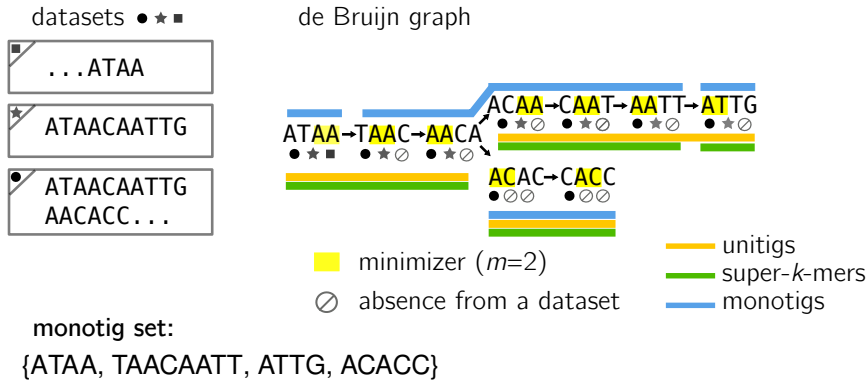


Figure 7: Three different datasets (samples, genomes, ...) are represented using a symbol (circle, star, square). A de Bruijn graph ( $k=4$ , similar to Figure 6) is built by pooling  $k$ -mers from all samples. Monotigs are built using minimizers ( $m=2$ ), and here, the presence/absence patterns are taken into account. In reality, monotig can also handle abundances. We can see that monotigs can span several unitigs, and break whether because a minimizer changes or because the profile changes.

in order for their operations to be run in a reasonable amount of time. Therefore, it is interesting to be able to extract some sequences that can show guarantees despite the possibly empiric choices implemented in assemblers. But recent work also shows that strings that were considered "safe", such as unitigs, are in fact not in all cases, since they can be substrings absent from the initial genome (Rahman and Medvedev [2022]). Previous works already demonstrated empirically that some overlaps between  $k$ -mers were spurious and had implemented correction steps guided by reads to remove them from assemblies (Bankevich et al. [2012]).

In short, when compacting unitigs in so-called contigs, the assembler has to make choices at ambiguous bifurcations. **Omnitigs** will be found in any contig set that is a solution of an assembly graph, regardless of the compaction choices.

**Omnitigs.** In a unitig graph, omnitigs (in their edge-centric definition) are a walk from node  $n_0$  to node  $n_{w-1}$  (with an edge  $e_l=(n_{l-1},n_l),0\leq l\leq w-1$ ), such that for all  $1\leq i\leq j\leq w-1$ , there is no path that allows to go from  $v_j$  to  $v_i$  without having  $e_{j+1}$  as first edge, and  $e_i$  as last edge. To illustrate this, we use two examples inspired from Tomescu and Medvedev [2017], Cairo et al. [2017] in Figure 8, one shows a walk that is an omnitig, the second shows a walk that is not an omnitig.

To describe the longest "safe" sequences from contigs, one can compute the set of maximal omnitigs.

**Introduction to omnitig construction.** To introduce omnitigs construction, we will present the Y to V operation which is a way compact the graph. This operations happens to be at the core of constructing omnitigs, although in some cases it is not sufficient, then more technicalities can be found in omnitigs papers. We will show Y to V operation's idea only.

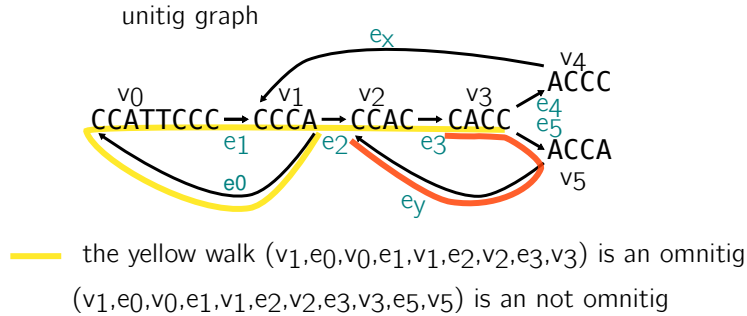


Figure 8: We work on a unitig graph built from a de Bruijn graph (see subsection 2.1). The yellow walk shows an omnitig.  $(v_1, e_0, v_0, e_1, v_1, e_2, v_2, e_3, v_3, e_5, v_5)$  is not an omnitig, indeed with this node sequence, there exist a path from  $v_3$  to  $v_2$  such that  $e_2$  is not included:  $(v_3, e_5, v_5, e_y, v_2)$  (red path).

Let's focus on bifurcations in de Bruijn graphs. In simple cases as shown in Figure 9, the two sink nodes have a single, unambiguous source. Y to V operation proposes to duplicate the content of this parent node and to compact it with the children, then remove the parent.

In Figure 9's example, the blue node is duplicated in the two children nodes. AA-

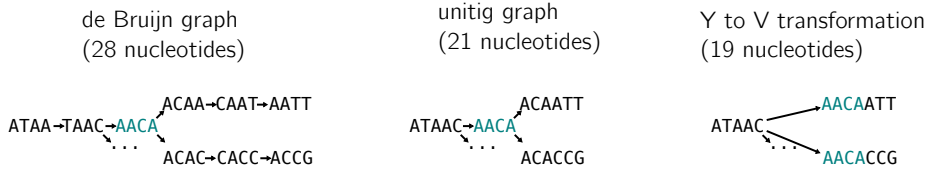


Figure 9: Y to V transformation. We start from a de Bruijn graph, converted to a unitig graph. In this branching pattern, the common source for the different sink nodes is the node AACA in blue. Note that in the unitig graph, all nucleotides of this node are redundant because of unitig overlaps. This node is duplicated and compacted to the sinks to create two distinct sink nodes in Y to V. transformation. This transformation consumes less nucleotides because the redundant overlaps AAC and ACA appear fewer times.

CAATT and AACAACG are two "safe" sequences that will be found in contigs. Y to V operation do almost all the work in building omnitigs, however an extra step is sometimes necessary. That is because Y to V is not always optimal as it can prevent finding the longest "safe" sequences in some cases. In Figure 10 we show an illustration of why Y to V operation is sometimes not enough, and can prevent from finding the maximal omnitigs in a graph.

Several papers which followed on omnitigs (Obscura Acosta et al. [2018], Díaz-Domínguez et al. [2018], Cairo et al. [2019]), and **macrotigs** were recently introduced (Cairo et al. [2020]) as a way to compute maximal omnitigs in  $\mathcal{O}(n)$  time, with  $n$  the number of edges in the graph.

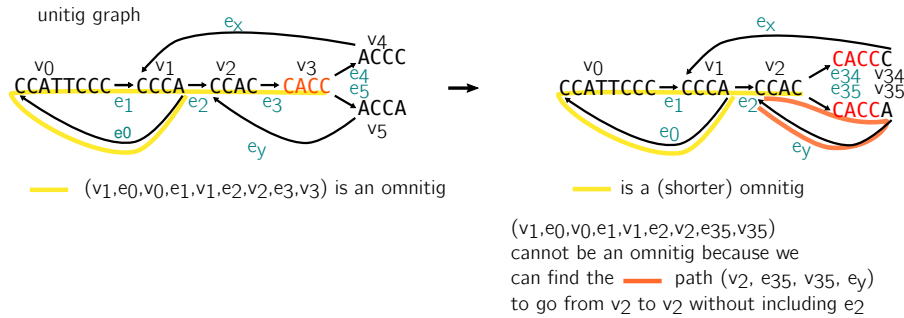


Figure 10: A unitig graph on the left undergoes a Y to V transformation.  $(v_1, e_0, v_0, e_1, v_1, e_2, v_2, e_{35}, v_{35})$  cannot be an omnitig because we can find the orange path  $(v_2, e_{35}, v_{35}, e_y)$  going from  $v_2$  to  $v_2$  without including  $e_2$ .

### 2.3.3 Long read assembly

**Disjointigs** (Kolmogorov et al. [2019]) look like simplitigs and UST. Their purpose is a bit different since they are meant to be helpful to assembly. They are constructed from a different assembly graph, the overlap graph, which is one of the data-structure used in long read assembly. Their purpose is to help solving repeats in assembly, not to reduce the storing space of a set of sequences. A different algorithm generates paths covering the graph and in particular going through left and right context of the repeats (these path can be chimeric with regard to the true genomic sequence). Then they are used to solve repeats by finding in which order these regions should be connected.

**Haplotigs** are contigs whose inner variants come from the same haplotype. Although the definition is quite ancient (Makoff and Flomen [2007]), the concept has been more frequently used with the advent of long read.

### 2.3.4 -Tig operations in pangenomics

Assembly graphs (de Bruijn graphs, overlap graphs) and pangenomics graphs (variation graphs and others) are related in the sense that they both represent the information of genomes. One major difference comes from the fact that the input of pangenomic graphs is ordered (a list of reconstructed genomes), while the data input to assembly graphs is not. Some structures make the bridge between the two, such as de Bruijn graphs built from reference genomes (Khan and Patro [2021] for the most recent). For computational needs, it can be interesting to move from one representation to the other. Principally, assembly graphs represent sequences in their nodes and overlaps in edges. Overlaps are important in this type of graphs because they materialize the support reads give to the adjacency of two sequences. On the contrary, pangenomics graph usually work from references and use edges to show direct adjacency between genomic subsequences, without overlaps. Such sequences are called **blunt sequences** (see Figure 11), with recent efforts to transition from one to the other (Eizenga et al. [2021]).

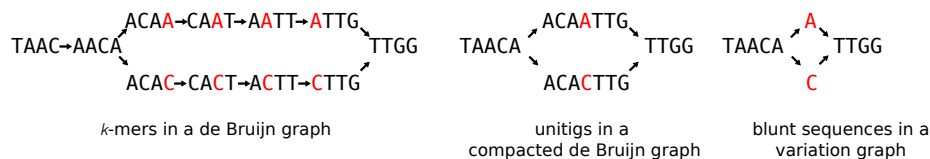


Figure 11: Difference between de Bruijn graph (left), compacted de Bruijn graph (middle) and variation graph with blunt sequences (right). An A/C mutation is shown in red.

## 2.4 Application of -tigs

### 2.4.1 Use of -tig sequences in the current literature

**Compact representation for  $k$ -mer set data-structures.** In associative data-structures, pairs of (*key,value*) can imply an explicit storage of the keys. These is the case for dictionary based on minimal perfect hash functions that emerged as highly space and time efficient data-structures for  $k$ -mers. Since these hashing techniques cannot handle duplicate input keys, they are complemented with efficient key ( $k$ -mer) set representations. SPSS, with their set properties, are interesting to that extent as they permit to record  $k$ -mers in a compact way, and to access them using quick rank/select (i.e., bit counting) operations. Therefore the UST/simpligtig concept has been adopted in recent  $k$ -mer dictionaries (Marchet et al. [2021b], Pibiri [2022a,b]). Being very recent and slower to construct, Eulertigs are not yet integrated in such implementations.

A *k-mer file format* (KFF)<sup>3</sup> marks an important milestones in  $k$ -mer management. Relying on UST and super- $k$ -mers, it has been proposed by Dufresne et al. [2022] and is currently integrated in several  $k$ -mer tools for compression,  $k$ -mer counting and pre-processing. It improves on the computation time and practicality of previous SPSS implementations. While most space efficient SPSS require around 3 bits per  $k$ -mer on a human genome, KFF takes 11-17 bits per  $k$ -mer and down to  $\sim 7$  bits when gz compressed, and is readily usable to encode them in binary.

**Compression.** General compression schemes like gzip cannot be optimal to compress  $k$ -mer sets since they have no clue about the specific nucleotide redundancy in these sets. SPSS facilitate compression by outputting objects smaller than the initial set and by working on specific features of  $k$ -mer sets. To improve compression, UST were used for  $k$ -mer sets (Rahman et al. [2021])<sup>4</sup> and another paper (Kitaya and Shibuya [2021]) introduces its own SPSS construction for compression.

**$K$ -mer partitioning.** Super- $k$ -mers are a powerful tool to allow efficient partitioning and parallel algorithms on sequence data. They are used in Marchet et al. [2021b] to accelerate the construction of a  $k$ -mer dictionary. Moreover, since all  $k$ -mers of a super- $k$ -mer can be treated as a whole, they also allow a speed-up at the query in this data-structure. The metagenomic classifier based on  $k$ -mers, Kraken (Wood and Salzberg [2014]) uses the same principal for fast  $k$ -mer retrieval. This way of partitioning  $k$ -mers becomes more and more frequent (Nyström-Persson et al. [2021]).

<sup>3</sup><https://github.com/Kmer-File-Format/>

<sup>4</sup>implementation: <https://github.com/medvedevgroup/ESSCompress>

**De Bruijn graph representation.** Noticing that more efficient  $k$ -mer set representation directly leads to better de Bruijn graph representations,  $-tigs$  have been integrated in graph compaction methods (Khan et al. [2021]). Unitigs have been used in short-read assembly context for a long time (Chikhi et al. [2016]). Globally, de Bruijn graph representation is a field in itself, with supplementary features and needs compared to plain SPSS representation (for instance, operations to navigate the graph are needed). Some methods are reviewed in Chikhi et al. [2021].

**Assembly.** De Bruijn graph and related objects have been a major asset in assembly, especially for short reads. Disjointigs are used in long read assembly from Pacific Biosciences and Oxford Nanopore technologies in several assemblers (Kolmogorov et al. [2019, 2020], Bankevich et al. [2021]). With haplotigs, we witness the progress of assembly since with some instances of long read data (for instance HiFi reads), currently being able to phase haplotypes (Cheng et al. [2021]).

**Collections of biological sequences.** Colored de Bruijn graphs can be informally defined as de Bruijn graphs built from more than one sample or genome, which  $k$ -mers are labelled with their dataset of origin. Monotigs are used in colored de Bruijn graphs (Marchet et al. [2020, 2021a]) as inner objects guaranteeing  $k$ -mer features (counts, presence/absence patterns) across different datasets. Some colored de Bruijn graphs or related structures are used as inner data-structures in pangenomics (Agret et al. [2020], Sheikhzadeh et al. [2016]).

#### 2.4.2 Possible directions

The SPSS application may seem closed because Eulertigs brought a time-linear solution to optimally solving the minimal SPSS problem, yielding a set of strings representing all  $k$ -mers once in the smallest cardinality. However, matchtigs showed that the problem can be defined differently, being changing the rules (e.g., recording a multiset of  $k$ -mers). For instance, all proposed solution work separately on each connected component (subgraphs that are not connected by any edge) of the de Bruijn graph, but it could be done differently by adding extra edges between them. Efficient indexing of the different  $-tigs$  is also vastly open.

SPSS also focused on the base-wise efficiency aspect. However, there can be other applications to SPSS, for instance there could be other ways to order the  $k$ -mers regarding assembly, so that overlaps or repeats are easier to detect. The reader might have noticed that SPSS applications were mainly directed towards short-reads, NGS data for which the volume is a bottleneck. Future direction include anticipating similar questions with long reads, and maybe include very large  $k$ -mers (more than 100 nucleotides) or more error-tolerant objects.

Finally, defining  $-tigs$  as relevant biological units would be another exploratory purpose. In RNA-seq, we need strong experimental validation to assert whether monotigs or other close objects can help studying genes by splitting them in different functional modules.



## Acknowledgments

This article followed active discussions on Twitter and a related blog post. Many thanks to Rayan Chikhi, Antoine Limasset and Paul Medvedev for their feedback and suggestions on the initial post. I also would like to thank Bastien Cazaux for his valuable input on the manuscript, and Jamshed Kahn for pointing out the first contig mention in a paper.

## References

- Clement Agret, Annie Chateau, Gaetan Droc, Gautier Sarah, Alban Mancheron, and Manuel Ruiz. Redoak: a reference-free and alignment-free structure for indexing a collection of similar genomes. *bioRxiv*, 2020.
- Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.
- Anton Bankevich, Andrey Bzikadze, Mikhail Kolmogorov, Dmitry Antipov, and Pavel A Pevzner. Lja: Assembling long and accurate reads using multiplex de bruijn graphs. *bioRxiv*, 2021.
- Karel Brinda, Michael Baym, and Gregory Kucherov. Simplitigs as an efficient and scalable representation of de bruijn graphs. *Genome biology*, 22(1):1–24, 2021.
- Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I Tomescu. Optimal omnitig listing for safe and complete contig assembly. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I Tomescu. An optimal  $o(nm)$  algorithm for enumerating all walks common to all closed edge-covering walks of a graph. *ACM Transactions on Algorithms (TALG)*, 15(4):1–17, 2019.
- Massimo Cairo, Romeo Rizzi, Alexandru I Tomescu, and Elia C Zironde. Genome assembly, from practice to theory: safe, complete and linear-time. *arXiv preprint arXiv:2002.10498*, 2020.
- Bastien Cazaux, Thierry Lecroq, and Eric Rivals. From indexing data structures to de bruijn graphs. In *Symposium on combinatorial pattern matching*, pages 89–99. Springer, 2014.
- Haoyu Cheng, Gregory T Concepcion, Xiaowen Feng, Haowen Zhang, and Heng Li. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature methods*, 18(2):170–175, 2021.
- Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

- Rayan Chikhi, Jan Holub, and Paul Medvedev. Data structures to represent a set of  $k$ -long dna sequences. *ACM Computing Surveys (CSUR)*, 54(1):1–22, 2021.
- Maxime Crochemore, Marek Cygan, Costas Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Algorithms for three versions of the shortest common superstring problem. In *Annual Symposium on Combinatorial Pattern Matching*, pages 299–309. Springer, 2010.
- Diego Díaz-Domínguez, Djamal Belazzougui, Travis Gagie, Veli Mäkinen, Gonzalo Navarro, and Simon J Puglisi. Assembling omnitigs using hidden-order de bruijn graphs. *arXiv preprint arXiv:1805.05228*, 2018.
- Yoann Dufresne, Teo Lemane, Pierre Marijon, Pierre Peterlongo, Amatur Rahman, Marek Kokot, Paul Medvedev, Sebastian Deorowicz, and Rayan Chikhi. The  $k$ -mer file format: a standardized and compact disk representation of sets of  $k$ -mers. *Bioinformatics*, 2022.
- Jordan M Eizenga, Ryan Lorig-Roach, Melissa M Meredith, and Benedict Paten. Walk-preserving transformation of overlapped sequence graphs into blunt sequence graphs with getblunted. In *Conference on Computability in Europe*, pages 169–177. Springer, 2021.
- Alexander Golovnev, Alexander S Kulikov, and Ivan Mihajlin. Approximating shortest superstring problem using de bruijn graphs. In *Annual Symposium on Combinatorial Pattern Matching*, pages 120–129. Springer, 2013.
- David B Jaffe, Jonathan Butler, Sante Gnerre, Evan Mauceli, Kerstin Lindblad-Toh, Jill P Mesirov, Michael C Zody, and Eric S Lander. Whole-genome sequence assembly for mammalian genomes: Arachne 2. *Genome research*, 13(1):91–96, 2003.
- Jamshed Khan and Rob Patro. Cuttlefish: fast, parallel and low-memory compaction of de bruijn graphs from large-scale genome collections. *Bioinformatics*, 37 (Supplement\_1):i177–i186, 2021.
- Jamshed Khan, Marek Kokot, Sebastian Deorowicz, and Rob Patro. Scalable, ultra-fast, and low-memory construction of compacted de bruijn graphs with cuttlefish 2. *bioRxiv*, 2021.
- Kazushi Kitaya and Tetsuo Shibuya. Compression of Multiple  $k$ -Mer Sets by Iterative SPSS Decomposition. In Alessandra Carbone and Mohammed El-Kebir, editors, *21st International Workshop on Algorithms in Bioinformatics (WABI 2021)*, volume 201 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-200-6. doi: 10.4230/LIPIcs.WABI.2021.12. URL <https://drops.dagstuhl.de/opus/volltexte/2021/14365>.
- Mikhail Kolmogorov, Jeffrey Yuan, Yu Lin, and Pavel A Pevzner. Assembly of long, error-prone reads using repeat graphs. *Nature biotechnology*, 37(5):540–546, 2019.
- Mikhail Kolmogorov, Derek M Bickhart, Bahar Behsaz, Alexey Gurevich, Mikhail Rayko, Sung Bong Shin, Kristen Kuhn, Jeffrey Yuan, Evgeny Polevikov, Timothy PL Smith, et al. metaflye: scalable long-read metagenome assembly using repeat graphs. *Nature Methods*, 17(11):1103–1110, 2020.

- Yang Li et al. Mspkmercounter: a fast and memory efficient approach for k-mer counting. *arXiv preprint arXiv:1505.06550*, 2015.
- Andrew J Makoff and Rachel H Flomen. Detailed analysis of 15q11-q14 sequence corrects errors and gaps in the public access sequence to fully reveal large segmental duplications at breakpoints for prader-willi, angelman, and inv dup (15) syndromes. *Genome biology*, 8(6):1–16, 2007.
- Camille Marchet, Zamin Iqbal, Daniel Gautheret, Mikaël Salson, and Rayan Chikhi. Reindeer: efficient indexing of k-mer presence and abundance in sequencing datasets. *Bioinformatics*, 36(Supplement\_1):i177–i185, 2020.
- Camille Marchet, Christina Boucher, Simon J Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 31(1):1–12, 2021a.
- Camille Marchet, Mael Kerbirou, and Antoine Limasset. BLight: efficient exact associative structure for k-mers. *Bioinformatics*, 37(18):2858–2865, 04 2021b. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab217. URL <https://doi.org/10.1093/bioinformatics/btab217>.
- Paul Medvedev and Mihai Pop. What do eulerian and hamiltonian cycles have to do with genome assembly? *PLoS Computational Biology*, 17(5):e1008928, 2021.
- Eugene W Myers, Granger G Sutton, Art L Delcher, Ian M Dew, Dan P Fasulo, Michael J Flanigan, Saul A Kravitz, Clark M Mobarry, Knut HJ Reinert, Karin A Remington, et al. A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204, 2000.
- Johan Nyström-Persson, Gabriel Keeble-Gagnère, and Niamat Zawad. Compact and evenly distributed k-mer binning for genomic sequences. *Bioinformatics*, 37(17):2563–2569, 03 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab156. URL <https://doi.org/10.1093/bioinformatics/btab156>.
- Nidia Obscura Acosta, Veli Mäkinen, and Alexandru I Tomescu. A safe and complete algorithm for metagenomic assembly. *Algorithms for Molecular Biology*, 13(1):1–12, 2018.
- Giulio Ermanno Pibiri. Sparse and skew hashing of k-mers. *bioRxiv*, 2022a.
- Giulio Ermanno Pibiri. On weighted k-mer dictionaries. *bioRxiv*, 2022b.
- Amatur Rahman and Paul Medvedev. Representation of k-mer sets using spectrum-preserving string sets. In *International Conference on Research in Computational Molecular Biology*, pages 152–168. Springer, 2020.
- Amatur Rahman and Paul Medvedev. Assembler artifacts include misassembly because of unsafe unitigs and under-assembly because of bidirected graphs. *Genome Research*, pages gr-276601, 2022.

- Amatur Rahman, Rayan Chikhi, and Paul Medvedev. Disk Compression of k-mer Sets. In Carl Kingsford and Nadia Pisanti, editors, *20th International Workshop on Algorithms in Bioinformatics (WABI 2020)*, volume 172 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISBN 978-3-95977-161-0. doi: 10.4230/LIPIcs.WABI.2020.16. URL <https://drops.dagstuhl.de/opus/volltexte/2020/12805>.
- Amatur Rahman, Rayan Chikhi, and Paul Medvedev. Disk compression of k-mer sets. *Algorithms for Molecular Biology*, 16(1):1–14, 2021.
- Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004.
- Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85, 2003.
- Sebastian Schmidt and Jarno N Alanko. Eulertigs: minimum plain text representation of k-mer sets without repetitions in linear time. *bioRxiv*, 2022.
- Sebastian Schmidt, Shahbaz Khan, Jarno Alanko, and Alexandru I Tomescu. Matchtigs: minimum plain text representation of kmer sets. *bioRxiv*, 2021.
- Siavash Sheikhezadeh, M. Eric Schranz, Mehmet Akdel, Dick de Ridder, and Sandra Smit. PanTools: representation, storage and exploration of pan-genomic data. *Bioinformatics*, 32(17):i487–i493, 08 2016. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw455. URL <https://doi.org/10.1093/bioinformatics/btw455>.
- R Staden. A mew computer method for the storage and manipulation of dna gel reading data. *Nucleic acids research*, 8(16):3673–3694, 1980.
- Alexandru I Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of computational biology*, 24(6):590–602, 2017.
- Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, 15(3):1–12, 2014.