Freie Universität Berlin

Bachelor thesis at the Institute of Computer Science

Fast and exact motif discovery using the SeqAn library GenMap algorithm

Gergana Stanilova
Student ID: 4877948

Advisor: Prof. Dr. Knut Reinert
Second examiner: Prof. Dr. Alexander Bockmayr

Berlin, 09.06.2023

Abstract

Motif discovery methods are of great significance because they can point to meaningful biological conclusions. One application of motif discovery is identifying DNA regulatory elements. In particular, transcription factor binding sites since they are the target of drug therapy given their crucial role in modulating transcription mechanisms.

The planted (l,d)-motif search is commonly used as a method for benchmarking the performance of motif finding algorithms. It hunts for a l-character long pattern with d mutations in a set of sequences with a fixed length. Heuristic approaches have had great success in obtaining a common pattern with high precision in a computationally efficient way. Nonetheless, they suffer from converging to local optima in the case of unfortunate initialization.

To this end, the utilization of two heuristic algorithms is described in this thesis - the GenMap algorithm, developed by Pockrand et al., from the SeqAn-library by Reinert et al., and the Projection algorithm, developed by Buhler and Tompa. The potential of these algorithms lies in their pre-processing phases. GenMap improves a genome mappability algorithm from the GEM Suite by refraining from approximations, whereas Projection decreases the bias of uniformly distributed mutations by grouping the l-mers based on only some of their positions. In their main phase the algorithms utilize an expectation-maximization (EM) algorithm in order to identify a consensus sequence which corresponds to the motif.

The algorithms were validated by being tested for 8 different planted (l,d)-motif problems with 10 synthetic datasets for each and on promoter sequences extracted from yeast. GenMap outperforms Projection with respect to computational speed, even when the latter had been optimized with the use of multithreading. The detection accuracy is similar for both approaches and is comparable to the the one performed by Projection's creators. However, the algorithms did not perform optimally with biological data so the experimentally verified motif was mutated and implanted in sequences with a random background for further testing.

Declaration of Academic Integrity

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such. This paper has neither been previously submitted to another authority nor has it been published yet.

09.06.2023

Gergana Stanilova

Contents

1 Introduction **7**
- 1.1 Motivation . . . 7
- 1.2 Background . . . 7
- 1.3 Scope of research . . . 8
- 1.4 Outline . . . 9

2 Materials and Methods **9**
- 2.1 General program structure . . . 9
- 2.2 The GenMap algorithm . . . 15
  - 2.2.1 The SeqAn library . . . 15
  - 2.2.2 GenMap background . . . 15
  - 2.2.3 Solving the Hidden Motif Problem using GenMap . . . 15
- 2.3 The Projection algorithm . . . 16
  - 2.3.1 Parameter selection . . . 19
  - 2.3.2 Generating projections . . . 19
- 2.4 Refinement via the EM-Algorithm . . . 20
  - 2.4.1 Initializing the weight matrix W_{init} . . . 20
  - 2.4.2 Calculating the position matrix . . . 20
  - 2.4.3 Iterating until convergence . . . 21
- 2.5 Multithreading . . . 22
- 2.6 Producing a consensus sequence . . . 23
- 2.7 Inferring the positions of the motif occurrences . . . 23
- 2.8 Generating synthetic datasets . . . 24
- 2.9 Experimentally verified transcription factors . . . 25
- 2.10 Implanting verified motifs in synthetic datasets . . . 26
- 2.11 Performance metric for detecting accuracy . . . 26

3 Results **29**
- 3.1 Running the programs . . . 29
- 3.2 Tests on synthetic sequences with a synthetic motif . . . 29
  - 3.2.1 Parameter selection . . . 29
  - 3.2.2 Performance comparison . . . 29
- 3.3 Tests on biological data . . . 32
- 3.4 Tests on synthetic sequences with a motif generated based on experimentally verified data . . . 32

4 Discussion **33**
- 4.1 Performance Assessment of the Tests on Synthetic Data . . . 33
  - 4.1.1 Detection Accuracy . . . 33
  - 4.1.2 Computational Speeds . . . 34
- 4.2 Performance Assessment of the Tests on Biological Data . . . 34
- 4.3 Performance Assessment of Synthetic Sequences with a Motif Generated Based on Experimentally Verified Data . . . 34

5 Conclusion **34**
5.1 Findings 34
5.2 Limitations and weaknesses 35
5.3 Future work 35

Bibliography **35**

List of Figures **43**

List of Tables **45**

A Appendix **46**

B Appendix **47**

1 Introduction

1.1 Motivation

Sequence motif discovery has been a major scientific objective for decades since it has a huge impact on a myriad of vital biological processes. A sequence motif refers to a particular nucleotide or amino acid sequence pattern that recurs at different positions in a molecule (DNA, RNA or protein) and is thought to have a biological significance. One aspect in which motif discovery plays a crucial role is the better comprehension of gene expression mechanisms. The expression of a gene is controlled by specific proteins called transcription factors (TF). They bind to regulatory DNA segments, known as transcription factor binding sites (TFBS), which are located in the upstream region relative to the transcription start site (TSS).

Moreover, it is sensible to assume that co-expressed genes are coregulated and therefore activated by identical or similar regulatory elements. In particular, a set of genes controlled by a common regulator is called a regulon. An example of a regulon is the collection of "clock" genes. The mutation of the CRY1 gene, for instance, can result in a genetic disease by the name of Familial Delayed Sleep Phase Disorder (FDSPD) [1]. Likewise, a variety of transcription factors play a crucial role in different diseases like diabetes [2], autoimmune diseases [3], and multiple cancers [4].

To this end, drug therapy is concentrated on transcriptional gene modulation, which is, in essence, promoting or suppressing transcription. However, obtaining the necessary information experimentally, despite the recent development of high-throughput gene expression analysis technologies, is an expensive and time-consuming process. To overcome these drawbacks, computational methods are used beforehand in order to reduce the number of possible TBFS sequences. Consequently, techniques for analysis of molecular interaction, for instance, the MicroScale Thermophoresis (MST), are applied to verify the correctness of the assumptions [5].

1.2 Background

A useful abstraction of the TBFS identification problem is the planted (l, d)-motif search [6]. Pevzner and Sze formulated it as the hunt of an l-character long string, containing d mismatches, in a set of t sequences, each of length n. In general, TFBSs are typically 5-20 nt

(some even >30 nt) long [7] whereas promoter regions can consist of 100-1000 nucleotides [8]. A parameterization of this algorithmic challenge that represents the biological reality well is the so-called (15,4)-motif problem or the search for a 15 nucleotide-long motif with 4 mismatches in a sample of 20 sequences of length 600. For that reason, it is often used for reflecting the accuracy of various algorithms.

There have been numerous publications detailing motif discovery algorithms. Early algorithms mostly considered two types of approaches: word-based and probabilistic strategies.

Word-based, or also called combinatorial algorithms, employ exhaustive enumeration. Sinha and Tompa's YMF (Yeast Motif Finder) [9], as well as the rigorous Oligo-analysis, introduced by [10], rely on biological knowledge for the integration of a clever model design. Additionally, Weeder [11] and MITRA [12], both enumerative tree-based algorithms, have proven to present advantages in various aspects by the motif hunt [13].

Moreover, an example of an approach that performed well on the (15,4)-motif problem is Pevzner and Sze's WINNOWER [6], a word-based algorithm that incorporates graph theory. Further, their combinatorial algorithm SP-STAR [6] is reported to outperform some of the classical probabilistic algorithms such as the deterministic MEME [14], the randomized GibbsDNA [15], and the greedy Consensus algorithm [16].

Enumerative algorithms are suitable for exploring short patterns and ensure that vital motifs are not overlooked. Therefore, they are useful for eukaryotic genome. However, with the increasing size of the motif and of the set of sequences, they become impractical. On the other hand, heuristic models are more efficient timewise but often end up at a local maximum depending on the initial random choice of potential motives to be explored.

Despite failing to solve some of the more difficult motif finding problems, these early approaches were remarkably successful and continue to serve as the ground on which more recently developed algorithms stand. Thus, it is important to mention, some of the strategies they have adopted. Namely, the inclusion of phylogenetic footprinting or orthologous sequences in algorithms like PhyloGibbs [17]. Also, the combination of DNA sequence data and transcription factor structural information to conclude the preferences of amino acid-nucleotide recognition [18]. And ultimately, an ensemble approach called EMD [19], which comprises five other algorithms: AlignACE [20], Bioprospector [21], MDScan [22], MEME, and MotifSampler [23] and has been observed to outperform each algorithm taken individually.

1.3 Scope of research

Two heuristic algorithms are compared in this thesis. The first one GenMap, developed by Christopher Pockrandt, Mai Alzamel, Costas S. Iliopoulos and Knut Reinert, uses an approach that approximates some of the frequency values and reduces redundant searches [24]. In the second one Projection, the authors, Buhler and Tompa, get around the problem of potential bias in the initial set of motifs towards an overrepresented randomly occurring pattern [25].

The main objectives of this thesis are: first, to make use of GenMap in the context of motif discovery, second, the implementation of the Projection algorithm in the programming language C++ with the use of the SEQAN library [26], and third, the analysis and assessment of their efficiency in terms of detection accuracy and computational speed in comparison

with each other as well as with other motif discovery tools.

The analysis was performed on 8 synthetically produced benchmarking problems, founded on the (l,d)-motif search. Furthermore, in order to test the algorithm's performance on real biological sequences, the upstream regions of the some of the genes, controlled by the Gcn4 regulatory protein, were chosen for the tests. Additional tests were performed on datasets where synthetic motifs were generated based on the frequency matrix of the above mentioned protein and implanted in sequences with a random background.

1.4 Outline

First, the general program structure is described in section 2.1, while section 2.2 focuses on the GenMap algorithm, including its background and application to solving the hidden motif problem. The Projection algorithm is described in section 2.3, with details on parameter selection and projection generation. The Expectation-Maximization (EM) Algorithm for refinement is detailed in section 2.4, including initialization of the weight matrix, calculation of the position matrix, and the iteration step. Multithreading is discussed in section 2.5, while sections 2.6 and 2.7 describe producing a consensus sequence and inferring the positions of motif occurrences, respectively. Section 2.8 covers generating synthetic datasets, and section 2.9 discusses experimentally verified transcription factors, in particular Gcn4. The process of implanting verified motifs in synthetic datasets is explained in section 2.10, and section 2.11 outlines the performance metric for detecting accuracy.

The results of running the programs are presented in section 3, including tests on synthetic sequences with synthetic and experimentally verified motifs, as well as tests on biological data.

Next, in section 4, the performance assessment of the tests on synthetic and biological data, including detection accuracy and computational speeds, is discussed. The thesis concludes with sections 5.1, 5.2 and 5.3 which summarize the thesis and give perspective on its findings, limitations and possible improvements.

2 Materials and Methods

The use of the algorithms and their mathematical foundation are both explained in this section. The Github repository contains all of the files related to the thesis [27]. To compile and run the programs, follow the instructions in `README.md`. Two different projects used the two algorithms. GenMap can be found in the `genmap_motiffinder` - folder and Projection in the `projection_motiffinder` - folder.

2.1 General program structure

Running the program `motiffinder.cpp` in the `build`-folder for either GenMap or Projection, respectively, can be used to locate a hidden motif in a series of sequences. The programs have the same general steps, where they:

- use the program's input files create hidden motif candidates (using either GenMap or Projection)
- refine the candidates using the Expectation-Maximization algorithm

- create a consensus sequence
- determine the places of the motif
- compute the performance coefficient
- monitor the execution time

Flowcharts of the programs are depicted in Figures 1 and 2.

The number of input arguments is dependent on the type of sequences - they are either synthetic or biological. The synthetic ones are generated with other programs (see Sections 2.9 and 2.10). Four input arguments are necessary for tests with synthetic data and two with biological. They are illustrated on the top of Figures 1 and 2 in grey:

- arg 1: fasta-file containing t sequences in which a motif could be hidden (an example in Figure 3)
- arg 2: csv-file containing the parameter values for l, d, k, s and m (seeFigure 4 for an example), where where l is the length of the motif, d is the maximum number of mutations the motif can have, k is the length of the projection of the original l-mer, s is the threshold for the Projection algorithm and m is the number of trials (the usage of these parameters is explained in Section 2.3)
- (optional) arg 3: csv-file containing the precise sites of the implanted motif and mutations (only in the case of synthetic sequences) (for an example see Figure 5)
- (optional) arg 4: the number of datasets to be processed (when testing with multiple synthetic datasets)

The `README.md`-file file located in the Github source contains a detailed description of the syntax. Sections 2.8, 2.9, and 2.10 go into detail about how the files were made.

The program reads the fasta-file using a Fai index from the SeqAn library and generates a consensus sequence (see Section 2.6) based on the desired algorithm using:

- the function `runGenMap` for the GenMap algorithm, illustrated in Figure 1 in green (see Section 2.2.3).
- the function `runProjection` for the Projection algorithm, illustrated in Figure 2 in blue (see Section 2.3)

The obtained optimal consensus sequence (see Section 2.6), the inferred positions at which it occurs (using the functions on Figures 1 and 2 shown in purple and described in Section 2.7), the average run time for one trial, the total run time, and the average performance coefficient (described in more detail in Section 2.11 and shown on the bottom of Figures 1 and 2 in grey) are all displayed in the results, which are then printed to the console. Figure 6 displays an example output.

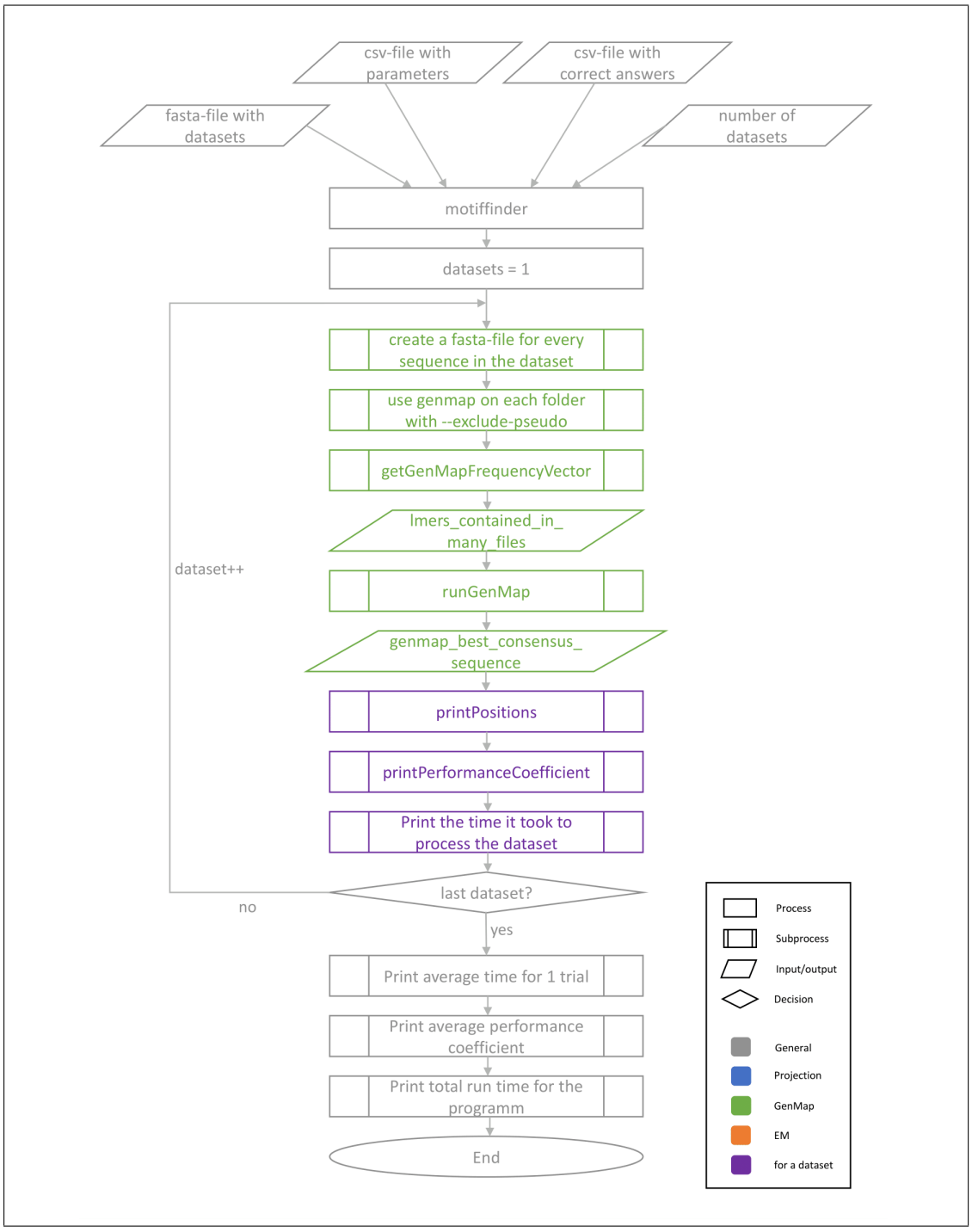

Figure 1: A flowchart of the GenMap `motiffinder.cpp`

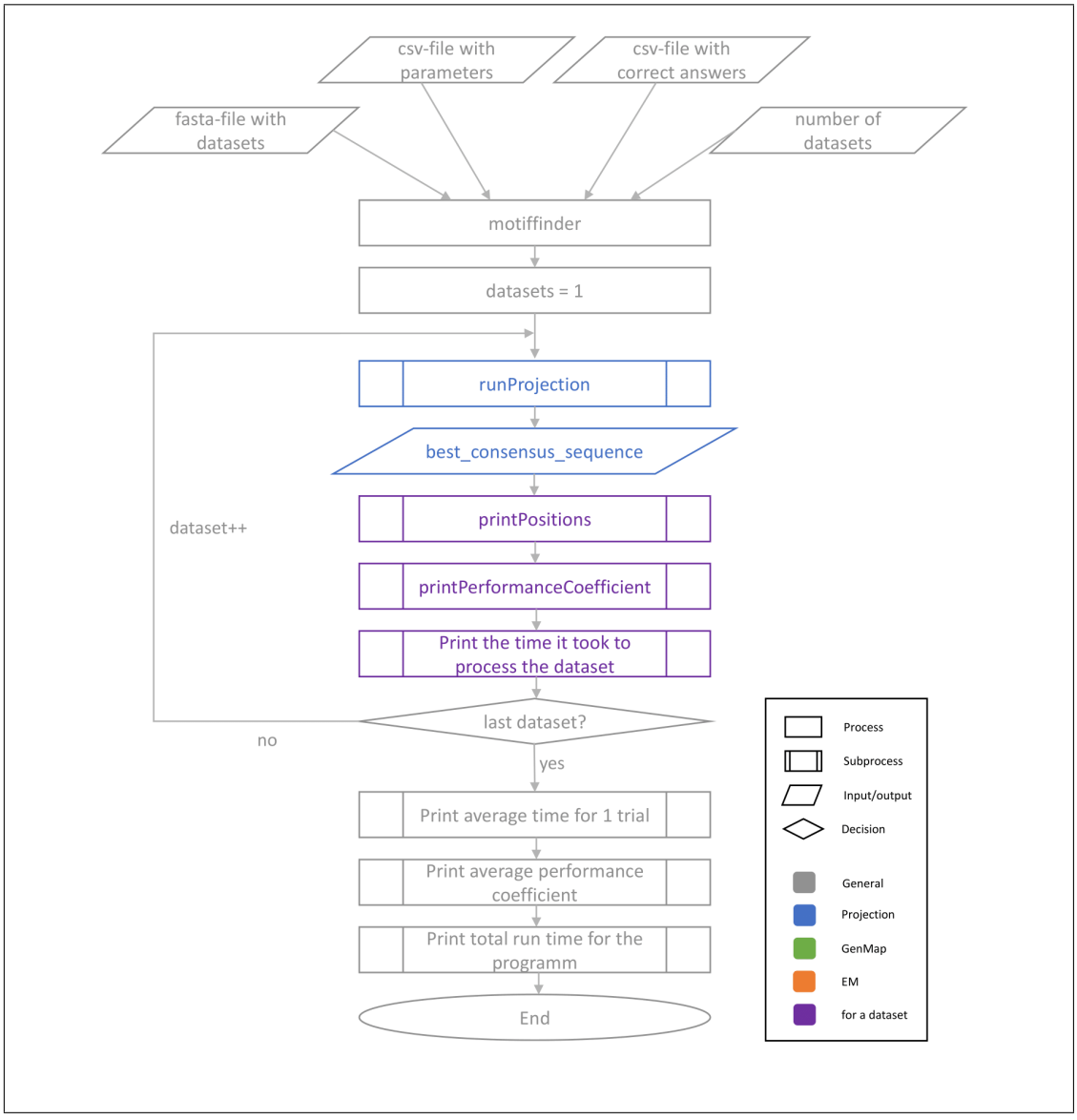

Figure 2: A flowchart of the Projection `motiffinder.cpp`

```
>seq0
ACAGTGAACTAATTTTTAGGGCCCCGTAGACGCTGTTGGGCGGGCTGTGGAGGCACATAGTTGAGAGGAG
TACTAGTCCAGCGATAGCGCTACTGCCCCGAAGCATGTCGACATCCGGCGATGCTTTAATTACCCGCCAG
AAGGACATATAGCTCTCTATGAATTTGTATGCACCGCGCAATCGTTCGTCCAGGTAGCTATGCATGCATA
CACATGAAGAACCGGGGTACAATGCAATTGGTTCAAATTTGATCATGGAGAGAACTGGCGGTTTATTGTT
TCTACGCTATCCGAGCGACTATGAGGTCGAATAACGATGCTAGTGACGCTCCCATTTTTCGAAGCACAAT
TTATACCCCGACGGATTCGACGATAGGAGCCTAGTAAAGGCCCCGATGTTGTGATCGCTATGCGTCTTGA
TAGACCAACACCAAAATCACAATGGCTCAAGATGCGCTCATCGGTGGTGATAGACTAACAGTAACGTTTC
GAGCATTCACCGCTGCGATTGCCACTACACACAGAGGTGATTAGCGACTATCCCTTCCGGAGGCATTAGA
TGCTTGACTAGGAGACACAGCGTGCGCACGATGTCACGGC
>seq1
TCCCGAGTGGCACGAGACTAACATGTAACCTCGGCGACTGGAAAGTTGCGACCACCACTTTTGCTTCGCA
AAATCCAGACGTTCTGAGGATAGCTCGAGCTTAAAAGATTTATTATTACTATTAACAGATCGTACCTTCT
GTTTTGACAGAGGTGTGCACACGGTCTTTTCACGGGCATACCCGAGTTTTCGCTTCGTCGTGTGGCTGGG
CCATGTCGGAGTGCTCGCTTTCGACAACTCCCCTAAGGCACAAGCCTGAAAAAGATCGGAAGGACAAAAG
AACCCCCCACATGGCTACTGTTCGTGGCAGCCTTCGGTATTAGTTAGAATGTGTTTTATTATGCGCAGCC
AATCGGCTGTATTCCGTCATCCGCGAAGACTCGGATGGGGTTATCGTCTTGCTCCGGGCGTGAAGAGCAT
GCTACCCGTCCTCTGCGCCAACTAGCCTTTCGTCATAAGAAGTGGGCCACGAGTGGTACCTAAAGTCCTG
AGGTAAATGCGGCATCTAGTATTCGGAAATCGCGTTCGCTTGCTTGAGGCGAAGTCGAGTATTATCGGTG
TCCCATTTGAAGGTTACTCAGCGTCTGCGACGGGCTACTA
>seq2
TATGCATCATGAGAGGGGGTCAAAGCGTAGGTAAACTCACCCCAGTGCTACTCTGCTAAAGTGGACGGCG
ATACCACATCGGGATATACGTTGTACTCTCCTAGTCAGCCTAACCCTACCTCTACCATTCGTATGCGACT
CGCCGACATGGGGACTTCGAGACGCCTCGAGACCTCTCATACGGCGAGCGATAGAATTTGGTGTCCTGGT
GTCTGCGTAGCCCACGTTCACAGCTCCTCTGAGAACTGGAAGGTGGACGTGGCAGGTAAAACGCCTGGTT
AAGCAACGTTTAATGCACAGTAGATGGCTTCTATGGCAGTTCGGTAAACGAATGTCAACAGATTGGAACA
ATACTGCGTCCAACCTCCAGAGTACTTGATAATACTGTGTTGCATCTACGAGACCCCGAGCCGTAACGTG
GAGCTTAGGCTAGCTAAGCCTTTAATCTTCGGGTGTCTAGAACACACAAAATCATAGTAACACACTTTTA
ATAGCGGCTGTTACGTTAGCCAAGCGACGCCCTACTAGCGAAACAGGGGGGCTAACAGAGGACGTTCCAC
GGTGAAGCCATGCATGGCCCCACTCCTAAGATAATATAGT
```

Figure 3: An example of a fasta-file with 3 sequences each of length 600 in which a motif could be hidden

l	d	k	s	m
10	2	7	4	72

Figure 4: An example of a csv-file with the parameter values for l, d, k, s and m

seq	pos	motif	mutations	mut_pos
X	X	GATGTCGAGA	0	
0	103	CATGTCGACA	2	|8|0
1	540	GAAGTCGAGT	2	|2|9
2	152	GACTTCGAGA	2	|3|2
3	66	GATGTGGAAA	2	|5|8
4	539	CATCTCGAGA	2	|3|0
5	356	GATGTCAAGG	2	|9|6
6	448	GATGTTGTGA	2	|5|7
7	136	GATGTCGAAC	2	|8|9
8	552	GATCACGAGA	2	|4|3
9	432	GATATCGAAA	2	|8|3
10	49	GGTGTCGTGA	2	|1|7
11	56	GGTCTCGAGA	2	|1|3
12	395	GATGCCGGGA	2	|4|7
13	441	GTTGTCGATA	2	|8|1
14	336	GATGCCTAGA	2	|4|6
15	262	GAGGTCGAGG	2	|2|9
16	504	GATGACTAGA	2	|4|6
17	581	GATGTTCAGA	2	|5|6
18	302	GTTGTCGCGA	2	|1|7
19	479	GATGTGTAGA	2	|5|6

Figure 5: An example of a csv-file with the correct implanted motif and mutation positions

```
searched consensus sequence: [TACCACCAAC] with a score of 4
Matches:
0: { [228 => TAACACCGAC] }
1: { [321 => TTGCACCAAC] }
2: { [307 => TAACACCAAA] }
3: { [399 => TACCACCTTC] }
4: { [301 => TCGCACCAAC] [211 => AACTACCAAC] }
5: { [172 => CACCATCAAC] }
6: { [109 => TACCGACAAC] }
7: { [422 => TACCTCCAAG] }
8: { [185 => TGCCACCAGC] }
9: { [253 => TAGTACCAAC] }
10: { [134 => TACGACCAGC] [399 => TCTCACCAAC] }
11: { [385 => GACCACTAAC] }
12: { [380 => TACTACAAAC] }
13: { [222 => TCCCACCAGC] [513 => TACGACTAAC] }
14: { [135 => TACCACCAGC] [481 => TAAAACCAAC] }
15: { [63 => TAACAACAAC] }
16: { [54 => TACCACCTCC] }
17: { [223 => TACCACACAC] }
18: { [476 => TACCAGGAAC] }
19: { [111 => TACCACAAAT] }
The performance coefficient is 1.
Processing this dataset took 36 seconds.

The average time for one trial was 0.4186 seconds.
For 10 out of 10 dataset(s) the correct planted motifs were found. The average performance coefficient is 1
Execution of program finished after (hh:mm:ss) 00:06:12.
```

Figure 6: An example output of the GenMap `motiffinder.cpp`

2.2 The GenMap algorithm

2.2.1 The SeqAn library

Both algorithms are implemented with the aid of the SeqAn2 library [26]. SeqAn is an open source C++ template library with many applications in the field of biological sequence analysis which is implemented in an efficient and reusable way. This is achieved by employing the concept of generic programming, specifically, by using templates as they have the property of maintaining a high degree of abstraction which proves useful since it reduces the memory and computational costs of many operations [28]. Instructions for the installation of the SeqAn library can be found on its website [29].

2.2.2 GenMap background

GenMap is an algorithm created with the purpose of computing the genome mappability and by extention the (l, d)-frequency [24] - this is the frequency with which an l-mer occurs in a sequence with up to d mismatches. It builds up on an algorithm, implemented in the GEM Suite [30], [31], by making three main improvements on it:

- it does not process every single l-mer separately since it is highly similar to its adjacent l-mers, therefore the number of l-mers is reduced
- searching for approximate matches is performed by utilizing a Bidirectional Index Search (opposed to a Backward or a Forward Search for example), which reduces the number of search steps
- redundant l-mers are skipped by avoiding the computation of the same l-mer multiple times, especially those from repeat regions, and setting all their frequency values accordingly

2.2.3 Solving the Hidden Motif Problem using GenMap

For the purpose of motif discovery, the GenMap algorithm is adapted to find initial probable occurrences of the motif, which are later refined. The objective is detecting exactly one motif occurrence per sequence (OPS), therefore the `–exclude-pseudo` - command is the most fitting choice for processing the input sequences [32]. It is a feature that allows the mappability algorithm to only count the number of indexed fasta-files that contain a l-mer at least once, excluding any l-mers that occur multiple times in a single fasta-file. Accordingly, each sequence in a single fasta-file is split up in a separate file and grouped in a folder.

The `getGenMapFrequencyVectorOPS`-function is then used to create an index file for the input fasta-files with the command:

"genmap index -FD /path/to/directory/with/fasta/files -I /multi/fasta/index/output;"

The SeqAn-GenMap program is then executed by running:

"genmap map -E no_of_mismatches -K motif_length -I /multi/fasta/index/output -O /path/to/output/folder -r -fs -ep;"

where:

- "-r" directs the program to output raw files (the binary format of std::vector<T> with T = uint8_t). For each fasta file that was indexed a separate file is created. File type is .freq8.

- "-fs" means that frequencies are stored by using 8 bit per value
- "-ep" corresponds to "–exclude-pseudo"

After that, the `load`-function is used to load the raw files [33] and save them in a vector `genmap_frequency_vector_freq8`. Each value of this vector represents the frequency with which an l-mer occurs in the sequence with up to d-mismatches [34]. This vector is then converted into integer values for further processing.

In biological data the hidden motif may not occur in every sequence so the variable `min_no_of_files` was introduced, which represents the minimum number of all files in which a motif has to be present calculated as a fraction of all files for a dataset.

The `runGenmap`-function regards only the l-mers that are contained in at least that many files (see the flowchart in Figure 7).

The function then iterates through the starting positions, converts them to buckets, initializes a weight matrix, and refines it until convergence (see Section 2.4). It then gets the consensus sequence for each bucket and chooses the one with the best score. Finally, it returns the best consensus sequence as a DNA string (see Section 2.6).

2.3 The Projection algorithm

The Projection algorithm has an important benefit in comparison to other motif discovery algorithms. In particular, to those with initialization techniques that first try to randomly guess one or some of the motif occurrences and then select l-mers similar to the first choice(s). The disadvantage of this approach lies in the possibility that the initial guess is actually inaccurate and therefore every successively chosen l-mer is more similar to a randomly occurring motif than to the true one. The Projection algorithm accounts for this by adopting a sophisticated initialization strategy that increases the likelihood that the true motif will be explored in the first place.

It considers the assumption that the mutations in a motif occurrence are uniformly distributed. Therefore, it categorizes the similarity between l-mers by ordering them into buckets. It does that by considering only a k number of positions in the l-mer and assigning a hash value to the new string, the so-called projection, that is derived by concatenating the characters at those positions.

After that, it utilizes an Expectation-Maximization (EM) algorithm, a widely used optimization model, for finding the positions of the most probable motif occurrences in each bucket with sufficiently enough elements. In the end, it outputs a consensus sequence representing the motif. A pseudo-code of the algorithm is illustrated as follows:

```
1 Projection(l, k, s, m, d, [seq1...seqt])
2 For i=1 -> m do
3       kpos = k random, distinctive positions between 0 and l
4       For each possible l-mer x of seq1...seqt do
5             HashValue = hash(x,kpos)
6             Add x to hash bucket at position HashValue
7       For each bucket with more than or s elements
8             Refine bucket with EM algorithm
9 Choose best consensus sequences of best bucket
```

It is complemented by a graphical illustration shown in Figure 8, and a more detailed explanation follows in the next subsections.

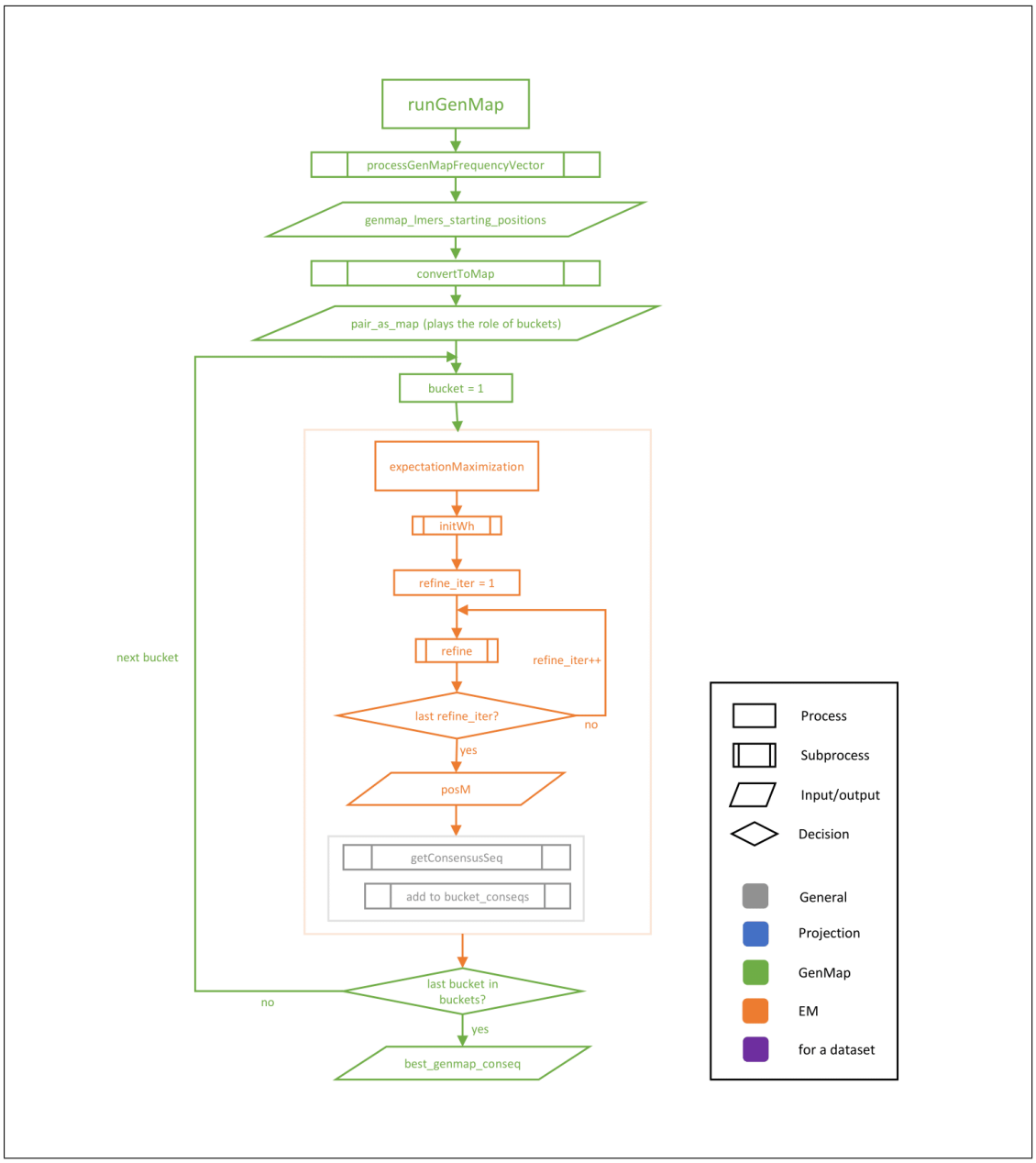

Figure 7: A flowchart of the implementation of the function `runGenmap`

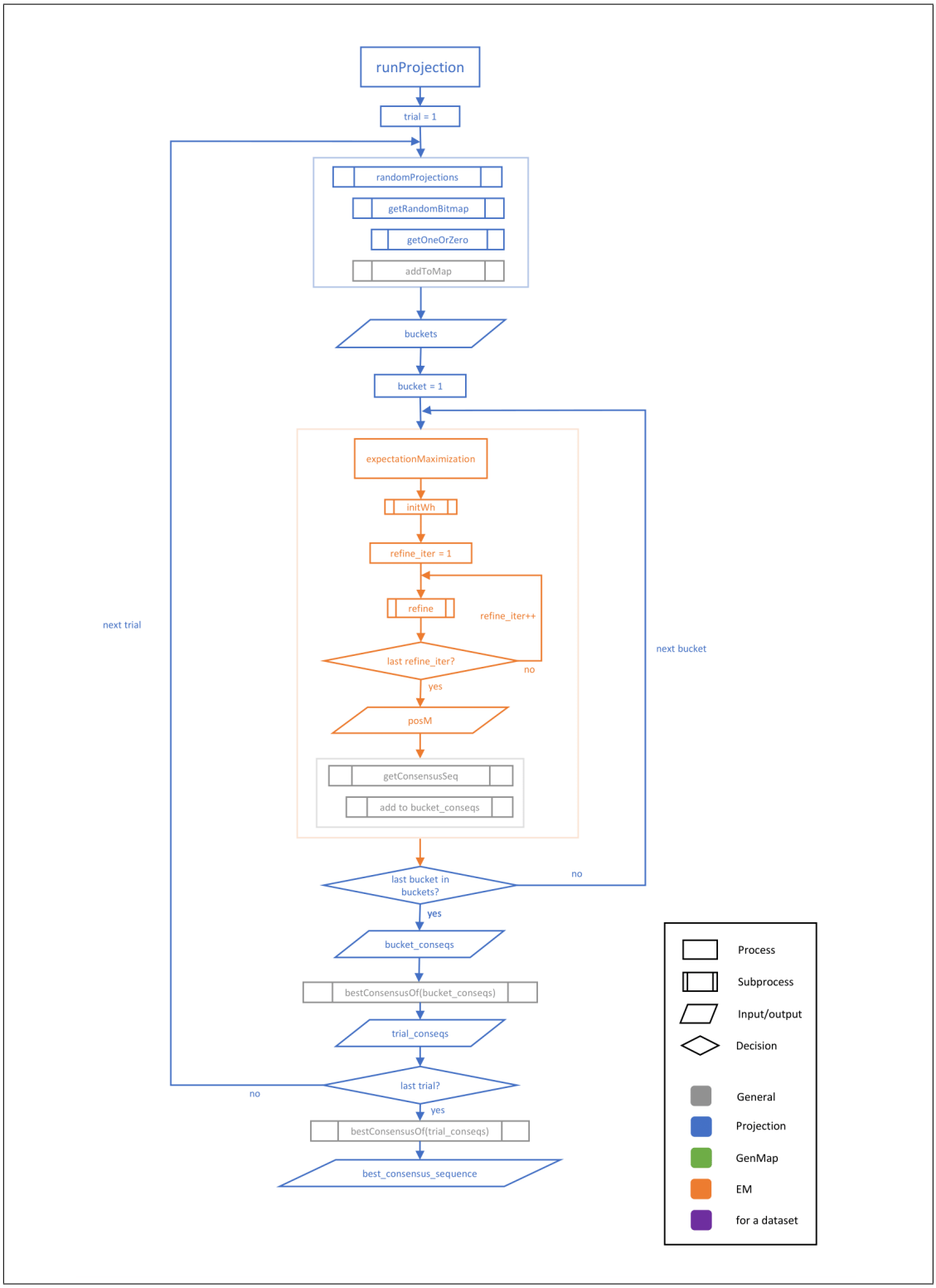

Figure 8: A flowchart of the implementation of the function `runProjection`

2.3.1 Parameter selection

As already mentioned some parameters are chosen based on biological knowledge, others, however, need to be inferred statistically. Buhler and Tompa argued that a reasonable range for the value of k is:

$$\log_4\left(\frac{t(n-l+1)}{E}\right) \leq k < l-d \tag{1}$$

E represents the expected number of l-long background patterns that occur with up to d substitutions at least once in each of t sequences and is approximated by the equation:

$$E(l,d) = 4^l(1-(1-p_d)^{(n-l+1)})^t \tag{2}$$

with the probability p_d that an l-mer occurs with up to d substitutions at a given position in a random sequence :

$$p_d = \sum_{i=0}^{d}\binom{l}{i}\left(\frac{3}{4}\right)^i\left(\frac{1}{4}\right)^{l-i} \tag{3}$$

Logically, the most desired value for E is smaller than one. Nonetheless, in the Discussion subsection, some examples are presented that deal with the cases in which E is bigger than 1 and with the repercussions that has on the algorithm's performance.

Another parameter, the number of trials m, is set in such a way so that there is a high probability of enough occurrences hashing to the bucket of the planted motif. Buhler and Tompa name this probability q and they set it to 0.95. The calculation of m is represented by the equation:

$$m = \left\lceil \frac{\log(1-q)}{\log(B_{\hat{t},\hat{p}(l,d,k)}(s))} \right\rceil \tag{4}$$

$B_{\hat{t},\hat{p}(l,d,k)}(s)$ denotes the Bernoulli formula (5) where $\hat{p}(l,d,k)$, Equation (6), is the probability with which each motif occurrence in the planted model hashes to the planted bucket.

$$B_{\hat{t},\hat{p}(l,d,k)}(s) = \binom{\hat{t}}{s}\hat{p}^s(1-\hat{p})^{\hat{t}-s} \tag{5}$$

$$\hat{p}(l,d,k) = \frac{\binom{l-d}{k}}{\binom{l}{k}} \tag{6}$$

For the value of s, the authors take the empirical value from their experiments which is s=3 or s=4.

2.3.2 Generating projections

The algorithm executes m trials as a whole, in each of which it projects an l-mer onto a k-mer by choosing k positions from the l-mer, randomly and uniformly. It then hashes the projections and orders them into buckets.

A useful tool for creating and hashing projections is the Q-gram Index structure from the SeqAn library. A q-gram is a string of length q. For the purposes of the Projection algorithm, a q-gram is used as a k-mer. To this end, a Q-gram Index is specialized with a specific Shape

type which gives the opportunity to insert gaps between the characters of the q-gram and therefore to choose the wanted k positions in an l-mer. The projected positions, chosen randomly and uniformly, are represented by '1'-s and the irrelevant positions by '0'-s, and saved into a String of integers called bitmap.

In order to explore each of the t sequences, we need to use an index constructor with a Q-gram Index as an Index type. The Q-Gram Index uses a GenericShape, which must first be initialized with a valid shape. To do so, the function stringToShape is called with a Shape object and the previously mentioned bitmap. After that, hash values are calculated for the projection of each l-mer with the SeqAn hash-function.

For grouping the hashed projections a map called *buckets* was created with the hashValue as the map-key and a vector of pairs as the map-value. The first element in each pair gives the number of the sequence, and the second - the starting position of the projected l-mer.

This phase was implemented as the function `randomProjections`.

2.4 Refinement via the EM-Algorithm

Only the buckets containing many elements are of interest for the refinement stage. For that reason, a threshold s is defined. It represents the minimal number of elements a bucket needs to have in order to be explored further.

The refinement (implemented as `expectationMaximization`) is done via the EM algorithm whereby the input sequences serve as the given data and the positions of the motif in these sequences as the missing information. First of all, an initial weight matrix is computed which is later on used in the E-step of the algorithm to estimate a position matrix. Subsequently, in the M-step the weight matrix is optimized using the position matrix. These steps are executed until convergence.

2.4.1 Initializing the weight matrix W_{init}

For every bucket with size greater or equal to the threshold s, an initial weight matrix W_{init} is initialized with zeros. Then $W_{init}(i, j)$ is set to be the frequency of base i among the jth positions of all l-mers in the bucket. Each time base i occurs in the l-mers, $W_{init}(i, j)$ gets incremented by 1 to get the absolute frequency. For the relative frequency, this number is divided by the number of elements in the bucket. Afterwards, a background probability distribution P is added in order to avoid "0"-probabilities. By default, the Laplace correction is used, for which P = (0.25, 0.25, 0.25, 0.25).
Implemented in the code as `initWh`.

2.4.2 Calculating the position matrix

The EM method weighs the extent to which the sequences could be explained based on the information from the current weight matrix $W_{cur}(i, j)$ in comparison to the background model taken separately. That relation is illustrated by the likelihood ratio:

$$LR(W_{cur}(i, j)) = \frac{Pr(sequences|W_{cur}(i, j), P)}{Pr(sequences|P)} \tag{7}$$

In the position matrix *PosM* the probabilities that the motif starts at a given position in each sequence are stored. The goal is to estimate this missing information based on the initial weight matrix W_{init} with the following formula:

$$PosM(i,j) = \frac{Pr(sequences|z_{ij} = 1, W_{cur}(i,j))}{\sum_{k=0}^{4} Pr(sequences|z_{ik} = 1, W_{cur}(i,j))} \tag{8}$$

Where j is the starting position of the motif in sequence i when $z_{ij} = 1$. The matrix was created by iterating through the sequences and for each l-mer's characters multiplying the corresponding probabilities from the $W_{cur}(i,j)$ matrix. These values were saved as numerators, then summed for the denominators and in the end the numerators were divided by the denominators.

The implementation of this step is done with the function `refine`.

2.4.3 Iterating until convergence

The current weight matrix $W_{cur}(i,j)$ was refined into W_{new} with the following formula:

$$W_{new}(i,j) = W_{new_{i,j}} = \frac{W'_{new_{i,j}}}{\sum_{i=A,C,G,T} W'_{new_{i,j}}} \tag{9}$$

for the first element of the matrix, where $W_{new_{i,j}}$ is normalized by dividing $W'_{new_{i,j}}$ through $\sum_{i=A,C,G,T} W'_{new_{i,j}}$. $W_{new_{i,j}}$ is the probability that base i is the jth letter in the motif and $\sum_{i=A,C,G,T} W'_{new_{i,j}}$ is the sum of the probabilities for every base being the jth letter in the motif.

To refine the current weight matrix $W_{cur}(i,j)$, a new weight matrix $W_{new}(i,j)$ is created. Let $M_{seqs}(i,j)$ be a two-dimensional matrix of the sequences, where each row represents one sequence and each column a position in the sequences. The program then needs to iterate through a window $M_{win}(i,j)$, a matrix that is a subset of $M_{seqs}(i,j)$ projected onto $M_{seqs}(i,j)$. The width of $M_{win}(i,j)$ is n-l+1. Thus, $M_{win}(i,j)$ can be projected onto $M_{seqs}(i,j)$ l times. For j denoting the column of $M_{seqs}(i,j)$, $M_{win}(i,j)$ is first projected on position j=0 of $M_{seqs}(i,j)$ and then shifted l-1 times to the right. The variable j represents the jth of the motif. The purpose of the window is to wrap every character in $M_{seqs}(i,j)$, that could possibly be at the j-th's position of the motif, which explains the windows width. $M_{seqs}(i,j)$ and $PosM$ have the exact same size.

For every position j that $M_{win}(i,j)$ is projected onto $M_{seqs}(i,j)$, the program iterates through $M_{win}(i,j)$. Let *row* be the number of the current row and *col* the number of the current column of $M_{win}(i,j)$. Then for every *row* every *col* of $M_{win}(i,j)$ is viewed. The numeric value B_{nv} of the base (0 for A, 1 for C, 2 for G, 3 for T) at the position $M_{win}[row][col]$ is the row of $W_{new}(i,j)$ and j the column ($W_{new}[row][j]$). The program then takes the value of the position matrix *PosM[row][col]* and adds it to the value in the weight matrix at $W_{new}[B_{nv}[j]]$.

After that the program iterates through the previous weight matrix W_{cur} and adds all its values to the new weight matrix W_{new} at the corresponding positions. W_{new} at this point is filled with every numerator (represented as W'_{new} in Formula (9)).

The denominators ($\sum_{i=A,C,G,T} W'_{new_{i,j}}$) are the sum of every column of W_{new}, so the program needs to iterate through W_{new} and collect the denominators. Subsequently, the program iterates one last time through W_{new} and sets each position to its current value divided by the corresponding denominator.

The number of iterations of the EM-algorithm correspond to the variable `refine_iter`.

2.5 Multithreading

To achieve faster execution on multicore systems, the program was implemented to use v threads for the refinements (worker threads), one thread for the main function (main thread) and another thread for printing the progress onto the console (printer thread). The variable v represents the number of available threads on all cores. The library pthreads was used to achieve the desired behavior. To assign the range for each thread, the program takes the number of buckets iter, which is also the number of iterations it has to make. The common span *span* for each thread is then $span = \left\lfloor \frac{iter}{v} \right\rfloor$. The remaining tasks get distributed from first to last, which can never be more than one task for each thread. So the first (iter mod v) threads make *span+1* and the rest of the threads *span* iterations. One example is given in Figure 9, where 11 tasks get divided over 3 worker threads. Floor(11/3) equals to 3, (11 mod 3) equals to 2. So the first 2 threads need to do 3+1 tasks, while the rest do only 3. Also, it has been considered, that each thread handles only consecutive tasks.

The calculation is demonstrated with the following pseudo-code:

```
1  NoOfThreads    = getAvailableThreads();
2  NoOfIterations = numberOf(Buckets);
3  Span      = floor(NoOfIterations / NoOfThreads);
4  Remainder     = NoOfIterations mod NoOfThreads;
5
6  For Th = 0 --> NoOfThreads do
7    TStart = Th*Span;
8    TEnd   = TStart+Span-1;
9    if(Th  < Remainder)
10     TStart = TStart+Th;
11     TEnd   = TEnd  +Th+1;
12   else
13     TStart = TStart+Remainder;
14     TEnd   = TStart+Span-1;
15   Thread T = spawnNewThread();
16   let T do motif refinement for buckets from TStart till TEnd;
```

Since the threads need to add elements to a shared vector, it was necessary to protect it against race conditions. If two threads write to the same vector at nearly the same time, the program will exhibit unexpected behavior. Protection is achieved by using a mutex lock, which is specifically made to solve race conditions. The lock is put around the add operation as so:

```
1  Mutex.lock();
2  Vector.push_back();
3  Mutex.unlock();
```

The underlying idea is that once a mutex is locked, every thread that tries to lock the same mutex is told to wait, until it gets unlocked again. This way only one thread at a time can add elements to the vector.

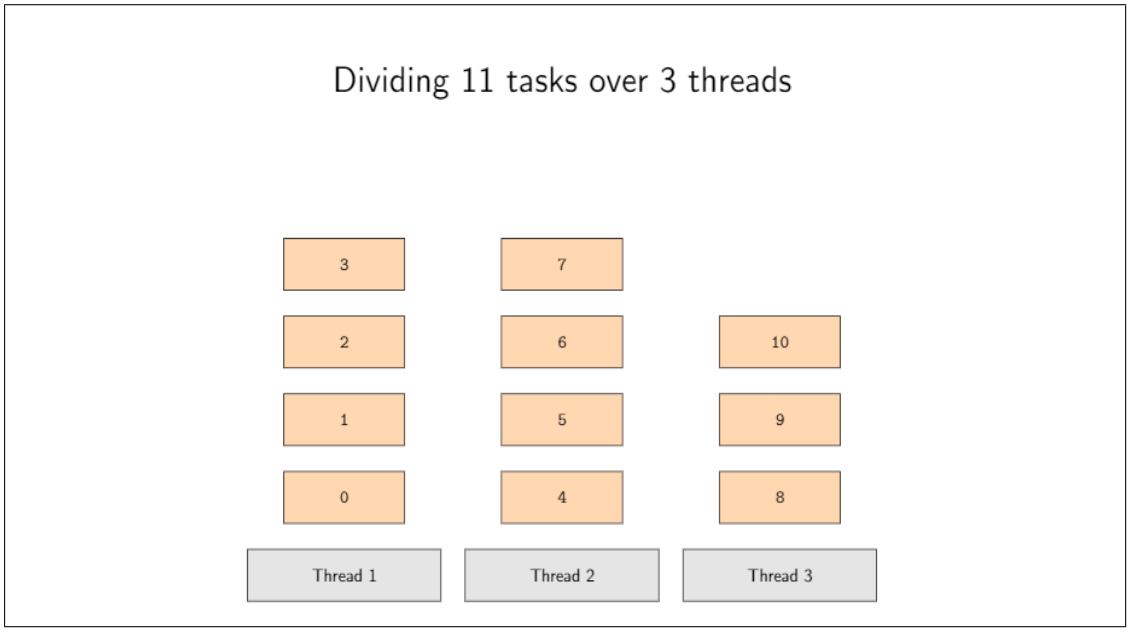

Figure 9: A multithreading example.

For certain operations there are lock-free alternatives that have a great performance advantage over mutexes. One such alternative is used to track the number of processed buckets. The variable *done_buckets* is defined as an atomic<int>. Objects of atomic types are not at risk of undefined behavior since the cases of simultaneous actions on the object from different threads are well defined. Atomic objects though are only available from C++11 and up.

2.6 Producing a consensus sequence

After the position matrix has converged, the estimated most probable starting positions of the motif can be used to extract an l-mer from each of the t sequences (see function `getConsensusSeq`). These l-mers are saved in a stringSet *T* from which a consensus sequence *Ct* is constructed by taking the most frequent character at a given position among all l-mers. Moreover, for each bucket a value *s(T)* is calculated which equals to the number elements in T whose hamming distance to the consensus is greater than d. The consensus sequence with the lowest *s(T)* value among all buckets is kept (with the help of the function `bestConsensusOf(bucket_conseqs)`). *S(T)* is further minimized over all trials to retrieve the best consensus sequence (`bestConsensusOf(trial_conseqs)`).

2.7 Inferring the positions of the motif occurrences

The motif finding algorithm implemented for this thesis follows the OOPS (one occurrence per sequence) model. On account of that, only one position in each sequence is considered as the starting position of the motif instance.

Inferring the positions of the motif occurrences is done by calling the SeqAn's find method. It utilizes Optimal Search Schemes by using a bidirectional FM-index which partitions the pattern into multiple pieces and allows the search to be performed more efficiently [35].

After obtaining the consensus sequence the find method was adjusted to search for it with up to d mismatches in each of the t sequences. Thus, the lower bound for allowed errors was passed with the value of 0 and the upper bound with d by using a switch statement. The Hamming Distance was chosen as a distance metric and the index was set as a Bidirectional FM-Index.

The implementation is done via the function `printPositions`.

2.8 Generating synthetic datasets

Evaluating the performance of the program can be done if there are tests performed on synthetically generated sequences where the correct consensus sequence of the motif and the positions, at which it occurs with up to a certain number of mismatches, are known. Generating the synthetic datasets is done with the program `createdatasets.cpp` [36].

Its general syntax is "./createDataset -key=value" where key and value are listed in Table 1. If a key is left out, then the default value is taken.

key	default value	description
t	20	the number of sequences
n	600	the length of a sequence
r	10	the number of datasets to be created
l	11	the length of the planted motif
d	2	the number of mutations in each planted motif
prefix	syn	the prefix for the files
help		displays this overview

Table 1: The possible key arguments with their default values and description for the program generating the synthetic datasets.

Executing the program with the default values for t, n, r and prefix for each (l,d)-motif problem results in the creation of a a csv-file for each fasta-file with a name of the form "syn_planted_motif_l_d" where l and d are replaced with the values for each instance and at the end, a suffix from 1 to r is added. The csv-file contains the number of the sequence, the position of the implanted motif, the synthetically generated motif itself, the number of mutations and the positions in the motif that were mutated. The consensus sequence can be found on the second line of the file. And the fasta-file each sequence is preceded by a line starting with the ">" character, which indicates the sequence identifier. After the identifier line is the sequence with the implanted motif.

The first step in the program is the creation of the planted motif. Each nucleotide is produced as an integer by a uniform random distribution generator and converted into a character. Then, the planted motif is mutated. The mutation positions are randomly generated and for each of them a character from the original planted motif is exchanged (without replacement) with another character which represents one of the other three nucleotides. After that, the positions at which the motif will occur in each sequence are again randomly chosen. Lastly, the t sequences are created by successively concatenating a random character up to the position at which the motif should occur, then appending the mutated motif and from then on continuing with the concatenation of random characters until the end of the sequence.

Figure 10: Sequence logo of the Gcn4 transcriptional activator. The x-axis represents the position in the motif. The y-axis shows the degree of conservation at the given position measured in bits [40].

2.9 Experimentally verified transcription factors

For the transcription factor binding site experiments, the positive transcriptional activator Gcn4 extracted from the yeast species Saccharomyces cerevisiae (strain S288C) was chosen. During a study investigating the regulation of gene expression during amino acid starvation in yeast, the authors conducted transcriptional profiling experiments and identified Gcn4p as a master regulator of gene expression in this context [37]. The study found that genes involved in amino acid biosynthesis, transport, and metabolism are regulated by Gcn4p. The promoter regions (-601 to -1) of the following of the genes mentioned in the article were extracted (sorted by gene function):

- Amino Acid Biosynthesis: ADE8, ARG4, HIS4, HIS5, LEU3, LPD1, LYS14, MET28, SHM2
- Amino Acid Transport and Metabolism: AGP1, ATR1, GAP1, URE2
- Signaling Pathways: TPK1, TPK2, MTD1

Gcn4 recognizes the 11 nucleotide-long sequence (5'-NNTGASTCANN-3') where the letter N stands for "any base" and S for either C or G [38]. The sequence logo of the binding site sequence is shown on Figure 10. Because there is some variability in the TFBS sequence, caused by mutations, it is reasonable to reflect it properly. Motifs can be illustrated in diverse ways. Some of the more common ones are a position frequency matrix (PFM) and a consensus sequence with a corresponding sequence logo [39]. A PFM describes the frequency with which a nucleotide occurs at a certain position in the motif whereas a consensus sequence only shows the most common nucleotide at that position. Sequence logos depict graphically the degree of the nucleotide conservation along the motif by measuring entropy. This is done by creating a two-dimensional plot where the positions are represented on the the x-axis and the information content on the y-axis in the form of bits. Bits are calculated according to the formula for the total entropy:

$$\sum_{i=1}^{n} p_i \log_2\left(\frac{1}{p_i}\right) \tag{10}$$

where p_i stands for the probability/frequency of a nucleotide i being at that given position. Therefore, the more conserved a position is the higher it appears on the plot.

The extraction of the promoter regions was done in four steps:

1. Step: searching for the desired gene in the National Center for Biotechnology Information (NCBI) [41]. The link to each of the mentioned genes can be found on Tables 2 and 3 in the column "Link to the gene sequence".

2. Step: by using the *Genome Data Viewer*-tool in the section "Genomic regions, transcripts, and products":
   - clicking on the panel menu option "Tools" -> "Markers"
   - giving in the desired range in the field "Pos/Range"
   - clicking on the panel menu option" Download" -> "Download FASTA" -> "FASTA (All Markers)"
3. Step: renaming the file and first line in it to the include the name of the gene. A link to each file can be seen on Tables 2 and 3 in the column "Link to the promoter region".
4. Step: adding each promoter sequence to create one file [42].

2.10 Implanting verified motifs in synthetic datasets

Motifs were generated using a Jaspar-frequency matrix and planted in random background sequences using the program `implantmotif.cpp` [43].

The Jaspar-frequency matrix was rewritten as a two-dimensional vector of double values which is then normalized by the maximum value in the input vector. The new vector represents probabilities of nucleotides at each position and serves as input for generating a random string of nucleotides (the motifs). Sequences with random background are then generated and the motifs are implanted in them at random positions. Similarly, to the output of `createDataset.cpp` creates a fasta-file and a csv-file for each dataset. However, there is a difference in the csv-file's structure which is the lack of the number of mutations and mutation positions.

2.11 Performance metric for detecting accuracy

The metric used for the detection accuracy measurement is the performance coefficient defined by Pevzner as:

$$\frac{|K \cap P|}{|K \cup P|} \tag{11}$$

where K is the set of the nucleotide positions of the planted motif and P is the set of the extrapolated positions by the algorithm. This ratio reflects the number of correctly predicted motif locations. The performance coefficient is calculated using the function `printPerformanceCoefficient` in the program `motiffinder.cpp`.

Gene	Link to the gene sequence	Gene location	Promoter location	Link to the promoter region
ADE8	[44]	Chromosome: IV; NC_001136.10 (1288215..1288859, complement)	1288860:1289459	[45]
AGP1	[46]	Chromosome: III; NC_001135.5 (76018..77919, complement)	77920:79434	[47]
ARG4	[48]	Chromosome: VIII; NC_001140.6 (140011..141402, complement)	141403:142568	[49]
ATR1	[50]	Chromosome: XIII; NC_001145.3 (38196..39824)	39825:41519	[51]
GAP1	[52]	Chromosome: XI; NC_001143.9 (515063..516871)	516872:517766	[53]
HIS4	[54]	Chromosome: III; NC_001135.5 (65934..68333, complement)	68334:69625	[55]
HIS5	[56]	Chromosome: IX; NC_001141.2 (142928..144085)	144086:145200	[57]
LEU3	[58]	Chromosome: XII; NC_001144.5 (1036093..1038753)	1038754:1039649	[59]

Table 2: A table of the genes ADE8, AGP1, ARG4, ATR1, GAP1, HIS4, HIS5, and LEU3, a link to each gene, gene location, promoter location and Github link to the promoter region.

Gene	Link to the gene sequence	Gene location	Promoter location	Link to the promoter region
LPD1	[60]	Chromosome: VI NC_001138.5 (101628..103127, complement)	103128:104434	[61]
LYS14	[62]	Chromosome: IV; NC_001136.10 (509737..512109, complement)	512110:513219	[63]
MET28	[64]	Chromosome: IX; NC_001141.2 (383556..384119, complement)	384120:385314	[65]
MTD1	[66]	Chromosome: XI; NC_001143.9 (590395..591357)	591358:592468	[67]
SHM2	[68]	Chromosome: XII; NC_001144.5 (257992..259401, complement)	259402:260615	[69]
TPK1	[70]	Chromosome: X; NC_001142.9 (109966..111159, complement)	111160:112243	[71]
TPK2	[72]	Chromosome: XVI; NC_001148.4 (166256..167398)	165656:166255	[73]
URE2	[74]	Chromosome: XIV; NC_001146.8 (219137..220201, complement)	220202:220801	[75]

Table 3: A table of the genes LPD1, LYS14, MET28, MTD1, SHM2, TPK1, TPK2, and URE2, a link to each gene, gene location, promoter location and Github link to the promoter region.

3 Results

3.1 Running the programs

The comparison of the performance of the two algorithms can be done by running the programm `motiffinder.cpp` for an (l,d)-Problem in the respective folder (either for GenMap or for Projection) in the format of:

For synthetically generated sequences with a synthetically generated motif (see each command in Appendix B):

```
./motiffinder ../datasets/l-d/syn_synthetic_data_l_d.fasta
../datasets/l-d/parameters_l_d.csv
../datasets/l-d/syn_planted_motif_l_d.csv 10
```

For the biological sequences:

```
./motiffinder ../datasets/gcn4_biological/GCN4_promoter_regions
../datasets/gcn4_biological/parameters_l_d.csv
```

For synthetically generated sequences with an experimentally verified but artificially mutated motif:

```
./motiffinder ../datasets/gcn4_synthetic/gcn4_synthetic_sequences
.fasta ../datasets/gcn4_synthetic/parameters_l_d.csv
../datasets/gcn4_synthetic/gcn4_implanted_motifs.csv 10
```

Where l is the motif length and d the maximum number of mutations in it.

The programs were run on a virtual machine with Centos 7 64bit, 16 Cores @ 2Ghz and 32Gb RAM.

3.2 Tests on synthetic sequences with a synthetic motif

3.2.1 Parameter selection

For the GenMap algorithm, at least 20% of the sequences had to contain the hidden motif. For Projection, the projection length k was set to 7, the threshold s to 4 and the number of trials m to the same values as in the tests performed by the algorithm's creators (see Appendix A). Different (l, d) - problems were considered whereby the effects of altering the l and d parameters were explored. 7 of the the problem instances that were unlikely to contain spurious motifs were considered for the experiments (Table 4). That property is determined by their E value (Equation (2)) described earlier. The problems likely to produce spurious motifs are listed in Table 5 and one of those problems (the (9,2)-problem) was used for testing.

3.2.2 Performance comparison

Detection accuracy

The average performance coefficient (a.p.c) over 10 datasets is presented in Table 4 and 5. In the same tables, the number of executed trials m is shown along with the number of correctly found consensus sequences. The execution of cases in Table 4, which exhibited

l	d	E(l,d)	a.p.c Projection	a.p.c GenMap	number of trials
10	2	$6.11 \cdot 10^{-8}$	1	1	72
11	2	$5.43 \cdot 10^{-17}$	1	1	16
12	3	$3.19 \cdot 10^{-7}$	0.7	0.8	259
13	3	$8.14 \cdot 10^{-16}$	1	1	62
14	4	$4.20 \cdot 10^{-7}$	0.43	0.22	647
15	4	$2.17 \cdot 10^{-15}$	1	1	172
16	5	$2.33 \cdot 10^{-7}$	-	-	1292
17	5	$2.00 \cdot 10^{-17}$	1	-	378
18	6	-	-	-	2217
19	6	-	-	-	711

Table 4: Statistics for tractable (l,d)-problems, where l is the motif length, d is the maximum number of possible mutations in a motif occurrence, E(l,d) is the probability that the motif occurs by chance and a.p.c. is the average performance coefficient.

l	d	E(l,d)	a.p.c. Projection	a.p.c. GenMap	number of trials
9	2	1.59	0.12	0.14	1483
11	3	4.72	-	-	-
13	4	5.23	-	-	-
15	5	2.84	-	-	-
17	6	0.89	-	-	-

Table 5: Statistics for intractable (l,d)-problems, where l is the motif length, d is the maximum number of possible mutations in a motif occurrence, E(l,d) is the probability that the motif occurs by chance and a.p.c. is the average performance coefficient.

more than four mismatches, was infeasible using the GenMap software due to technical limitations. The Projection algorithm, on the other hand, consumed an extensive amount of computational resources. Similar difficulties were encountered in Table 5 while attempting to apply the Projection algorithm. Since no Projection results were available for comparison, it was decided not to run the cases in question using GenMap.

Computational speed

The computational speed of the genmap-program and two versions of the projection-program (with and without multithreading) was measured and displayed in Figure 11 in the form of the average time (in seconds) needed for completion of one trial for particular values of the l and d parameters and in Figure 12 for the total time. Various numbers of threads were tried out, the best result was reached with 12. Both, more and less threads, resulted in a slow down on the virtual machine. On another virtual machine with only 4 cores the optimal number of threads was 8. In each execution of the program the average time for a trial is calculated. In the end, it used to calculate the average run time for one trial over all (l,d)-motif finding problems.

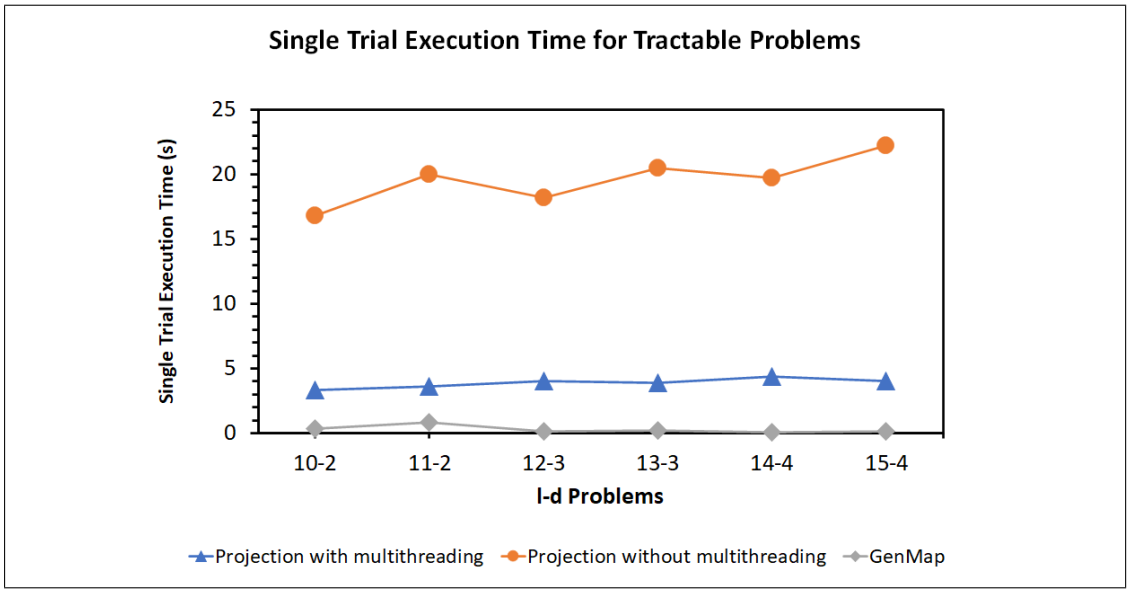

Figure 11: Line graph representing the changes in the average computational time for one trial over all 100 datasets. The (l,d) parameters are illustrated on the x-axis and the average execution time on the y-axis. The orange line shows the performance of the serial implementation of Projection, the blue line of the parallel one and the grey line of GenMap. The dashed lines show the respective linear regression slopes.
The chart was created using the Microsoft Excel Version 2303.

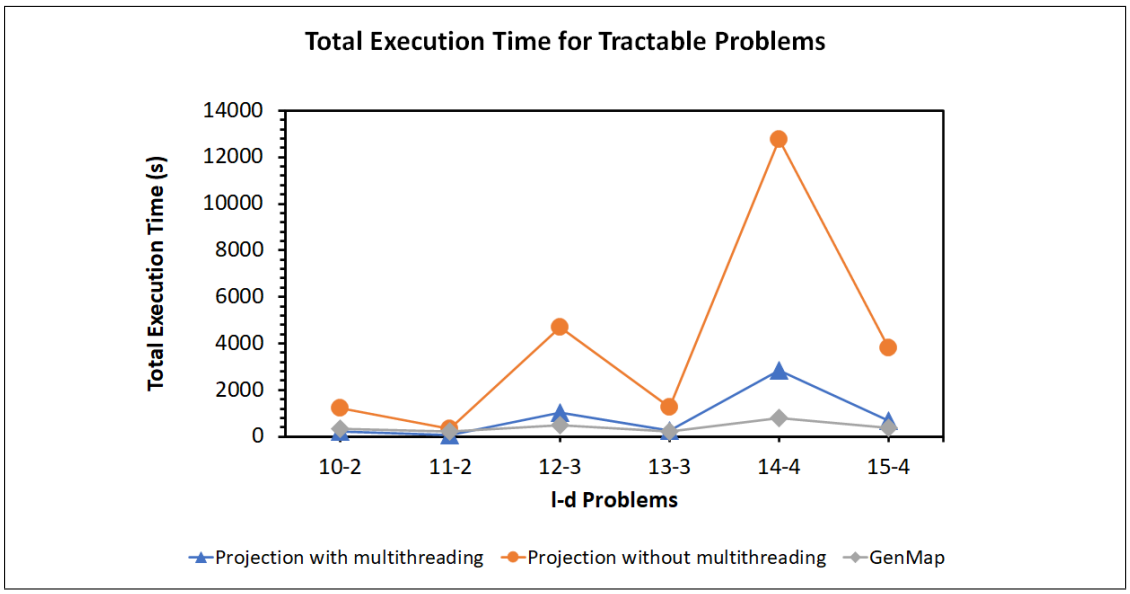

Figure 12: Line graph representing the changes in the total run timefor each problem. The (l,d) parameters are illustrated on the x-axis and the total run time on the y-axis. The orange line shows the performance of the serial implementation of Projection, the blue line of the parallel one and the grey line of GenMap. The dashed lines show the respective linear regression slopes.
The chart was created using the Microsoft Excel Version 2303.

3.3 Tests on biological data

Three algorithms, namely GenMap, Projection, and MEME [76], were utilized for the analysis of biological data. The data analyzed consisted of 16 promoter regions of length 600 nucleotides that were extracted from genes regulated by Gcn4, and were present in a single file [42].

The motif sequences obtained from the analysis are presented in Table 6, where each row corresponds to a particular problem with the length and maximum mutations of each motif. In the identified motifs, the DNA letters used to represent variability within a sequence motif are as follows: A for Adenine, C for Cytosine, G for Guanine, T for Thymine, N for any nucleotide, Y for Pyrimidine (Cytosine or Thymine), M for Amino (Adenine or Cytosine), S for Strong (Guanine or Cytosine), D for Not A (not Adenine), R for Purine (Adenine or Guanine), and W for Weak (Adenine or Thymine). The results from the MEME algorithm can be found in the Github repository [77], [78], [79].

Problem	GenMap	Projection	MEME
10-2	AAAAAATGAA	AAAAAATGAA	TTTTTTTTYT
11-2	TTTTTTTTTCA	AATTTTTTTTT	MTTTTTTTYYT
12-3	AATTTTTTTTTC	ATTTTTTTTTAT	TTTTTTTTYYG

Table 6: Results from the motif finders GenMap, Projection and MEME. The letter M represents A or C and Y represents C or T.

Additionally, another test was conducted with the MEME algorithm, which identified six MEME motifs with an acceptable length range between 8 and 15 nucleotides [80] illustrated in Table 7.

MEME motif
ARMAAAAAARRAAAA
AAAAAGAGCANAGCA
TTTTTTTC
CTGTGCTG
YTGSCDGAGTCACYA
WTGACTCR

Table 7: Results MEME with allowed motif length between 8 and 15 nucleotides, where R stands for A or G, M for A or C, N for any nucleotide, Y for C or T, W for A or T, R for A or G, D for not A, and S for G or C.

The commands used for MEME are available in the respective Github results file under COMMAND LINE SUMMARY.

3.4 Tests on synthetic sequences with a motif generated based on experimentally verified data

The program `implantmotif.cpp` generated 10 datasets [81] which were then given as input to the `genmap`- and `projection`-motiffinders. The parameters were chosen on the basis of prior knowledge about the consensus sequence of the transcription regulator Gcn4 (see Section 2.10). Table 8 shows the average performance coefficients of the two programs

for different l-d Problems. However, some cases did not include the specified number of Projection trials in the research article authored by Buhler and Tompa. For the remaining cases, the performance limitations described in Section 3.2.2 hindered the generation of results using the Projection method.

Problem	a.p.c. GenMap	a.p.c. Projection
7-1	0.41	-
7-2	0.04	-
8-1	0.49	-
8-2	0.29	-
8-3	0.15	-
9-1	0.38	-
9-2	0.58	0.58
9-3	0.24	-
9-4	0.24	-
10-1	0.27	-
10-2	0.51	0.53
10-3	0.52	-
11-1	0.09	-
11-2	0.48	0.45
11-3	0.58	-
11-4	0.2	-
12-3	0.34	0.35
13-3	0.19	0.13

Table 8: Average performance coefficient (a.p.c.) for the motif finders GenMap and Projection which were ran on random sequences with implanted motifs generated based on experimentally validated biological data.

4 Discussion

4.1 Performance Assessment of the Tests on Synthetic Data

4.1.1 Detection Accuracy

The programs performed well in detecting the planted motifs in most of the experiments, including the (10,2)-, (11,2)-, (13,3)-, and (15,4)-problems, with an optimal average performance coefficient (a.p.c.) for both motiffinders. These results are consistent with those reported in the original paper, except for the (17,5)-problem, where the a.p.c. was slightly higher for the `projection`-motiffinder in this study.

However, the (12,3)- and (14,4)-problems had significantly lower a.p.c. values for both motif finders, which is consistent with the results reported by Buhler and Tompa in the original paper. Additionally, the (9,2)-problem had a very poor performance with an a.p.c. of only 0.12, likely due to the high probability of containing spurious motifs. This value was twice as high in the original paper, but the number of correct motifs was similar in both studies.

The differences between the results of this study and the original paper could be attributed to the smaller number of datasets used for statistics in this study (10 vs. 100 in

the original paper).

4.1.2 Computational Speeds

The parallelized `projection`-motiffinder generally outperformed the version without multithreading for each problem size, while the `genmap`-motiffinder consistently had the lowest execution time for one trial in all problem sizes. The analysis showed that parallelization led to a significant improvement in runtime, ranging from 4- to 8-fold across the (l,d)-problems. The original experiments of Buhler and Tompa were carried out on a much less powerful workstation, and the authors reported that easier problems, such as (15,4), were typically solved in only a few minutes, which is consistent with the runtime expense of the implemented program with parallel computing.

4.2 Performance Assessment of the Tests on Biological Data

The `projection`- and `genmap`-programs were not successful in inferring consensus sequences of the reported motif for Gcn4, and the program MEME also experienced challenges. However, on the sixth iteration, MEME generated a sequence ("WTGACTCR") that closely resembled the one obtained from the JASPAR database ("NNTGASTCANN"). The most appropriate alignment could be illustrated as:

```
 WTGACTCR
 ||||||||
NNTGASTCANN
```

Most of the discovered motifs were rich in "A" and "T" which leads to the possibility that the promoter regions were rich in these two bases.

4.3 Performance Assessment of Synthetic Sequences with a Motif Generated Based on Experimentally Verified Data

For all problems except (13,3), `genmap` outperformed `projection` in terms of a.p.c. Specifically, for problems (9,2), (10,2), and (11,2), `genmap` and `projection` had similar a.p.c. values, but for larger problems ((12,3) and (13,3)), `genmap` significantly outperformed `projection`. The experiments performed solely with `genmap` showed notable results for the (8,1)-, (10,3)-, and (11,3)-problems, which is not surprising as the consensus sequence of Gcn4 ("NNTGASTCANN") contains seven highly conserved positions among its 11 nucleotides.

5 Conclusion

5.1 Findings

On the whole, the results suggest that GenMap is a promising algorithm for solving problems of this nature. Both programs performed well on synthetic data by succeeding to correctly solve some of the easier motif finding problems but could not identify the consensus sequence of an experimentally verified transcription factor. By making use of C++ optimization strategies the `projection`-program executed with the desired run time expense, it was however still significantly outperfomed by the `genmap`-program.

5.2 Limitations and weaknesses

GenMap has the potential to improve by allowing more than 4 mutations and accounting for the background single nucleotide frequencies. Additionally, the performance assessment was conducted using a limited number of datasets (10 for each (l,d)-problem), which may not provide sufficient statistical robustness. To obtain more reliable results, a larger number of tests should be performed. Moreover, the programs were designed to generate a consensus sequence, which does not fully capture the information regarding the conservation of individual positions within the motif. Furthermore, performance issues prevented the execution of certain cases using the Projection method, rendering the generation of results impractical in those instances.

5.3 Future work

The motif discovery models explored in this thesis are based on the assumption that the genes in question are coregulated and therefore requires a reliable method for identifying them. A technique called phylogenetic footprinting could be used for this purpose. It works by aligning the orthologous sequences in different species and then by looking for conserved regions and analysing them.

Another aspect that should be considered is observing the cases in which the motif occurs zero or more than one times per sequence. Models with those properties were already proposed by the authors of the MEME algorithm: the zero or one occurrence per sequence (ZOOPS) and the two-component mixture (TCM) models.

Bibliography

[1] Patke et al. Mutation of the human circadian clock gene cry1 in familial delayed sleep phase disorder. *Cell*, 169(2), April 2017.

[2] Faizeh Al-Quobaili and Mathias Montenarh. Pancreatic duodenal homeobox factor-1 and diabetes mellitus type 2 (review). In *INTERNATIONAL JOURNAL OF MOLECULAR MEDICINE 21: 399-404*, 2008.

[3] Hans J. J. van der Vliet and Edward E. Nieuwenhuis. IPEX as a result of mutations in FOXP3. *Clinical and Developmental Immunology*, 2007:1–5, 2007.

[4] Institute of Advanced Studies at the University of Surrey (IAS). Transcription factors as targets and markers in cancer. 2007.

[5] Moran Jerabek-Willemsen, Timon André, Randy Wanner, Heide Marie Roth, Stefan Duhr, Philipp Baaske, and Dennis Breitsprecher. MicroScale thermophoresis: Interaction analysis and beyond. *Journal of Molecular Structure*, 1077:101–113, dec 2014.

[6] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in dna sequences. 2000.

[7] A. J. Stewart, S. Hannenhalli, and J. B. Plotkin. Why transcription factor binding sites are ten nucleotides long. *Genetics*, 192(3):973–985, aug 2012.

[8] Shaul Karni Roded Sharan and Yifat Felder. Analysis of biological networks: Transcriptional networks - promoter sequence analysis. 2007.

[9] Saurabh Sinha, Martin Tompa, Department of Computer, Science, and Engineering. A statistical method for finding transcription factor binding sites. From: ISMB-00 Proceedings. Copyright Â© 2000, AAAI (www.aaai.org). All rights reserved., 2000.

[10] B. Andre J. van Helden and J. Collado-Vides. Extracting regulatory and sites from the upstream and region of yeast and genes by computational and analysis of oligonucleotide frequencies. In *J. Mol. Biol.*, 1998.

[11] Giancarlo Mauri Giulio Pavesi and Graziano Pesole. An algorithm for finding signals of unknown length in dna and sequences. In *BIOINFORMATICS*, volume 17, pages 207–214, 2001.

[12] Eleazar Eskin and Pavel A. Pevzner. Finding composite regulatory patterns in dna sequences. volume 18, pages 354–363. Bioinformatics, 2002.

[13] Timothy L Bailey George M Church Bart De Moor Eleazar Eskin Alexander V Favorov Martin C Frith Yutao Fu W James Kent Vsevolod J Makeev Andrei A Mironov William Stafford Noble Giulio Pavesi Graziano Pesole Mireille RÃ©gnier Nicolas Simonis Saurabh Sinha Gert Thijs Jacques van Helden Mathias Vandenbogaert Zhiping Weng Christopher Workman Chun Ye & Zhou Zhu Martin Tompa, Nan Li. Assessing computational tools for the discovery of transcription factor binding sites. *Nat Biotechnol*, 23:137–144, 2005.

[14] Timothy L. Bailey and Charles Elkan. The value of prior knowledge in discovering motifs with meme. In AAAI Press, editor, *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, 1990.

[15] Sriram Ramabhadran Alkes Price and Pavel A. Pevzner. Finding subtle and motifs by branching and from sample and strings. In *Bioinformatics*, volume 1, pages 1–7, 2003.

[16] lII Gerald Z. Hertz, George W.Hartzell and Gary D.Stormo. Identification of consensus patterns in unaligned dna sequences known to be functionally related. volume 6, pages 81–92, 1990.

[17] Rahul Siddharthan, Eric D. Siggia, and Erik van Nimwegen. PhyloGibbs: A gibbs sampling motif finder that incorporates phylogeny. *PLoS Computational Biology*, 1(7):e67, 2005.

[18] Tommy Kaplan, Nir Friedman, and Hanah Margalit. Ab initio prediction of transcription factor targets using structural knowledge. *PLoS Computational Biology*, 1(1):e1, 2005.

[19] Jianjun Hu, Yifeng D Yang, and Daisuke Kihara. Emd: an ensemble algorithm for discovering regulatory motifs in dna sequences. *BMC Bioinformatics*, 7(1):342, 2006.

[20] Preston W. Estep & George M. Church Frederick P. Roth, Jason D. Hughes. Finding dna regulatory motifs within unaligned noncoding sequences clustered by whole-genome mrna quantitation. *Nature Biotechnology*, 16:939â945, October 1998.

[21] Liu JS. Liu X1, Brutlag DL. Bioprospector: discovering conserved dna motifs in upstream regulatory regions of co-expressed genes. In *Pacific Symposium on Biocomputing*, volume 6, pages 127–138, 2001.

[22] Douglas L. Brutlag & Jun S. Liu X. Shirley Liu. An algorithm for finding proteinâdna binding sites with applications to chromatin- immunoprecipitation microarray experiments. *Nature Biotechnology*, 20:835â839, 2002.

[23] Magali Lescot Stephane Rombauts Bart De Moor Pierre Rouze Gert Thijs, Kathleen Marchal and Yves Moreau. A gibbs sampling method to detect over-represented motifs in the upstream regions of co-expressed genes. In *RECOMB*, volume 5, pages 305–312, 2001.

[24] Christopher Pockrandt, Mai Alzamel, Costas S. Iliopoulos, and Knut Reinert. Genmap: Fast and exact computation of genome mappability. *bioRxiv*, 2019.

[25] Jeremy Buhler and Martin Tompa. Finding motifs and using random and projections. volume 9, pages 225–242. Mary Ann Liebert, Inc., 2002.

[26] Knut Reinert, Temesgen Hailemariam Dadi, Marcel Ehrhardt, Hannes Hauswedell, Svenja Mehringer, Ren'e Rahn, Jongkyu Kim, Christopher Pockrandt, J"org Winkler, Enrico Siragusa, Gianvito Urgese, and David Weese. The seqan c++ template library for efficient sequence analysis: A resource for programmers. *Journal of Biotechnology*, 261:157–168, November 2017.

[27] Github repository: Fast and exact motif discovery using the SeqAn library GenMap algorithm. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm. Accessed: 02.05.2023.

[28] SeqAn library. https://seqan.readthedocs.io/en/master/Tutorial/GettingStarted/BackgroundAndMotivation.html. Accessed: 26.02.2019.

[29] User Guide - SeqAn installation. https://seqan.readthedocs.io/en/master/Infrastructure/Use/Install.html. Accessed: 07.12.2022.

[30] Hui Pan, Joanna D Holbrook, Neerja Karnani, Chee-Keong Kwoh, Xueling Sim, Jianjun Liu, Woon-Puay Koh, Rob M van Dam, Kee-Seng Chia, David Y Goh, et al. Gene, environment and methylation (gem): a tool suite to efficiently navigate large scale epigenome wide association studies and integrate genotype and interaction between genotype and environment. *BMC bioinformatics*, 17(1):299, 2016.

[31] Tristan Derrien, Jordi Estellé, Santiago Marco Sola, David G Knowles, Emanuele Raineri, Roderic Guigó, and Paolo Ribeca. Fast computation and applications of genome mappability. *PLoS One*, 7(1):e30377, 2012.

[32] GenMap: How to compute the mappability on multiple fasta files or genomes? https://github.com/cpockrandt/genmap/wiki/#how-to-compute-the-mappability-on-multiple-fasta-files-or-genomes. Accessed: 29.04.2023.

[33] GenMap: How to load raw files (.map, .freq8, .freq16) in C++? https://github.com/cpockrandt/genmap/wiki/#how-to-load-raw-files-map-freq8-freq16-in-c. Accessed: 29.04.2023.

[34] GenMap: Introduction. https://github.com/cpockrandt/genmap#introduction. Accessed: 29.04.2023.

[35] Kiavash Kianfar, Christopher Pockrandt, Bahman Torkamandi, Haochen Luo, and Knut Reinert. Optimum search schemes for approximate string matching using bidirectional fm-index. 2018.

[36] Program createdatasets.cpp. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/create_datasets/src/createdatasets.cpp. Accessed: 02.05.2023.

[37] Kannan Natarajan, Matthew R Meyer, Brian M Jackson, Debra Slade, Christopher Roberts, Alan G Hinnebusch, and Matthew J Marton. Transcriptional profiling shows that gcn4p is a master regulator of gene expression during amino acid starvation in yeast. *Molecular and cellular biology*, 21(13):4347–4368, 2001.

[38] Detailed information of matrix profile MA0303.2 for GCN4. https://jaspar.genereg.net/matrix/MA0303.2/. Accessed: 02.05.2023.

[39] Thomas D.Schneider and R.Michael Stephens. Sequence logos a new way to display consensus sequences. In *Nucleic Acids Research*, volume 18, pages 6097–6100. National Cancer Institute, Frederick Cancer Research and Development Center, Laboratory of Mathematical Biology, September 1990.

[40] Sequence logo of Gcn4. https://jaspar.genereg.net/static/logos/all/svg/MA0303.2.svg. Accessed: 02.05.2023.

[41] NCBI Gene Databank. https://www.ncbi.nlm.nih.gov/gene/. Accessed: 01.05.2023.

[42] GCN4 combined file. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/genmap_motiffinder/datasets/gcn4_biological/GCN4_promoter_regions.fasta. Accessed: 02.05.2023.

[43] Program implantmotif.cpp. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/implant_motif/src/implantmotif.cpp. Accessed: 02.05.2023.

[44] ADE8. https://www.ncbi.nlm.nih.gov/gene/852017. Accessed: 01.05.2023.

[45] ADE8 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/ADE8%20promoter.fa. Accessed: 01.05.2023.

[46] AGP1. https://www.ncbi.nlm.nih.gov/gene/850333. Accessed: 01.05.2023.

[47] AGP1 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/AGP1%20promoter.fa. Accessed: 01.05.2023.

[48] ARG4. https://www.ncbi.nlm.nih.gov/gene/856411. Accessed: 01.05.2023.

[49] ARG4 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/ARG4%20promoter.fa. Accessed: 01.05.2023.

[50] ATR1. https://www.ncbi.nlm.nih.gov/gene/854924. Accessed: 01.05.2023.

[51] ATR1 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/ATR1%20promoter.fa. Accessed: 01.05.2023.

[52] GAP1. https://www.ncbi.nlm.nih.gov/gene/853912. Accessed: 01.05.2023.

[53] GAP1 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/GAP1%20promoter.fa. Accessed: 01.05.2023.

[54] HIS4. https://www.ncbi.nlm.nih.gov/gene/850327. Accessed: 01.05.2023.

[55] HIS4 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/HIS4%20promoter.fa. Accessed: 01.05.2023.

[56] HIS5. https://www.ncbi.nlm.nih.gov/gene/854690. Accessed: 01.05.2023.

[57] HIS5 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/HIS5%20promoter.fa. Accessed: 01.05.2023.

[58] LEU3. https://www.ncbi.nlm.nih.gov/gene/851172. Accessed: 01.05.2023.

[59] LEU3 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-

algorithm/blob/master/promoter_regions/GCN4_promoters/LEU3%20promoter.fa. Accessed: 01.05.2023.

[60] LPD1. https://www.ncbi.nlm.nih.gov/gene/850527. Accessed: 01.05.2023.

[61] LPD1 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/LPD1%20promoter.fa. Accessed: 01.05.2023.

[62] LYS14. https://www.ncbi.nlm.nih.gov/gene/851598. Accessed: 01.05.2023.

[63] LYS14 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/LYS14%20promoter.fa. Accessed: 01.05.2023.

[64] MET28. https://www.ncbi.nlm.nih.gov/gene/854834. Accessed: 01.05.2023.

[65] MET28 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/MET28%20promoter.fa. Accessed: 01.05.2023.

[66] MTD1. https://www.ncbi.nlm.nih.gov/gene/853955. Accessed: 01.05.2023.

[67] MTD1 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/MTD1%20promoter.fa. Accessed: 01.05.2023.

[68] SHM2. https://www.ncbi.nlm.nih.gov/gene/850747. Accessed: 01.05.2023.

[69] SMH2 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/SMH2%20promoter.fa. Accessed: 01.05.2023.

[70] TPK1. https://www.ncbi.nlm.nih.gov/gene/853275. Accessed: 01.05.2023.

[71] TPK1 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/TPK1%20promoter.fa. Accessed: 01.05.2023.

[72] TPK2. https://www.ncbi.nlm.nih.gov/gene/855898. Accessed: 01.05.2023.

[73] TPK2 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/TPK2%20promoter.fa. Accessed: 01.05.2023.

[74] URE2. https://www.ncbi.nlm.nih.gov/gene/855492. Accessed: 01.05.2023.

[75] URE2 promoter. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/promoter_regions/GCN4_promoters/URE2%20promoter.fa. Accessed: 01.05.2023.

[76] MEME Suite. http://meme-suite.org/tools/meme. Accessed: 26.02.2019.

[77] MEME Results for Gcn4 with motif length 10. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/meme_results/meme10.txt. Accessed: 06.05.2023.

[78] MEME Results for Gcn4 with motif length 11. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/meme_results/meme11.txt. Accessed: 06.05.2023.

[79] MEME Results for Gcn4 with motif length 12. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/meme_results/meme12.txt. Accessed: 06.05.2023.

[80] MEME Results for Gcn4 with motif length 8 to 15. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/blob/master/meme_results/meme8-15.txt. Accessed: 06.05.2023.

[81] Synthetic files generated by the program `implantmotif.cpp` for the motif of the transcriptional regulator Gcn4. https://github.com/GerganaStanilova/Fast-and-exact-motif-discovery-using-the-SeqAn-library-GenMap-algorithm/tree/master/genmap_motiffinder/datasets/gcn4_synthetic. Accessed: 06.05.2023.

List of Figures

1 A flowchart of the GenMap `motiffinder.cpp` . . . 11
2 A flowchart of the Projection `motiffinder.cpp` . . . 12
3 An example of a fasta-file with 3 sequences each of length 600 in which a motif could be hidden . . . 13
4 An example of a csv-file with the parameter values for l, d, k, s and m . . . 13
5 An example of a csv-file with the correct implanted motif and mutation positions 14
6 An example output of the GenMap `motiffinder.cpp` . . . 14
7 A flowchart of the implementation of the function `runGenmap` . . . 17
8 A flowchart of the implementation of the function `runProjection` . . . 18
9 A multithreading example . . . 23
10 Sequence logo of the Gcn4 transcriptional activator. . . . 25
11 Line graph representing the changes in the average computational time for one trial over all 100 datasets. . . . 31
12 Line graph representing the changes in the total run time over all problems. . . . 31
13 Average performance coefficients for planted (l,d)-motifs in synthetic data. Source: Jeremy Buhler and Martin Tompa's paper *Finding Motifs Using Random Projections*. . . . 46

List of Tables

1 The possible key arguments with their default values and description for the program generating the synthetic datasets. 24
2 A table of the genes ADE8, AGP1, ARG4, ATR1, GAP1, HIS4, HIS5, and LEU3, a link to each gene, gene location, promoter location and Github link to the promoter region. 27
3 A table of the genes LPD1, LYS14, MET28, MTD1, SHM2, TPK1, TPK2, and URE2, a link to each gene, gene location, promoter location and Github link to the promoter region. 28
4 Statistics for tractable (l,d)-problems, where l is the motif length, d is the maximum number of possible mutations in a motif occurrence, E(l,d) is the probability that the motif occurs by chance and a.p.c. is the average performance coefficient. 30
5 Statistics for intractable (l,d)-problems, where l is the motif length, d is the maximum number of possible mutations in a motif occurrence, E(l,d) is the probability that the motif occurs by chance and a.p.c. is the average performance coefficient. 30
6 Results from the motif finders GenMap, Projection and MEME. The letter M represents A or C and Y represents C or T. 32
7 Results MEME with allowed motif length between 8 and 15 nucleotides, where R stands for A or G, M for A or C, N for any nucleotide, Y for C or T, W for A or T, R for A or G, D for not A, and S for G or C. 32
8 Average performance coefficient (a.p.c.) for the motif finders GenMap and Projection which were ran on random sequences with implanted motifs generated based on experimentally validated biological data. 33

A Appendix

Results from the Buhler and Tompa's experiments on their synthetic examples.

TABLE 1. AVERAGE PERFORMANCE COEFFICIENTS FOR PLANTED (l, d)-MOTIFS IN SYNTHETIC DATA[a]

l	d	*GibbsDNA*	*WINNOWER*	*SP-STAR*	*PROJECTION*	*Correct*	m
10	2	0.20	0.78	0.56	0.80 ± 0.02	100	72
11	2	0.68	0.90	0.84	0.94 ± 0.01	100	16
12	3	0.03	0.75	0.33	0.77 ± 0.03	96	259
13	3	0.60	0.92	0.92	0.94 ± 0.01	100	62
14	4	0.02	0.02	0.20	0.71 ± 0.05	86	647
15	4	0.19	0.92	0.73	0.93 ± 0.01	100	172
16	5	0.02	0.03	0.04	0.67 ± 0.06	77	1292
17	5	0.28	0.03	0.69	0.94 ± 0.01	98	378
18	6	0.03	0.03	0.03	0.73 ± 0.06	82	2217
19	6	0.05	0.03	0.40	0.94 ± 0.01	98	711

[a]Each problem instance consists of $t = 20$ sequences each of length $n = 600$. Average performance coefficients of GibbsDNA, WINNOWER ($k = 2$), and SP-STAR are from Pevzner and Sze (personal communication), who averaged over eight random instances. For PROJECTION, we report averages and 95% confidence intervals for performance coefficient over 100 random instances with projection size $k = 7$ and threshold $s = 4$.

Figure 13: Average performance coefficients for planted (l,d)-motifs in synthetic data.
Source: Jeremy Buhler and Martin Tompa's paper *Finding Motifs Using Random Projections.*

B Appendix

A list of the command that were run for the synthetic datasets with synthetic motifs.
For the (9,2)-Problem

```
./motiffinder ../datasets/9-2/syn_synthetic_data_9_2.fasta
../datasets/9-2/parameters_9_2.csv
../datasets/9-2/syn_planted_motif_9_2.csv 10
```

For the (10,2)-Problem:

```
./motiffinder ../datasets/10-2/syn_synthetic_data_10_2.fasta
../datasets/10-2/parameters_10_2.csv
../datasets/10-2/syn_planted_motif_10_2.csv 10
```

For the (11,2)-Problem:

```
./motiffinder ../datasets/11-2/syn_synthetic_data_11_2.fasta
../datasets/11-2/parameters_11_2.csv
../datasets/11-2/syn_planted_motif_11_2.csv 10
```

For the (12,3)-Problem:

```
./motiffinder ../datasets/12-3/syn_synthetic_data_12_3.fasta
../datasets/12-3/parameters_12_3.csv
../datasets/12-3/syn_planted_motif_12_3.csv 10
```

For the (13,3)-Problem:

```
./motiffinder ../datasets/13-3/syn_synthetic_data_13_3.fasta
../datasets/13-3/parameters_13_3.csv
../datasets/13-3/syn_planted_motif_13_3.csv 10
```

For the (14,4)-Problem:

```
./motiffinder ../datasets/14-4/syn_synthetic_data_14_4.fasta
../datasets/14-4/parameters_14_4.csv
../datasets/14-4/syn_planted_motif_14_4.csv 10
```

For the (15,4)-Problem:

```
./motiffinder ../datasets/15-4/syn_synthetic_data_15_4.fasta
../datasets/15-4/parameters_15_4.csv
../datasets/15-4/syn_planted_motif_15_4.csv 10
```

For the (17,5)-Problem (only for the Projection-algorithm since GenMap does not allow for more than 4 mutations):

```
./motiffinder ../datasets/17-5/syn_synthetic_data_17_5.fasta
../datasets/17-5/parameters_17_5.csv
../datasets/17-5/syn_planted_motif_17_5.csv 10
```