Heuristics for the De Bruijn Graph Sequence Mapping Problem ⋆

Lucas B. Rocha[1], Said Sadique Adi[1], and Eloi Araujo[1]

Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, MS, Brasil
lucas.lb.rocha@gmail.com {said,feloi}@facom.ufms.br

Abstract An important problem in Computational Biology is to map a sequence s into a sequence graph G. One way to do this mapping is to find for a walk (or path) p in G such that p spells a sequence s' most similar to s and this problem is addressed by the Graph Sequence Mapping Problem – $GSMP$. In this article we consider the $GSMP$ using De Bruijn graph and we addressed by the De Bruijn Graph Sequence Mapping Problem – $BSMP$. Given a sequence s and a De Bruijn graph G_k, with $k \geq 2$, $BMSP$ consists of finding a walk p in G_k such taht the sequence spelled by p is the most similar to s given a distance, for example, edit distance.

Keywords: Edit distance · De Bruijn Graph · Mapping · Sequences

1 Introduction

A very relevant task in Bioinformatics is to map (transform) a sequence into another one for the purpose of comparing them. The comparison of two sequences, usually, involves comparing (or mapping) sequence with a reference sequence to find the shortest distance between them. A reference sequence represents a set of sequences and this sequence is a high-quality sequence. A reference sequence is used in the mapping task because it is a sequence of high quality. On the other hand, it is a biased sequence since it represents a limited set of sequences and, for this reason, it is not able to represent all possible variations within the sequences. One way to get around of? this situation and reduce the effect of reference sequence biases is to represent several sequences in a robust structure [1] like a sequence graph. Among the existing sequence graphs, this work considers two of them: the simple sequence graph and the De Bruijn graph.

The *sequence graph* is a graph such that each node is labeled with one or more characters and the *simple sequence graph* is a graph such that each node is labeled with a character [2]. In the *De Bruijn graph* [3, 4], of order k, each node is labeled with a distinct sequence of length k and there is an edge between two nodes if and only if? there is an overlap of length $k-1$ between the two sequences associated with them.

⋆ Supported by CAPES and UFMS.

Informally, A walk p in a graph G is an alternating list of vertices and the edges that connect them (usually described using just the vertices or just the edges). Given a sequence graph G, observe that a walk p in G induces a sequence s' by concatenating the characters associated with each node of p. Given the above information, a mapping involving a sequence and a graph of sequences gives rise to the *Graph Sequence Mapping Problem* – $GSMP$. In this problem, given a sequence s and a sequence graph G, the goal is to find a walk p in G such that the sequence s' induced by p is as similar as possible to s.

In the last decades, the $GSMP$, restricted to sequence graph and simple sequence graph, was first addressed by Manber and Wu [5] where the authors proposed an algorithm to find a pattern in a hypertext. In this work, given a pattern s and a hypertext H (represented by an acyclic directed graph), the authors proposed, for approximate mapping, a polynomial time algorithm to map s into H, allowing changes (only) in s. For exact mappings, when no changes are allowed, Akutsu proposed a polynomial time algorithm when hypertext is represented by a tree [6], and Park and Kim proposed a polynomial time algorithm when the hypertext is a directed acyclic graph (or DAG) [7].

One of the first articles that addresses $GSMP$ in more detail was written by Amir et al. in the article entitled *Pattern Matching in Hypertext* [8]. The results of this article were improved by Navarro and these improvements were detailed in the article entitled *Improved Approximate Pattern Matching on Hypertext* [9]. For the approximate mapping, Amir *et al.* were the first authors in the literature who identified an asymmetry in the location of the changes, showing the importance of understanding whether changes happen only in the pattern, only in the hypertext or in both. Considering the asymmetry identified by Amir *et al.*, the $GSMP$ allow three variants:

1. allows changes only in the pattern when mapping the pattern in hypertext;
2. allows changes only in the hypertext when mapping the pattern in hypertext;
3. allows changes in the pattern and hypertext when mapping the pattern in hypertext.

For variant 1, Amir *et al.* proposed an algorithm that runs in time $O(|V| + m \cdot |A|)$ which was improved by Navarro to run in time $O(m(|V| + |A|))$, where $|V|$ is the number of nodes in the graph, $|A|$ is the number of edges, and m is the length of the mapped pattern. For variants 2 and 3, Amir *et al.* proved that they are NP-complete problems considering Hamming and edit distance.

More recently, the $GSMP$ was addressed in the article entitled *Aligning sequences to general graphs in* $O(|V| + m \cdot |A|)$ *time* [10]. In this work, Rautiainen and Marschall proposed an algorithm for the variant 1 that runs in time $O(|V| + m \cdot |A|)$. In the work entitled *On the Complexity of Sequence-to-Graph Alignment* [11], Jain *et. al* proposed a correction of the algorithm proposed by Rautiainen and Marschall.

Previous works consider a walk (a path that allows repeating nodes) in the graph when performing the sequence mapping, but the $GSMP$ was studied for De Bruijn graphs and this gives rise to the *De Bruijn Graph Sequence Mapping*

Problem – $BSMP$. For the $BSMP$, the problem was studied considering paths (a path that does not allow repeating nodes). One of the first articles dealing with this topic with De Bruijn's graph was written by Limasset *et. al* in the article entitled *Read Mapping on De Bruijn Graphs* [12]. In this article, it was proved that the problem studied is NP-complete considering the Hamming distance when searching for a path. The authors developed a heuristic for the problem called BGREAT, based on the *seed-and-extend* technique. Note that for this version it doesn't make sense to talk about variants because there are no nodes repetitions.

The main focus of this work is to propose and investigate the use of heuristics for the $BSMP$ for the variant 1 (*changes in sequence*). In this work we consider two versions of $GSMP$:

1. $GSMP$: algorithm that returns the sequence mapped in the graph in time $O(m|V|\log(m \cdot |V|) + m \cdot |E|)$;
2. $GSMP_{ac}$: algorithm that returns *only* the cost of the mapping in time $O(|V| + m \cdot |E|)$.

In this work we adapt $GSMP$ and $GSMP_{ac}$ algorithms to run with De Bruijn graphs and these two algorithms are easily adapted. To perform this adaptation it is necessary to convert a De Bruijn graph into a simple sequence graph, basically, we need to transform each node into k nodes and make the necessary connections and run the $GSMP$ or $GSMP_{ac}$ algorithms. This adaptation was performed here and we called De Bruijn sequence Mapping Tool – BMT for version 1 and BMT_{ac} for version 2, but the algorithms take a long time to execute and so we propose a heuristic that applies to BMT and BMT_{ac}, and we call here BMT_{h1} and $BMT_{h1_{ac}}$, respectively. For both heuristics we developed a version called BMT_{h1_b} and $BMT_{h1_{acb}}$ that builds De Bruijn graphs with the Bifrost tool [13]. For the BMT idea, we developed two more heuristics considering the *seed-and-extend* technique and we call them BMT_{h2} and BMT_{h3}.

This work is organized as follows: in Section 2, we describe the basic concepts we use throughout this work; in Section 3 we present the main contributions of this paper which are three heuristics for the BMT and one heuristic for the BMT_{ac}; in Section 4 we perform experiments; and finally, in Section 5 we discuss the results and perspectives.

2 Preliminaries

In this section we describe some basic concepts that are used in this paper.

2.1 Sequences and algorithms

Let Σ be an **alphabet** with a finite number of characters. We denote a sequence (or string) s over Σ by $s[1]s[2]\ldots s[n]$ where each character $s[i] \in \Sigma$. We say that the **length** of s, denoted by $|s|$, is n and that s is a n**-length** sequence.

We denote the 0-length or **empty** sequence by λ. We say that the sequence $s[i]s[i+1]\ldots s[j]$ is a **substring** of s and we denote it by $s[i,j]$. A substring of length k of a sequence s is called the k**-mer** of s. For $j \leq n = |s|$, a substring $s[1,j]$ is called a **prefix** of s and a substring $s[j,n]$ is called a **suffix** of s. Given two sequences s and t, such that $s = uw$ (u is a prefix and w is a suffix of s) and $t = wv$ (w is a prefix and v is a suffix of t), the substring w is an **overlap** of s and t. For the two distinct sequences s and t, we define from st the **concatenation** of s and t.

The **Hamming distance** d_h of two n-length sequences s and t is defined as

$$d(s,t) = \sum_{i=1}^{n} [s[i] \neq t[i]],$$

where $[s[i] \neq t[i]]$ is equal to 1 if $s[i] \neq t[i]$, and 0 otherwise. We also say that s and t have $d_h(s,t)$ differences.

The **edit distance** is the minimum number of edit operations (insertion, deletion and substitution to transform one sequence into the other. Formally, the edit distance $d_E(s,\ t)$ between s and t, such that $n = |s|$ and $m = |t|$, is defined as

$$d_e(s,t) = \begin{cases} n+m & \text{if } n = 0 \text{ or } m = 0, \\ \min \left\{ \begin{array}{l} d_e(s[1,n], t[1,m-1]) + \Delta_{ins} \\ d_e(s[1,n-1], t[1,m]) + \Delta_{del} \\ d_e(s[1,n-1], t[1,m-1]) + \Delta_{sub} \end{array} \right\} & \text{otherwise} \end{cases}$$

and $\Delta_{sub} = \Delta_{n,m}$ which is the cost of replacing the character $s[n]$ with $t[m]$, and vice versa. At edit distance, the penalty for inserting and deleting a character is $\Delta_{del} = \Delta_{ins} = 1$ and the cost for replacing a character is $\Delta_{sub} = 1$.

A **graph** is an ordered pair (V, A) of two sets where V is a set of elements called **nodes** (of the graph) and A is a set of elements called **edges** (from the graph). Each edge corresponds to a pair ordered of nodes.

Given a graph G, a **walk** in G is a sequence of nodes $v_1, \ldots, v_k$, where each $(v_i, v_{i+1}) \in A$. A **path** in G is a walk that does not repeat nodes. Given a graph G with costs on the edges, the **cost** of a path p with l nodes in G is determined by the cost function $c(p) = \sum_{j=1}^{l-1} c(v_j, v_{j+1})$ where $c : A \times A \to \mathbb{R}$.

For graphs without costs in edges, the **shortest path** between two distinct nodes s and t in a graph G is determined by the path that contains the least amount of edges from s to t. For graphs with costs associated with edges, the shortest path is determined by the least cost path from s to t.

Graphs whose nodes are labeled with a sequence of characters in Σ^* are called **sequence graph**. In this paper, for ease of reading, we denoted that the nodes are sequences in sequence graphs. We denoted a **simple sequence graph** a graph where each node has a character.

Given a set of sequences $S = \{r_1, \ldots, r_m\} \subset \Sigma^k$ and an integer $k \geq 2$, a **De Bruijn graph** of order k, $G_k = (V, A, x : V \to \Sigma^k)$, built on S corresponds to a graph where each node is a k**-mer** distinct from the sequences of S. Formally,

given a sequence set S and an integer $k \geq 2$, we define a De Bruijn graph as a graph $G_k = (V, A)$ where:

- $V = \{d \in \Sigma^k |$ such that d is a substring of length k of $r \in S\}$; and the edges are induced with
- $A = \{(d, d')|$ the length suffix $k-1$ of d is a prefix of $d'\}$.

Notice that there is no need the set edge does not exist explicitly because the edges are induced by pairs of nodes (k-mers). Thus we can represent a De Bruijn graph only by its set of nodes V.

Given a walk (or path) $p = v_1, v_2, \ldots, v_l$ in a graph of De Bruijn G, the **sequence induced** by p is formed by concatenating the characters associated with each node of p, with the overlap between the k-mers appearing a single time in this concatenation, that is, $x(p) = v_1 v_2[k] \ldots v_n[k]$, it is the concatenation of v_1 k-mer with the last character of each k-mer $v_2, \ldots, v_l$.

Given the definitions above, the following problem can be formulated:

*Problem 1 (**De Bruijn Graph Sequence Mapping Problem** – $BSMP$).* Given as input a sequence s, of length m, and a De Bruijn graph of order k, G_k, find a walk (or path) p in G_k such that the distance between induced (by p) sequence s' and sequence s is as small as possible.

Below we will show how the algorithm for to map a sequence into general graphs ($GSMP$) works and how to adapt it to use a De Bruijn graph. Given as input a sequence s and a graph of sequences G, the mapping of s into G is performed using an auxiliary graph, the multilayer graph.

A **multilayer graph** G' contains $|q|$ layers where each layer contains a copy of the graph G and a node *dummy*. In addition to the nodes of all layers, we have a source node s connected to all nodes of layer 1 and each node of layer $|q|$ connecting to a target node t (see Figure 1). Therefore, the multilayer graph is defined as follows: given a sequence q, a sequence graph $G = (V, A, x : V \to \Sigma)$ and parameters Δ_{del}, Δ_{ins}, and a replacement cost Δ_{sub}, the multilayer graph G' is defined with the set of nodes $V' = (\{1, \ldots, |q|\} \times (V \cup \{\delta\})) \cup \{\mathrm{s}, \mathrm{t}\}$ and the cost function on the edges

$$\sigma((v_k, j), (v_i, j')) = \begin{cases} \Delta_{sub} & \text{if } (v_k, v_i) \in A \text{ e } j+1 = j', \\ \Delta_{del} & \text{if } v_k = v_i \text{ and } j = j', \\ \Delta_{ins} & \text{if } (v_k, v_i) \in A \text{ e } j = j', \\ \infty & \text{otherwise} \end{cases}$$

where $\Delta_{sub} = \Delta_{i,j'}$ and there is an edge in the set of edges A' of G' if its weight is finite. Additionally, we have the nodes s, t and *dummy* and in the edge set A' we also have edges between s and each $(v_i, 1)$ with cost $\Delta_{i,1}$ for $i > 0$ and cost Δ_{del}, for $i = 0$, we have edges between each $(v_i, |q|)$ and t with cost 0 and for each node *dummy* δ, (v_0, j), there is an edge for $v_{v_i, j+1}$ with cost $\Delta_{i,j+1}$ for $i > 0$ and cost Δ_{del}, for $i = 0$.

Based on the multilayer graph G', the solution to map q to G consists of finding a minimum path (minimum cost) in G' from s to t. An example of the mapping is shown in Figure 1, for this same example, the solution is the path from s to t in G'. This path is $p = s, (v_1, 1), (v_2, 2), (v_3, 2), (v_4, 3), t$ at a cost of 1. Considering the example sequence $q =$`ACT`, the path p represents replacing with cost 0, q_1 with $\sigma(v_1)$ and q_2 with $\sigma(v_2)$, insert $\sigma(v_3)$ into q with penalty 1 and replace q_3 with $\sigma(v_4)$ with cost 0.

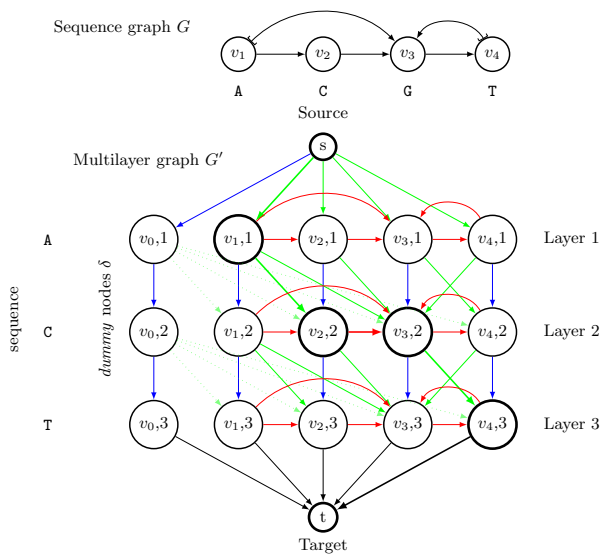

Figure 1. Example of mapping the sequence $q =$`ACT` in the sequence graph G. The red, blue, and green edges are the insertion, deletion, and substitution costs, respectively. [[10, 11]].

To use this proposed idea of $GSMP$ with De Bruijn graphs, we need to convert a De Bruijn graph into a simple sequence graph and which we will simply call a sequence graph. In summary, BMT converts a De Bruijn graph to a sequence graph and runs $GSMP$'s algorithm. An example of conversion is shown in the Figure 2. Considering the $GSMP_{ac}$, the authors describe its functioning in their articles [10, 11] and in this work we will use it, but we will omit the description of its functioning. We use the De Bruijn graph to sequence graph conversion above for $GSMP_{ac}$ as well.

3 Heuristic approaches

The $PSMB$ algorithm determines the mapping of a sequence s into a De Bruijn graph G_k and the main problem with this version is the amount of memory

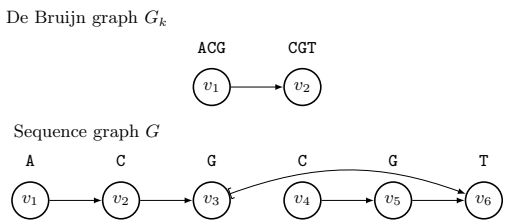

Figure 2. Example of converting a De bruijn graph G_k to a sequence graph G.

consumed by the algorithm. In summary, considering that the De Bruijn graph G_k has N nodes, the memory consumed is $O(N \cdot k)$. Now, the sequence s has $|s|$ characters, and the multilayer graph has $|s| \cdot (N \cdot k) + |s| + 2$ nodes, all memory consumed is

$$
\begin{aligned}
&= O((N \cdot k) + |s| + (|s| \cdot (N \cdot k) + |s| + 2)) \\
&= O((N \cdot k) \cdot (|s| + 1) + 2 \cdot |s| + 2) \\
&\leq O((N \cdot k) \cdot (|s| + 1) + 2(|s| + 1)) \\
&= O((N \cdot k + 2)(|s| + 1))
\end{aligned}
$$

As the BMT consumes a lot of memory to execute, the main idea is to propose a heuristic that allows the execution of the BMT and that our approaches do not lose precision in the answers, for that we use the idea of anchors (k-mers that are present in s and G_k) and then fill with BMT all gaps between two sequential anchors.

Heuristic 1 In detail, BMT_{h1} follows these steps:

- **Step 1**: find the anchors in G_k; and with BMT
- **Step 2**: find the *begin* walk: the walk that ends in the first anchor.
- **Step 3**: find walks between the inner anchors.
- **Step 4**: find the *end* walk: the walk that starts in the last anchor.

A generic example is shown in Figure 3. The squares are anchors, the first and last wavy lines are the begin and end walks, the other walks are internal.

Note that it is disadvantageous and very expensive to consider the entire De Bruijn graph to determine the walks in G_k with the BMT. For this reason, BMT_{h1} considers only one set of nodes in the search: considering that the gap has size N, only nodes that are $0.5 \cdot N$ distant from the source node and target node are considered. With these features, the heuristic is able to run larger tests, but the result may not be able to map the entire s sequence and for our work we consider as an answer the mapping that covers most of the sequence.

For BMT_{ac}, it only returns the cost, and this algorithm is capable of processing huge graphs, but it takes a lot of time. Our main focus is to apply the

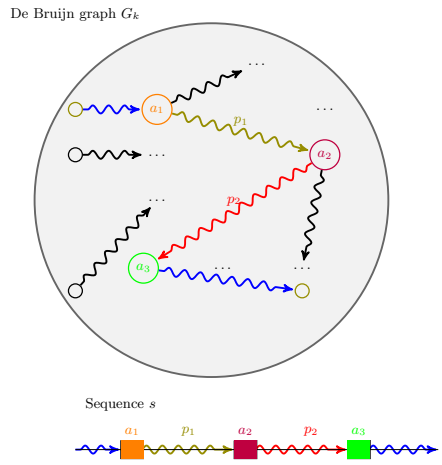

Figure 3. Representation of the mapping of the sequence s in G_k. Each square is an anchor found in G_k and the wavy lines between the anchors are the best paths found between them with BMT.

same idea of anchors where we will use BMT_{ac} to look for the costs of each gap and $BMT_{h1_{ac}}$ is our proposed heuristic for BMT_{ac}

In order to improve the performance of Heuristic 1, we have developed a version called BMT_{h1_b} and $BMT_{h1_{acb}}$ that builds De Bruijn graphs with the Bifrost tool [13].

Another way to perform the mapping of a sequence in a De Bruijn graph is using the seed-and-extend technique which consists of: for each k-mer of the sequence, extend through the graph comparing the sequence with the k-mer of the graph. Given a sequence s and an omitted De Bruijn graph G_k, a seed-and-extend idea is presented in Figure 4. Given `TACAC` and `GCTGC` as anchors in the graph and for each anchor we were able to extend by a walk comparing with the sequence s, the last line presents, from the first anchor, a mapping in the graph with only one difference.

```
s   TACGTACACACTCGTAGCTAGCTGCATCTATCTATA
        TACAC           GCTGC
        TACACACACGTAGCTAGCTGC
        TACAC           GCTGCATCTATCTATA
        TACACACTCGTAGCTAGCTGCATCTATCTATA
```

Figure 4. Example of mapping the sequence s into a De Bruijn graph G_k using seed-and-extend.

Heuristic 2 In detail, BMT_{h2} follows this idea: extend each anchor and take the longest walk that covers the sequence. From the first anchor, extend considering all possibilities of k-mers changing the last base and taking the longest walk. If the longest walk reaches the next anchor, continue extending, otherwise, take the next anchor and extend from there. We can have multiple walks covering the sequence, consider the longest walk.

Heuristic 3 With the idea of improving BMT_{h2} coverage, here we consider removing bases from the mapped sequence. In details, BMT_{h3} follows this idea: from the first anchor, extend considering all possibilities of k-mers changing the last base and taking the longest walk. If the longest walk reaches the next anchor, continue extending, otherwise, remove the length suffix $k-1$ from the initial anchor and extend again.

Note that in heuristic 2 we only perform substitution operations and in heuristic 3 we only perform substitution and deletion operations in the sequence. All implementations are available on github: github.com/LucasBarbosaRocha.

4 Experiments and Results

The experiments were performed on on a computer with processor Intel(R) Xeon(R) CPU E5-2620 2.00GHz of 24 CPUs and 62 GB of RAM,. The tests in Section 4.1 include BMT_{ac} with $BMT_{h1_{ac}}$ and $BMT_{h1_{acb}}$. In Section 4.2 include BMT with BMT_{h1} and BMT_{h1_b}, as BMT is not capable of performing large tests, we use BMT_{ac} to obtain the mapping cost to compare with our heuristics. In Section 4.3 include tests with BMT_{h1}, BMT_{h1_b}, BMT_{h2} and BMT_{h3}; and BMT_{ac} to obtain the mapping cost.

In our tests we used sequences from NCBI to build the graph and a distinct sequence (in relation to the sequences in graph) for the mapping. The sequences are from *Escherichia coli*, we used sequences of length on average 100 (short read) to build the graph and sequences of length over 1000 (long read) to map in the graph.

We have time and accuracy comparisons. In Section 4.1, we compare the average costs, accuracy and runtime of implementations BMT_{ac}, $BMT_{h1_{ac}}$ and $BMT_{h1_{acb}}$. In Section 4.2, we compared the average costs, accuracy and runtime of implementations of BMT_{h1} and BMT_{h1_b} and we used the BMT_{ac} to get the mapping cost because the BMT is not capable of processing large numbers of k-mers and we compared the times of the BMT_{h1} and BMT_{h1_b}. Finally, in Section 4.3, we compare the the average costs, accuracy and runtime of implementations between all heuristics and we also use BMT_{ac} to get the cost and we compare the time among the heuristics.

To verify the average of costs, we add up all the costs obtained by the heuristics BMT_{ac}, $BMT_{h1_{ac}}$ and $BMT_{h1_{acb}}$ and divide by the number of tests performed. For the BMT_{h1}, BMT_{h1_b}, BMT_{h2} and BMT_{h3} heuristics, we take the result sequences of the heuristics, calculate the edit distance between the mapped sequence (which is the cost), add and divide by the number of tests. To verify

the accuracy of the results, we checked for each row on costs table the difference between the output of the algorithm BMT_{ac} and that one of each heuristic.

4.1 Tests to obtain only costs

To run the tests, we take sequences of varying lengths and use De Bruijn graphs that a varying amount of k-mer. The average running times obtained for tests are shown on Table 2, and the average costs are shown on Table 1, and the charts for each table are shown on Figure 6and 5, respectively. We can see that for the version that only looks for the cost of the mapping, it is possible to have a great time gain because we take advantage of anchors to not process the entire sequence. The anchors in these tests increase as the number of k-mers in the graph increases and the number of anchors here is between 23 and 6000 anchors.

A relevant point of using Bifrost in $BMT_{h1_{acb}}$ is that it has its own way of efficiently creating the De Bruijn graph and its own interpretation of k-mers, in general, the amount of k-mers between our tool ($BMT_{h1_{ac}}$) and the use of Bifrost ($BMT_{h1_{acb}}$) can vary and this has an impact on the final answer. We can observe this impact in tests when looking at the difference between the responses of $BMT_{h1_{ac}}$ and $BMT_{h1_{acb}}$, so $BMT_{h1_{acb}}$ has advantages in terms of time and cost compared to BMT_{ac} and $BMT_{h1_{ac}}$.

Table 1. Average costs: tests with long reads and their length between 2381 and 6225. The "Qty. k-mers" column is the number of k-mers in the graph and the $k = 10$.

	Average costs			Accuracy	
Qty. k-mers	BMT_{ac}	$BMT_{h1_{ac}}$	$BMT_{h1_{acb}}$	$BMT_{h1_{ac}}$	$BMT_{h1_{acb}}$
4577	1537,50	3147,00	3181,00	2,05	2,07
9119	1453,25	3184,25	3098,25	2,19	2,13
11375	1427,00	3142,25	3028,00	2,20	2,12
22485	1201,75	3111,25	2835,00	2,59	2,36
43842	995,25	2850,50	2558,00	2,86	2,57
84683	662,75	1663,25	1550,00	2,51	2,34
557522	314,50	552,25	477,25	1,76	1,52
760644	261,00	477,75	483,00	1,83	1,85

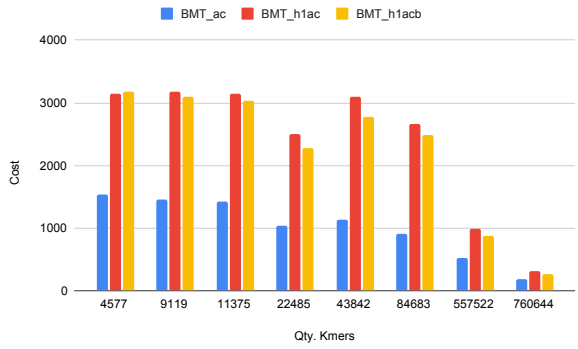

Figure 5. Average costs: chart for Table 1

Table 2. Average time (seconds): tests with long reads and their length between 2381 and 6225. The "Qty. k-mers" column is the number of k-mers in the graph and the $k = 10$.

Qty. k-mers	BMT_{ac}	$BMT_{h1_{ac}}$	$BMT_{h1_{acb}}$
4577	73,75	5,00	3,50
9119	51,50	7,75	5,75
11375	60,00	32,50	24,75
22485	118,00	24,00	13,25
43842	393,75	261,75	177,75
84683	1708,00	66,00	41,75
557522	2898,25	45,00	29,75
760644	1185,00	16,25	12,75

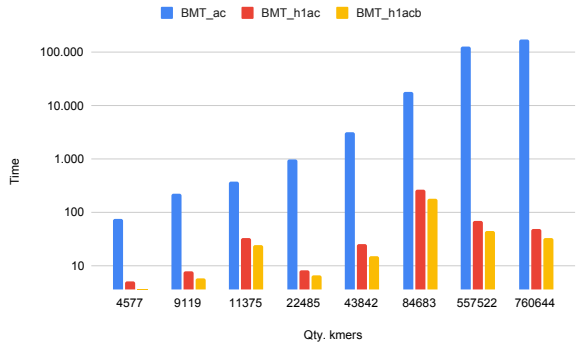

Figure 6. Average time (seconds): chart for Table 2

We performed another tests varying the number of sequences and varying the value of k. The average running times obtained for tests are shown on Table 4, and the average costs are shown on Table 3, and the charts for each table are shown on Figure 8 and 7, respectively. In these new tests we get several sequences with length between 1000 and 10000; and the value of k varying between 5, 10 and 20.

In these tests we can see how the use of the Bifrost tool is advantageous and that our tool, $BMT_{h1_{ac}}$, manages to be competitive in this scenario of seeking only the cost. In general, we can obviously conclude that these tools are related to the amount of anchors between the mapped sequence and the graph. These new tests present a high amount of anchors and the results were better. In general, the more anchors, the more similar the sequence is to the graph and this helps our tools.

Here the amount of anchors is between 0 and 9234. For the $k = 5$ we have the largest amount of anchors and this helps both in time and response. For $k = 10$ we have the amount of anchors being around 50% of the length of the sequence and this gave us a reasonable result. For $k = 20$ we have a special case that consists of having almost no anchors, for example in many cases we have 0 anchors, but the result is very close to the BMT_{ac} and this happens because the BMT_{ac} with few anchors has the cost of editing the whole sequence to map in the graph and the heuristics when not being able to map anything in the sequence return the cost of editing the whole sequence.

Table 3. Average costs: tests with long reads and their length between 1000 and 10000. The "Qty. k-mers" column is the number of k-mers in the graph.

		Average Costs			Accuracy	
Qty. k-mers	k	BMT_{ac}	$BMT_{h1_{ac}}$	$BMT_{h1_{acb}}$	$BMT_{h1_{ac}}$	$BMT_{h1_{acb}}$
1017	5	8,25	11,46	9,99	1,39	1,21
11375	10	2363,48	5334,36	5173,17	2,26	2,19
10243	20	5278,52	5383,83	5743,46	1,02	1,09

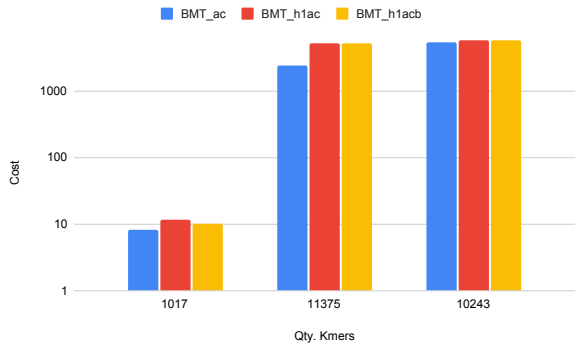

Figure 7. Average costs: chart for Table 3

Table 4. Average time (seconds): tests with long reads and their length between 1000 and 1000. The "Qty. k-mers" column is the number of k-mers in the graph.

Qty. k-mers	k	BMT_{ac}	$BMT_{h1_{ac}}$	$BMT_{h1_{acb}}$
1017	5	8,25	3,72	1,72
11375	10	2363,48	42,32	21,16
10243	20	5278,52	6,29	3,02

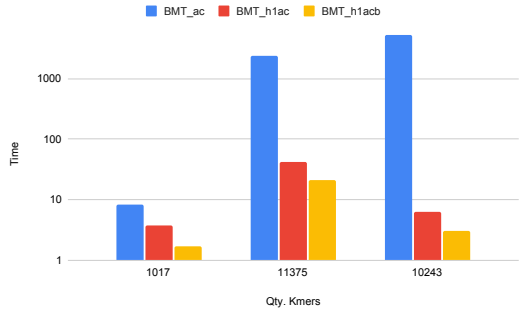

Figure 8. Average time (seconds): chart for Table 4

4.2 Tests to obtain mapping - heuristic 1

To run the tests, we take sequences of varying lenghts and we use the De Bruijn graph with varying number of k-mers. The heuristics return the spelled string

and we calculate the edit distance between the original string and the result and compare it to the mapping cost returned by the BMT_{ac}. In these tests, the number of anchors is between 0 and 2015 and for this reason we can observe, in terms of accuracy, that the heuristics became more distant in reaction to the cost obtained by the BMT_{ac}. The average running times and costs obtained for tests are shown on Table 5. The graphs for average running time and costs are shown on Figure 10 and Figure 9, respectively.

The version that returns the mapping was not able to process the vast majority of tests and for that reason the main advantage we have here is the mapping being performed with the heuristics, thus providing the possibility of performing the mapping.

Table 5. Average costs and accuracy: tests with long reads and theirlength between 2381 and 6225. The "Qty. k-mers" column is the number of k-mers in the graph.

		Average costs			Accuracy	
k	Qty. k-mers	BMT_{ac}	BMT_{h1_b}	BMT_{h1}	BMT_{h1_b}	BMT_{h1}
10	78661	1746,38	3046,88	5623,00	1,74	3,22
15	86214	2416,25	5197,50	6255,88	2,15	2,59

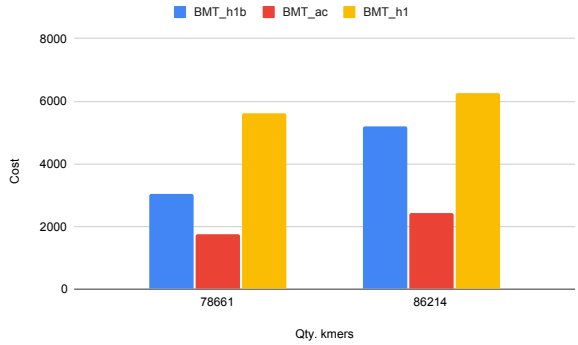

Figure 9. Average costs: chart for Table 5

Table 6. Average time (seconds): tests with long reads and their length between 2381 and 6225. The "Qty. k-mers" column is the number of k-mers in the graph.

k	Qty. k-mers	BMT_{h1}	BMT_{h1_b}
10	78661	2250,00	1125,00
15	86214	3160,25	1365,00

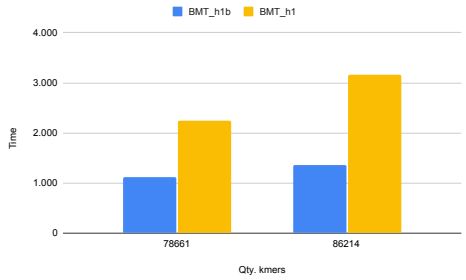

Figure 10. Average time (seconds): chart for Table 6

4.3 Tests to obtain mapping - all heuristics

To run the tests, we take sequences of varying lenghts and use De Bruijn graphs that a varying amount of k-mer. Here we follow the same idea as in the previous section to calculate the costs and compare all the heuristics. The average running times and costs obtained for tests are shown on Table 5. The graphs for average running time and costs are shown on Figure 10 and Figure 9, respectively.

We can see that BMT_{h1_b} is the best heuristic in time and quality and is our most powerful version because its graph construction tool (Bifrost) is capable of creating much larger graphs. BMT_{h3} is a heuristic that, despite being simple, manages to be the second best and deliver results in an accessible time. Another important detail is that the number of anchors is relevant for the good execution of the heuristics BMT_{h1} and BMT_{h1_b} and the greater the number of anchors, the better the answer will be and therefore we will process less of the sequence. For heuristics BMT_{h2} and BMT_{h3}, anchors are not used, but it is important to note that a sequence with many anchors is a sequence very similar to the graph and this indicates that the h2 and h3 heuristics will obtain better answers.

Table 7. Average costs: tests with long reads and their length between 2381 and 6225. The "Qty. k-mers" column is the number of k-mers in the graph.

			Average costs			
k	Qtd. k-mers	BMT_{ac}	BMT_{h1_b}	BMT_{h1}	BMT_{h2}	BMT_{h3}
10	78661	1746,38	3046,88	5623,00	5435,63	4722,75
15	86214	2416,25	5197,50	6255,88	6216,25	5022,38
			Accuracy			
			1,74	3,22	3,11	2,70
			2,15	2,59	2,57	2,08

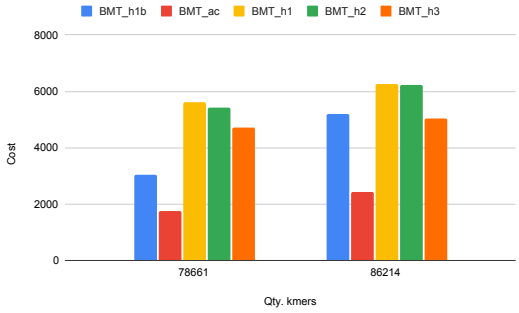

Figure 11. Average costs: chart for Table 7

Table 8. Average time (seconds): tests with long reads and their length between 2381 and 6225. The "Qty. k-mers" column is the number of k-mers in the graph.

k	Qtd. k-mers	BMT_{h1_b}	BMT_{h1}	BMT_{h2}	BMT_{h3}
10	78661	1125,00	2250,00	1508,38	2160,63
15	86214	1365,00	3160,25	1882,50	4560,00

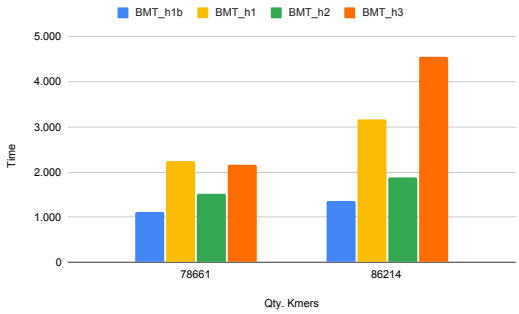

Figure 12. Average time (seconds): chart for Table 8

We performed another tests varying the number of sequences and varying the value of k. The average running times obtained for tests are shown on Table 10, and the average costs are shown on Table 9, and the charts for each table are shown on Figure 14 and 13, respectively. In these new tests we get several sequences with length between 1000 and 10000; and the value of k varying between 5, 10 and 20. In these new tests we can verify a more interesting

competitiveness in terms of time and quality. The heuristics were very close in values and heuristics BMT_{h2} and BMT_{h3} showed their efficiency in achieving results close to the other heuristics. These two heuristics, BMT_{h2} and BMT_{h3}, despite not using the idea of anchors, we can see that if the sequence has many anchors it means that it is more similar to the graph and it is relevant for these last two heuristics because they process less of the sequence.

Here the number of anchors is between 0 and 9234. For the small k 5 we have the largest number of anchors and this gives us a very mapped sequence. For k 10 we have the amount of anchors being around 50% of the sequence length and this gives us a partially mapped sequence. For k 20 we have a special case that consists of having almost no anchors, for example in many cases we have 0 anchors, but the result is very close to the BMT_{ac} and this happens because our heuristics return a walk p that spells a small sequence and when calculating the accuracy with the original sequence we have a high edit distance and this is close to the cost returned by BMT_{ac}.

Table 9. Average costs: tests with long reads and their length between 1000 and 1000. The "Qty. k-mers" column is the number of k-mers in the graph.

			Average costs			
k	Qty. k-mers	BMT_{ac}	BMT_{h1_b}	BMT_{h1}	BMT_{h2}	BMT_{h3}
5	1017	8,25	9,99	11,46	54,76	26,86
10	11375	2363,48	5173,17	5334,36	5841,82	5684,60
20	10243	5278,52	5743,46	5383,83	6158,55	5850,59
			Accuracy			
			1,21	1,39	6,64	3,26
			2,19	2,26	2,47	2,41
			1,09	1,02	1,17	1,11

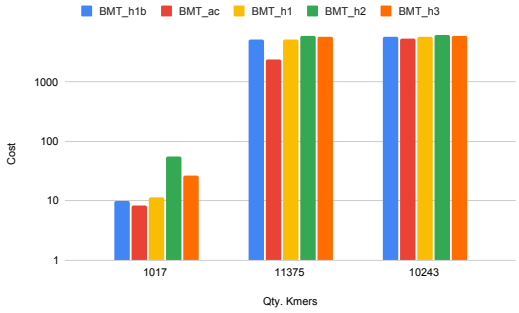

Figure 13. Average costs: chart for Table 9

Table 10. Average time (seconds): tests with long reads and their length between 1000 and 1000. The "Qty. k-mers" column is the number of k-mers in the graph.

k	Qty. k-mers	BMT_{h1_b}	BMT_{h1}	BMT_{h2}	BMT_{h3}
5	1017	5,60	6,09	6,39	6,39
10	11375	242,43	263,51	276,69	276,69
20	10243	226,61	246,31	258,63	258,63

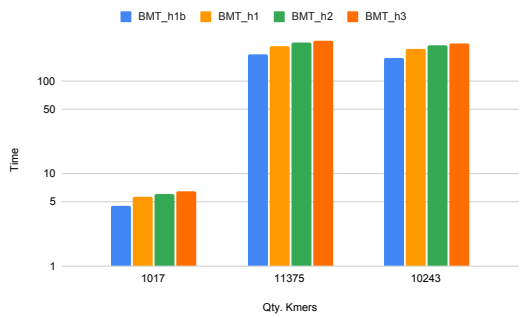

Figure 14. Average time (seconds): chart for Table 10

5 Discussion, perspectives and conclusions

This work addressed the De Bruijn Graph Sequence Problem – $BSMP$. More specifically, we were interested in designing algorithms and heuristics that could handle huge amounts of data and that are able to map a sequence into larger graphs. The best exact solution [10, 11] for the $BSMP$ was implemented and we have named it here BMT and we noticed that this version is only able to map a sequence into small graphs and this is impractical in terms of memory and BMT_{ac} (cost-only version) is capable of supporting huge graphs, but is impractical in terms of time.

Our experiments showed that BMT_{ac} takes a lot of time to return the mapping cost and our heuristics $BMT_{h1_{ac}}$ and $BMT_{h1_{acb}}$ are time efficient to process the cost and the higher the number of anchors, the greater its efficiency in terms of time and quality. $BMT_{h1_{ac}}$ is more limited than $BMT_{h1_{acb}}$ because $BMT_{h1_{acb}}$ contains a tool called Bifrost to build the De Bruijn graph and for that reason we can create much larger graphs.

In our tests the version that returns the mapping (BMT) was not able to process the tests because it consumes a lot of memory, in view of this, our heuristics are efficient because they can perform the mapping. The heuristic BMT_{h1_b} managed to be the best in terms of time and quality, followed by BMT_{h3}. The BMT_{h2} heuristic did not produce good results, but in relation to

the mapping, it is capable of processing large graphs and the greater the number of k-mers in the graph, the greater the chance of achieving good results. Finally, BMT_{h1} is the most expensive in memory and despite being able to perform the processing, it is more limited and the smaller the number of anchors, the more incapable it is to process, thus being dependent on having a greater number of anchors, however, cannot create large graphs. Our best heuristic is BMT_{h1_b} as it is the fastest and is able to process larger graphs.

References

1. E. Garrison, J. Siren, A. M. Novak, G. Hickey, J. M. Eizenga, E. T. Dawson, W. Jones, S. Garg, C. Markello, M. F. Lin, et al.: Title of Variation graph toolkit improves read mapping by representing genetic variation in the reference. In: Nature biotechnology, vol. 36, no. 9, pp. 875–879, (2018).
2. E. W. Myers: Title of the fragment assembly string graph. In: Bioinformatics, vol. 21 Suppl 2, pp. ii79–85, (2005).
3. N. G. De Bruijn: Title of a combinatorial problem. In: Proc. Koninklijke Nederlandse Academie van Wetenschappen, vol. 49, pp. 758–764, (1946).
4. P. A. Pevzner, H. Tang, and M. S. Waterman: Title of an eulerian path approach to dna fragment assembly. In: Proceedings of the national academy of sciences, vol. 98, no. 17, pp. 9748–9753, (2001).
5. U. Manber and S. Wu. Title of approximate string matching with arbitrary costs for text and hypertext. In: Advances In Structural And Syntactic Pattern Recognition, pp. 22–33, World Scientific, (1992).
6. T. Akutsu. Title of a linear time pattern matching algorithm between a string and a tree. In: Annual Symposium on Combinatorial Pattern Matching, pp. 1–10, Springer, (1993).
7. K. Park and D. K. Kim. Title of string matching in hypertext. In: Annual Symposium on Combinatorial Pattern Matching, pp. 318–329, Springer, (1995).
8. A. Amir, M. Lewenstein, and N. Lewenstein. Title of pattern matching in hypertext. In: Journal of Algorithms, vol. 35, no. 1, pp. 82–99, (1997).
9. G. Navarro. Title of improved approximate pattern matching on hypertext. In: Theoretical Computer Science, vol. 237, no. 1-2, pp. 455–463, (1998).
10. M. Rautiainen and T. Marschall. Title of aligning sequences to general graphs in o $(v + me)$ time. In: bioRxiv, p. 216127, (2017).
11. C. Jain, H. Zhang, Y. Gao, and S. Aluru. Title of on the complexity of sequence-to-graph alignment. In: Journal of Computational Biology, vol. 27, no. 4, pp. 640–654, (2019).
12. A. Limasset, B. Cazaux, E. Rivals, and P. Peterlongo. Title of read mapping on de bruijn graphs. In: BMC bioinformatics, vol. 17, no. 1, pp. 1–12, (2016).
13. G. Holley and P. Melsted. Title of Bifrost: highly parallel construction and indexing of colored and compacted de bruijn graphs. In: Genome biology, vol. 21, no. 1, pp. 1–20, (2020).