Algorithms and Statistical Methods for Exact Motif Discovery

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Tobias Marschall

Dortmund

2011

Tag der mündlichen Prüfung: 25.03.2011

Dekan: Prof. Dr. Peter Buchholz

1. Gutachter: Prof. Dr. Sven Rahmann
2. Gutachter: Prof. Dr. Jens Stoye

To Julia

Abstract

The *motif discovery problem* consists of uncovering exceptional patterns (called *motifs*) in sets of sequences. It arises in molecular biology when searching for yet unknown functional sites in DNA sequences.

In this thesis, we develop a motif discovery algorithm that (1) is exact, that means it returns a motif with optimal score, (2) can use the statistical significance with respect to complex background models as a scoring function, (3) takes into account the effects of self-overlaps of motif instances, and (4) is efficient enough to be useful in large-scale applications.

To this end, several algorithms and statistical methods are developed. First, the concepts of *deterministic arithmetic automata* (DAAs) and *probabilistic arithmetic automata* (PAAs) are introduced. We prove that they allow calculating the distributions of values resulting from deterministic computations on random texts generated by arbitrary finite-memory text models. This technique is applied three times: first, to compute the distribution of the number of occurrences of a pattern in a random string, second, to compute the distribution of the number of character accesses made by window-based pattern matching algorithms, and, third, to compute the distribution of clump sizes, where a clump is a maximal set of overlapping motif occurrences. All of these applications are interesting theoretical topics in themselves and, in all three cases, our results go beyond those known previously.

In order to compute the distribution of the number of occurrences of a motif in a random text, a deterministic finite automaton (DFA) accepting the motif's instances is needed to subsequently construct a PAA. We therefore address the problem of efficiently constructing *minimal DFAs* for motif types common in computational biology. We introduce *simple non-deterministic finite automata* (NFAs) and prove that these NFAs are transformed into minimal DFAs by the classical subset construction. We show that they can be built from (sets of) generalized strings and from consensus strings with a Hamming neighborhood, allowing the direct construction of minimal DFAs for these pattern types.

As a contribution to the field of *motif statistics*, we derive a formula for the *expected clump size* of motifs. It is remarkably simple and does not involve laborious operations like matrix inversions. This formula plays an important role in developing bounds for the expected clump size of partially known motifs. Such bounds are needed to obtain bounds for the p-value of a partially known motif. Using these, we are finally able to devise a *branch-and-bound algorithm* for motif discovery that extracts provably optimal motifs with respect to their p-values in *compound Poisson* approximation. Markovian text models of arbitrary order can be used as a *background model* (or *null model*). The algorithm is further generalized to jointly handle a motif and its reverse complement. An Open Source implementation is publicly available as part of the MoSDi software

package.

An experimental evaluation using synthetic and real data sets follows. On the carefully crafted benchmark set of Sandve et al. (2007), the proposed algorithm outperforms Weeder (Bailey and Elkan, 1994) and MEME (Pavesi et al., 2004) in terms of the commonly used average nucleotide-level correlation coefficient. With respect to this measure, it is also superior to other algorithms tested by Fauteux et al. (2008) on the same benchmark suite; namely Seeder (Fauteux et al., 2008), BioProspector (Liu et al., 2001), GibbsSampler (Lawrence et al., 1993), and MotifSampler (Thijs et al., 2001).

Besides the comparison to other algorithms, we perform motif discovery on the non-coding regions of *Mycobacterium tuberculosis* and on *CpG-rich regions in the human genome*. In both cases, we report on found motifs that are strikingly over-represented. While the function of most of these motifs remains unknown to us, some motifs found in *M. tuberculosis* can be attributed to a known biological function.

Acknowledgments

First of all, I want to thank my advisor Prof. Sven Rahmann for giving me the opportunity to work with him. He gave me the freedom to pursue my own research ideas and always found time for fruitful discussions. I am deeply grateful for what I learnt from him.

Furthermore, I want to thank Prof. Jens Stoye for immediately agreeing to serve as second examiner of this thesis. I am also thankful to Prof. Joachim Biskup and Dr. Thomas Röblitz for agreeing to complete my examination commitee.

The *Center for Biotechnology* (CeBiTec) at Bielefeld University kindly provided me with computational resources, which I gratefully acknowledge.

I want to thank Katrin Rademacher for answering all my questions on CpG islands and for proof-reading parts of this thesis. I deeply thank Marcel Martin for carefully working himself through the whole thesis and making many suggestions which substantially improved the exposition. As well, I thank him for the long years in the same office, for sharing his encyclopedic knowledge of almost everything by answering so many of my questions, and for all the interesting (scientific and private) discussions.

I want to express my deep gratitude to my parents, who gave me the best possible start in life and supported me ever since.

Most of all, I want to thank Julia for all her support and love.

Tobias Marschall

Dortmund, February 2011

Contents

1 Introduction

Without sequence information encoded as deoxyribonucleic acid (DNA), none of the known living cells could exist. Understanding sequences is therefore fundamental to understanding biology. In this thesis, we study theoretical foundations and algorithms for the *motif discovery problem*: given a set of such sequences over a finite alphabet, we are asked to find exceptional patterns (called motifs). The reason for searching for exceptional patterns is the hope that they carry a biological meaning. This hope is justified by the assumption that exceptional patterns that do not play a role in a cell's function are likely to vanish in the course of evolution. Therefore, discovered patterns can serve as a basis for new biological hypotheses and direct further experimentation.

Example 1.1. A DNA molecule consists of two anti-parallel chains of nucleotides forming a double-helix. In DNA, each nucleotide contains one of the four bases **a**denine, **c**ytosine, **g**uanine, or **t**hymine and, thus, a strand of DNA can be formalized as a string over the alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. *Transcription factors* are proteins that bind to DNA in a sequence specific manner. That means they recognize special (sets of) sequences of nucleotides, called *binding motifs*. Refer to Alberts et al. (2007) for a detailed introduction to the molecular biology of cells and, in particular, to DNA, proteins, and transcription factors. The *motif discovery problem* now arises when we have experimental evidence that a set of DNA sequences contain instances of binding sites of a transcription factor but the binding motif of this factor is unknown. Then, a motif discovery algorithm might reveal a pattern that is shared by all sequences which can then be hypothesized to be the transcription factor's binding motif. In this thesis, we use the terms *pattern* and *motif* interchangeably. △

The goal of this thesis is to develop motif discovery algorithms. To do this, several questions need to be answered.

1. How should motifs be *modeled* in order to capture patterns with a biological function best?
2. How can *exceptionality* be measured?
3. How can the most exceptional motifs be *extracted*?

These questions can be attributed to the areas of biology, statistics, and algorithmics, respectively. The focus of this thesis is on the last two questions. Our aim is to develop methods that are exact, use a statistically sound definition of exceptionality, and at the same time are fast enough to be applicable in practice.

We start with an introduction to motif models in Section 1.1. That is, we discuss common answers to Question 1. In Section 1.2, we formulate the aims of this thesis informally. After introducing notation in Section 1.3, a precise formal problem statement

follows in Section 1.4. We survey existing approaches to motif discovery in Section 1.5. The chapter is concluded with an overview of this thesis given in Section 1.6.

1.1 Motif Models

The purpose of motif models is to generalize a limited number of observed motif instances to a larger set of similar strings that we predict to be motif instances as well. That means, when we encounter one of these strings in an unknown piece of DNA, we would annotate this position as a putative motif instance. Motif models differ in the way this generalization is done. The most widely used models are *consensus strings* with a Hamming neighborhood (i.e. the set of strings with a limited Hamming distance to the consensus), *position weight matrices* (PWMs) along with a score threshold, and *generalized strings* (also known as IUPAC strings in the context of DNA). In the following, we explain the three models using the example of a DNA binding site, namely record `MA0107.1` in the Jaspar database (Sandelin et al., 2004) which represents the binding site of the *Rel* protein domain as described by Kunsch et al. (1992). The database contains 18 experimentally verified binding sites that are shown in Figure 1.1a. Some of these sites occur multiple times such that the set of all sites contains 14 distinct sequences.

Consensus Strings with Hamming Neighborhood. When confronted with a set of motif instances, a *consensus* string can be obtained by majority vote at each position as depicted in Figure 1.1b. In the shown case, the consensus `GGGAATTTCC` has a Hamming distance of up to three to the original sequences. Therefore, the motif might be modeled as the set of all strings with a Hamming distance of at most three to the consensus. Thus, the given 18 strings are generalized to all 3676 strings with at most three differences to `GGGAATTTCC`. The underlying assumption is that only the number of mismatches is important and not their position. Whether or not this assumption is justified depends on the application. In case of transcription factor binding sites, some positions are usually more conserved than others, challenging this assumption. Nonetheless, the Weeder algorithm by Pavesi et al. (2004), which uses this motif model, outperformed many other methods in a benchmark study by Tompa et al. (2005).

Position Weight Matrices. A *position weight matrix* (PWM) of length ℓ is a $|\Sigma| \times \ell$ matrix, where each entry $w_{\sigma i}$ gives the score to be counted when character σ is found at position i (Staden, 1984). Given a putative motif instance, the scores for the characters at all positions are summed up to obtain a score for the instance. When the score is above a pre-chosen threshold, the string is said to be a motif instance. A common method to obtain a PWM is the translation of a position frequency matrix (PFM) into log-odds scores, where the PFM contains, for each position, the number of times each character occurred at this position in the sample sequences. An example is shown in Figure 1.1d. Formally, the PWM entries are defined by $w_{\sigma i} := \log(r_{\sigma i}/p_\sigma)$, where $r_{\sigma i}$ is the observed relative frequency of character σ at position i and p_σ is the background probability of σ (i.e., the assumed probability of σ in non-motif sequences).

(a)

```
GGGAATTTCC
GGGAATTTCC
GGGAATTTCC
TGGAATTTCC
TGGAATTTCC
CGGAATTTCC
GGGGATTTCC
GGGACTTTCC
CGGAGTTTCC
GGGAATTCCC
CGGACTTTCC
CGGACTTTCC
GGGGAATTCC
TGGGGTTTCC
GGGGATTCCC
GGGGTTTTCC
GGGGGATTCC
GTGGGTTTCC
```

(b) Consensus: `GGGAATTTCC`

(c) IUPAC string: `BKGRNWTYCC`

(d)

A:	0	0	0	11	10	2	0	0	0	0
C:	4	0	0	0	3	0	0	2	18	18
G:	11	17	18	7	4	0	0	0	0	0
T:	3	1	0	0	1	16	18	16	0	0

(e)

A:	-2.30	-2.30	-2.30	0.83	0.74	-0.69	-2.30	-2.30	-2.30	-2.30
C:	-0.11	-2.30	-2.30	-2.30	-0.36	-2.30	-2.30	-0.69	1.31	1.31
G:	0.83	1.25	1.31	0.41	-0.11	-2.30	-2.30	-2.30	-2.30	-2.30
T:	-0.36	-1.20	-2.30	-2.30	-1.20	1.19	1.31	1.19	-2.30	-2.30

(f)

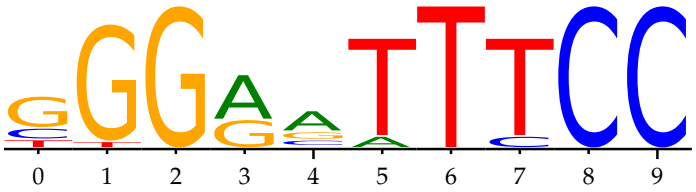


Figure 1.1: Different motif models of the binding site `MA0107.1` from the Jaspar database are shown: (a) original sequences (b) consensus string obtained by position-wise majority vote (c) minimal IUPAC string matching all sequences (d) position frequency matrix (e) position weight matrix containing log-odds scores computed assuming uniform background distribution and 0.5 pseudocounts (f) sequence logo.

When computing the relative frequencies $r_{\sigma i}$, pseudocounts should be used to avoid singularities: $r_{\sigma i} := (\gamma + f_{\sigma i})/(|\Sigma|\gamma + \sum_{\sigma'} f_{\sigma' i})$, where the $f_{\sigma i}$ are absolute frequencies and γ is a small pseudocount constant. Refer to Rahmann et al. (2003) for a detailed discussion of pseudocounts. For the PWM shown in Figure 1.1e, the largest score threshold such that all sample sequences obtain a score equal to or larger than this threshold is 7.55. There exist 75 strings with a score above this threshold. Thus, the motif model is less *degenerate*, that is, more specific, than that obtained in the previous paragraph using a consensus string. PWMs are commonly visualized by *sequence logos* as exemplified in Figure 1.1f.

Remark 1.2 (PWM score reflects binding affinity)**.** For a PWM derived from the binding sites of a transcription factor, the PWM score of a sequence can be interpreted in terms of the binding affinity of the transcription factor to this sequence. Roider et al. (2007) use a biophysical model to derive a monotonic function mapping the PWM score to the binding probability. Depending on the application, this probability can be used directly rather than imposing a score threshold to make a binary decision whether or not a string is a motif instance. △

Generalized Strings. A *generalized string* is a sequence of character sets. Each set gives the allowed characters at that position. The generalized string $g = \{\mathtt{A}, \mathtt{G}\}\{\mathtt{A}\}\{\mathtt{A}, \mathtt{T}\}$, for example, matches AAA, AAT, GAA, and GAT. Sets containing more than one letter (like $\{\mathtt{A}, \mathtt{G}\}$ or $\{\mathtt{A}, \mathtt{T}\}$) are called *wildcards*. In case of nucleotide sequences, all possible sets are commonly abbreviated by IUPAC one-letter codes (IUPAC stands for *International Union of Pure and Applied Chemistry*, the one-letter codes for sets of DNA characters have been proposed by Cornish-Bowden, 1985). Using these codes, g is written RAW. Appendix A.2 contains a table of all IUPAC codes. Therefore, generalized strings over the nucleotide alphabet are sometimes called *IUPAC strings*. The minimal IUPAC string matching all sample sequences shown in Figure 1.1a is BKGRNWTYCC. It matches a total of 192 strings. Thus, it is more specific than the consensus string with Hamming neighborhood and less specific than the PWM model.

Limits of flexibility of each of these three types of motif models can be judged in terms of the number of different expressible motifs. For length ℓ, there are 4^ℓ different consensus strings and 15^ℓ different IUPAC strings. Every string set expressible as a consensus string with its Hamming neighborhood or a IUPAC string can also be expressed by a PWM with a threshold. Additionally, PWMs allow us to express string sets that cannot be described using the former two models. Therefore, PWMs are most flexible. In all data mining tasks and in particular in motif discovery, however, higher model flexibility comes with an increased risk of over-fitting. Or, speaking from a statistician's perspective, when a larger motif space is explored, more hypotheses are tested, leading to a higher number of false positive results.

All three types of models do not infer dependencies between different positions within the sample sequences. However, in a set of observed sites, two (either neighboring or distant) positions might be correlated. For example, the characters observed at position i might vary but at the same time equal the character at position j in all sample sequence. If such dependencies can reliably be detected, then they allow a more informed generalization. To this end, Zhang and Marr (1993) propose an extended matrix model incorporating dependencies between adjacent positions. The method by Agarwal and Bafna (1998) identifies, for each position, the most influential other positions and models their dependency. An even more general approach using Bayesian networks is given by Barash et al. (2003). Other approaches include mixtures of PWMs (King and Roth, 2003) and models based on dense subgraphs in a k-mer graph (Fratkin et al., 2006). All these methods have in common that they introduce larger motif spaces compared to approaches that do not model position dependencies.

In this thesis, motifs are assessed according to their statistical significance. For two models that capture the same motif instances in a set of sample sequences, the more specific one usually leads to a higher significance. On the other hand, too flexible models lead to many false predictions due to multiple testing. Therefore, the type of motif model we choose should be neither too inflexible nor have too many free parameters. Although most ideas presented in this work also apply to other motif models, we focus on *generalized strings* which we consider to be a suitable compromise between these demands. This choice is further justified in Chapter 6, where this model

is shown to yield good results in applications.

1.2 Aims

In this section, we identify desired properties of the motif discovery algorithm to be developed in this thesis. Before doing that, we discuss the user input to be made to such an algorithm.

Every motif discovery algorithm should require that the user provides three pieces of information; first, a set of input sequences to be searched, second, some hypothesis on the kind of pattern to be looked for, and third, knowledge on sequence composition, that is, a specification of sequence features that are to be expected and should hence *not* be reported as new motifs. While it is clear that input sequences must be provided, the second and third aspect are often not recognized as input to be made by the user. Both are, however, fundamental to motif discovery. To exemplify that, let us assume that no assumptions on pattern type are made. Then, in an extreme case, an algorithm might report the whole input text itself as a long motif with one occurrence and a high significance score. Testing a larger space of motifs is therefore not always an advantage, even when runtimes are not considered. A motif space should therefore consciously be chosen by the user based on the scientific question to be addressed. When searching for transcription factor binding sites, for instance, we can use knowledge on known binding motifs to specify a suitable range of motif lengths to be considered and possibly a maximal allowed degree of degeneracy, that is, a limit on the use of wildcard characters. The third requirement, providing information on expected sequence composition, can be seen as answering the question of what is already known about the studied sequences. For example, if nothing is known, it is a new result that a genome contains overrepresented patterns comprised of many `C`s and `G`s. If, in contrast, a biologist knows that the `GC` content (i.e. the fraction of nucleotides that are `C` or `G`) is high, these news are hardly surprising. As the goal of automated motif discovery is to find novel sequence features, it is inevitable that information about what is already known needs to be available. Therefore, the user should provide a *background model* or *null model*.

This analysis of input to be made by the user directly translates into the first two requirements on a motif discovery algorithm listed below. Additionally, we formulate two further requirements.

Flexibility of background models. Background text models should be general enough to describe complex dependencies. I.i.d. models and first order Markovian models are insufficient for many applications, especially those concerning DNA. Text models are discussed in detail in Sections 1.4 and 2.3.

Appropriate scoring functions. The scoring function used to assess motifs should reflect the statistical significance with respect to the background model. The statistical properties of motifs can strongly be affected by self-overlaps (see Example 1.8). Hence, overlapping must be accounted for.

Exactness. To eliminate the risk of missing significant motifs, the algorithm should not work heuristically but extract a motif that is provably optimal with respect to the scoring function.

Practicality. The algorithm should be able to solve practical problem instances on current hardware in reasonable time.

The development of mathematical and algorithmical means to achieve this is the main goal of the present thesis. Before we can formalize the problem, we need to introduce some notation.

1.3 Notation and Conventions

Random variables are represented by capital Latin letters (in normal typeface), for example, X might be the number of occurrences of a motif in a random text and S_t might be the character at position t in this random text. Sets are also named by capital Latin letters, either in normal or script typeface; script is used, for instance, when disambiguation from names of random variables is needed. For example, $\mathcal{Q}$ might be a set of states and $(Q_i)_{i\in\mathbb{N}}$ a sequence of random variables taking values in $\mathcal{Q}$. Power sets are also commonly typeset in script; we might, for instance, write $\mathcal{K}$ to denote 2^K, the power set of K. For elements of sets, the same letter as for the set is used, but in lower case; e.g. a state $q \in \mathcal{Q}$.

Due to the limited supply of different letters, some have to be reused in the course of this thesis. It will be clear from the context which variable is meant. The meaning of the most important symbols and variables, however, is the same throughout the whole text. A list can be found on pages 149ff. Important symbols include the following: Σ is a finite alphabet, Σ^* denotes the set of all finite strings; ε is the empty string; $\mathcal{S} \subset \Sigma^*$ is a finite set of texts; $N := \sum_{s\in\mathcal{S}} |s|$ is the total text length; ℓ is the pattern length. By $\mathbb{P}$, we refer to a probability measure and $\mathcal{L}(X)$ denotes the *distribution* or *law* of the random variable X. It will always be clear from the context which probability measure is referred to. All stochastic processes considered in this thesis are discrete. Therefore, appropriate probability spaces can always be constructed; we do not clutter notation by stating them explicitly. The set of all values that can be taken by a random variable X, that is, the image of X, is denoted by $\mathrm{range}(X)$. The convolution of two probability distributions φ and ψ, written $\varphi * \psi$, is defined by $(\varphi * \psi)(k) := \sum_{i=0}^{k} \varphi(i) \cdot \psi(k-i)$ for all $k \in \mathbb{N}_0$. The j-fold convolution of φ with itself is denoted by φ^{*j}. Iverson brackets are written $[\![\cdot]\!]$, i.e. $[\![A]\!] = 1$ if the statement A is true and $[\![A]\!] = 0$ otherwise. The sign function is written *sgn*. Multisets are written $\{\{1, 2, 2, 7\}\}$. All indices are zero-based, i.e. $s = s[0] \dots s[|s| - 1]$. Substrings are written $s[i \dots j] := s[i] \dots s[j]$, where $s[i \dots j] := \varepsilon$ for $j < i$. Furthermore, prefix, suffix, and reverse are written $s[..i] := s[0 \dots i]$, $s[i..] := s[i \dots |s| - 1]$, and $\overleftarrow{s} := s[|s| - 1] \dots s[0]$, respectively. $\mathbb{N}$ is the set of natural numbers (excluding zero) and $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. The set of real numbers is denoted by $\mathbb{R}$, while $\mathbb{R}^+$ and $\mathbb{R}_0^+$ refer to the real numbers greater than zero and

greater than or equal to zero, respectively. $\mathbb{R}^{n\times m}$ is the set of $n \times m$ matrices over $\mathbb{R}$; the zero matrix is written $\mathbf{0}$, and the identity matrix is denoted by $\mathbf{1}$. We adopt the bra-ket notation, also known as Dirac notation, for vectors. That means that a row vector is written $\langle x|$ and its transposed is written $|x\rangle$. Then, using the standard inner product, we have $\langle x| \cdot |y\rangle = \langle x|y\rangle$. Furthermore, $|0\rangle \in \mathbb{R}^m$ and $|1\rangle \in \mathbb{R}^m$ refer to the vectors whose components all equal zero and one, respectively.

Experiments

Unless stated otherwise, experiments have been performed on an Intel Core 2 Quad CPU at 2.66 GHz with 8 GB RAM running Linux. Although many algorithms discussed in this thesis are parallelizable, the reported times refer to a single core. Algorithms have been implemented in Java. Implementations are available as part of the MoSDi software package at

`http://mosdi.googlecode.com/`.

It is distributed under the terms of the *GNU General Public License* (GPL).

Human Genome

We use Build 36.1 of the human genome as provided by the National Center for Biotechnology Information (NCBI). This assembly is also known as *HG18*.

1.4 Problem Formalization

Based on the goals given in Section 1.2, we now formally state the problems to be solved. As an objective function, we use the *statistical significance* which is also called *p-value*. In order to introduce it formally, we need definitions of *random texts* and *text models*.

Definition 1.3 (Random text)**.** A *random text* is a stochastic process $(S_t)_{t\in\mathbb{N}_0}$, where each S_t takes values in a finite alphabet Σ. Then, the concatenation of $S_0, \ldots, S_{N-1}$, written $S_0 \cdots S_{N-1}$, is called *random text of length* N. Furthermore, we define $S_i^\ell := S_i \cdots S_{i+\ell-1}$, that means S_i^ℓ refers to the substring of length ℓ starting at position i. ◊

Definition 1.4 (Text model)**.** A *text model* $\mathbb{P} : 2^{(\Sigma^{\mathbb{N}})} \to [0, 1]$ is a probability measure assigning probabilities to sets of infinitely long strings. ◊

Therefore, each random variable S_i is a function from $\Sigma^{\mathbb{N}}$ to Σ returning the i-th character of the given infinite string. As probability measures are countably additive, this definition of text models implies that all probabilities are consistent. In particular, $\mathbb{P}\big(S_0^{|s|} = s\big) \geq \mathbb{P}\big(S_0^{|s|+1} = s\sigma\big)$ for all $s \in \Sigma^*$ and $\sigma \in \Sigma$.

According to Kolmogorov's existence theorem (see, for instance, Billingsley, 1995), consistently specifying the probabilities $\mathbb{P}(S_0 \cdots S_{|s|-1} = s)$ for all $s \in \Sigma^*$ ensures the existence of a corresponding text model.

Example 1.5 (Uniform Text Model)**.** The *uniform text model* is obtained by setting

$$\mathbb{P}_{\mathit{uniform}}\big(S_0 \cdots S_{|s|-1} = s\big) := \frac{1}{|\Sigma|^{|s|}}$$

for all $s \in \Sigma^*$. △

Example 1.6 (I.i.d. Text Model)**.** Let a probability p_σ for each character $\sigma \in \Sigma$ be given. Furthermore, let all S_i be independent and identically distributed (i.i.d.). Then, we get an *i.i.d. text model* with

$$\mathbb{P}_{\mathit{iid}}\big(S_0 \cdots S_{|s|-1} = s\big) := \prod_{i=0}^{|s|-1} p_{s[i]}$$

for all $s \in \Sigma^*$. △

In this thesis, text models are used as background models. That is, they encode what we know or assume *a priori* about the composition of the input sequence(s). We might, for instance, use an i.i.d. model with $p_{\mathtt{A}} = p_{\mathtt{T}} = 0.6$ and $p_{\mathtt{C}} = p_{\mathtt{G}} = 0.4$ to reflect the nucleotide composition of the human genome. As we see in Example 1.8, however, the use of more elaborate models is advisable for DNA. Therefore, we discuss general *finite-memory text models* in Section 2.3.

Definition 1.7 (p-value for the total number of occurrences in a single string)**.** Let a motif p, an input text $s \in \Sigma^*$, and a text model $\mathbb{P}$ be given. The *p-value of p in s* with respect to the text model $\mathbb{P}$ is given by

$$\mathit{pvalue}\,(p, s) := \mathbb{P}\big(\mathit{occ}_p\left(S_0 \cdots S_{|s|-1}\right) \;\geq\; \mathit{occ}_p\,(s)\big)\,,$$

where $\mathit{occ}_p\,(s)$ is the number of occurrences of p in s. ◊

That means, when a motif occurs k times in a given text, the p-value is the probability that it occurs k or more times in a random text of the same length.

Example 1.8. Suppose we have analyzed a portion of human DNA of length 10 000 and have found the motifs shown in Table 1.2. They differ in frequency, character composition, and overlap potential. The p-values allow a comparison despite these differences. Regarding the overlap potential, for example, note that the first two motifs `AAAAATTTTT` and `ATATATATAT` have different p-values, even with respect to a uniform text model, although the number of occurrences and character composition are the same. The first one cannot overlap itself, while the second has a quite strong tendency to do so. This leads to an increased probability of observing many instances in a random text. When using an i.i.d. model reflecting the character composition in the human genome instead of a uniform model, the p-values for the `AT`-rich motifs increase, while they decrease for the `CG`-rich motifs. According to a second order Markovian model, that is, a model where the character distribution depends on the two preceding characters (see Section 2.3), the motif with the lowest p-value is `CGCGNNCGCG`. The information that the dinucleotide `CG` is largely avoided in the human

Table 1.2: Assuming that the shown IUPAC motifs (left column) have been observed in a stretch of human DNA of length 10 000, their p-values with respect to different text models are given, namely (from left to right) a uniform text model, an i.i.d. text model, and a second order Markovian text model. The latter two models have been estimated from the human genome. Refer to Example 1.8 for more details.

Motif	Occurrences	p-value (uniform)	p-value (i.i.d.)	p-value (Markovian)
AAAAATTTTT	6	$1.0 \cdot 10^{-15}$	$2.2 \cdot 10^{-11}$	$6.8 \cdot 10^{-07}$
ATATATATAT	6	$1.2 \cdot 10^{-08}$	$6.0 \cdot 10^{-07}$	$3.1 \cdot 10^{-07}$
ANANNNTTNT	50	$6.6 \cdot 10^{-19}$	$7.7 \cdot 10^{-07}$	$1.1 \cdot 10^{-02}$
CGCGNNCGCG	5	$2.1 \cdot 10^{-06}$	$9.1 \cdot 10^{-09}$	$1.1 \cdot 10^{-13}$
GGCCNNGGCC	5	$9.2 \cdot 10^{-07}$	$5.5 \cdot 10^{-10}$	$1.7 \cdot 10^{-07}$

genome is encoded in the text model and, thus, finding CGCGNNCGCG five times is rightly considered surprising. This example shows that the p-value is a powerful tool to compare the significance of motifs. It also shows that an appropriate text model must be used to make relevant findings. △

Instead of searching a single string s for motifs, a finite set of strings $\mathcal{S}$ might be given as input. In this case, we use a set of random texts to define the p-value.

Definition 1.9 (p-value for the total number of occurrences in a set of strings)**.** Let a motif p, a finite set of input strings $\mathcal{S} = \{s_0, \ldots, s_k\}$, and a text model $\mathbb{P}$ be given. The *p-value of p in $\mathcal{S}$* with respect to the text model $\mathbb{P}$ is given by

$$\mathit{pvalue}\,(p, \mathcal{S}) := \mathbb{P}\Big(\sum_{i=0}^{k} \mathit{occ}_p\left(S_{i,0} \cdots S_{i,|s_i|-1}\right) \;\geq\; \sum_{i=0}^{k} \mathit{occ}_p\left(s_i\right) \Big)\,,$$

where $(S_{i,t})_{t \in \mathbb{N}_0}$ are random texts distributed according to $\mathbb{P}$ for $i \in \{0, \ldots, k\}$. ◊

Furthermore, we can consider not only the total number of occurrences, but also the number of strings a pattern occurs in.

Definition 1.10 (Number of matching strings)**.** Given an finite set of strings $\mathcal{S} \subset \Sigma^*$ and a motif p. The *number of matching strings* is given by

$$\mathit{occ\text{-}seqs}_p\left(\mathcal{S}\right) := \sum_{s \in \mathcal{S}} [\![\mathit{occ}_p\left(s\right) > 0]\!]\,.$$

◊

Definition 1.11 (p-value for the number of matching strings)**.** Let a motif p, a finite set of input strings $\mathcal{S} = \{s_0, \ldots, s_k\}$, and a text model $\mathbb{P}$ be given. The *p-value of p in $\mathcal{S}$* with respect to the text model $\mathbb{P}$ is given by

$$\mathit{pvalue}_{\mathit{seqs}}(p, \mathcal{S}) := \mathbb{P}\Big(\sum_{i=0}^{k} [\![(\mathit{occ}_p(S_{i,0} \cdots S_{i,|s_i|-1}) > 0]\!] \geq \mathit{occ\text{-}seqs}_p(\mathcal{S}) \Big),$$

where $(S_{i,t})_{t \in \mathbb{N}_0}$ are random texts distributed according to $\mathbb{P}$ for $i \in \{0, \ldots, k\}$. ◊

The above definitions are applicable for any kind of motif model. In this thesis, we model motifs as generalized strings, that is, as sequences of sets of characters.

Definition 1.12 (Generalized string)**.** Given an alphabet Σ, we call the set $\mathcal{G}_\Sigma := 2^\Sigma \setminus \{\emptyset\}$ *generalized alphabet over* Σ and a string $g \in \mathcal{G}_\Sigma^*$ *generalized string*. We say that a string $s \in \Sigma^*$ and a generalized string $g \in \mathcal{G}_\Sigma^*$ *match*, written $s \lhd g$, if $|s| = |g|$ and $s[i] \in g[i]$ for $0 \leq i < |g|$. ◊

We write $\mathcal{G}$ instead of $\mathcal{G}_\Sigma$ if the used alphabet is clear from the context. Every string $s \in \Sigma$ can be translated into the generalized string $\{s[0]\}\{s[2]\} \ldots \{s[|s|-1]\}$. In this sense, strings can be seen as special cases of generalized strings.

Now we can state those two variants of the motif discovery problem whose solution is the main goal of this thesis.

Problem 1 (Discovery of generalized string w.r.t. the total number of occurrences)**.** Given a finite set of input sequences $\mathcal{S} \subset \Sigma^*$, a text model $\mathbb{P}$, and a pattern length ℓ, compute a motif $p \in \mathcal{G}^\ell$ with minimal p-value according to Definition 1.9. That means, choose p such that there does not exist a $p' \in \mathcal{G}^\ell$ with $\mathit{pvalue}(p', \mathcal{S}) < \mathit{pvalue}(p, \mathcal{S})$.

Problem 2 (Discovery of generalized string w.r.t. the number of matching strings)**.** Given a finite set of input sequences $\mathcal{S} \subset \Sigma^*$, a text model $\mathbb{P}$, and a pattern length ℓ, compute a motif $p \in \mathcal{G}^\ell$ with minimal p-value according to Definition 1.11. That means, choose p such that there does not exist a $p' \in \mathcal{G}^\ell$ with $\mathit{pvalue}_{\mathit{seqs}}(p', \mathcal{S}) < \mathit{pvalue}_{\mathit{seqs}}(p, \mathcal{S})$.

A subproblem that needs to be solved first is to compute the p-value for a given motif with respect to a given text model.

1.5 Related Work

On the one hand, motif discovery is a hard task. This can be exemplified by the fact that even simple formalizations of the problems are NP-hard; respective references to hardness results are given in Section 1.5.1. On the other hand, the problem needs to be solved in order to aid the understanding of biological sequences. This dichotomy might explain why the topic has received an enormous amount of attention. More than 400 articles on motif discovery algorithms have been published. Most of these algorithms work *heuristically*. That means they do not guarantee to find a motif that maximizes the chosen objective function. We call motif discovery algorithms that do

make this guarantee *exact*. This thesis aims at the development of exact algorithms. For the sake of completeness, we briefly explain the most successful strategies for heuristic motif discovery in Section 1.5.2 before surveying the literature on exact algorithms in Section 1.5.3.

1.5.1 Complexity Results

One formalization of motif discovery is the *Closest Substring Problem*: Given a pattern length ℓ and a finite set $\mathcal{S}$ of length-n sequences over a finite alphabet Σ, find a minimal distance threshold d and a string $p \in \Sigma^\ell$ such that each $s \in \mathcal{S}$ has a length-ℓ substring with a Hamming distance to p of at most d. This problem was shown to be NP-hard by Lanctot et al. (1999). An algorithm by Ma and Sun (2009) solves it in $\mathcal{O}\big((16|\Sigma|)^d \cdot |\mathcal{S}| \cdot n^{\lceil \log d \rceil + 2}\big)$ time. Solving the Closest Substring Problem yields motifs represented as a consensus string with a Hamming neighborhood. Searching for the best IUPAC string can similarly be formalized when minimizing the motif's degeneracy instead of the parameter d, where the degeneracy of a IUPAC string is defined as the number of strings it matches. Thus, the string `ANNC` has a degeneracy of 16. Now the problem is formalized as follows. Given a set $\mathcal{S} \subset \Sigma^n$ and a pattern length ℓ, find a length-ℓ IUPAC string p with minimal degeneracy such that all $s \in \mathcal{S}$ contain a substring that matches p. Precisely this problem has been encountered in the context of PCR primer design and was shown to be NP-hard, too (Linhart, 2002; Linhart and Shamir, 2005).

To be useful in practical applications, a motif discovery problem should be formulated as an optimization problem where the objective function takes a background model into account. The effect of self-overlaps of motifs should be paid attention to as well. Both requirements are further discussed in Sections 1.2 and 1.4. The focus of this thesis is not on formal hardness proofs, but it can be said that, in general, considering background models and overlaps does not make problems easier. Therefore, the subject of this thesis are hard problems which we cannot hope to find polynomial time algorithms for.

1.5.2 Heuristic Algorithms

The advantage of heuristic algorithms lies in their speed. In general, however, their use implies the risk of missing motifs, especially when the signal is weak. Therefore, trying a heuristic algorithm is advisable whenever short running times are important and strong signals are expected in the input sequences.

Expectation Maximization

The *Expectation-Maximization* (EM) method allows estimating parameters of probabilistic models with *hidden variables* by iteratively improving the model's *likelihood*. That means, given observed data, the algorithm tries to find a model that explains the data best. The EM principle has first been generically formulated by Dempster et al. (1977). One important application is the estimation of mixture models; that is, models

composed of several simpler models called *components*. The assumption is that a datum is drawn by choosing one of the components with fixed yet unknown probability and generating the datum according to this component. Here, the hidden variables tell which datum was generated by which component. Starting from a randomly (or informedly) chosen set of model parameters, new parameters are computed in each iteration. One EM step consists of calculating new model parameters that maximize the expected (log-)likelihood, where the expectation is taken over the hidden variables with respect to the current model. By using the expectation with respect to the current, and therefore known, model parameters, the maximization is usually computationally inexpensive. One key feature of this EM update strategy is that the likelihood cannot decrease and, thus, converges to a local optimum.

The first EM-based motif discovery algorithm we are aware of was published by Lawrence and Reilly (1990). Based on their work, Bailey and Elkan (1994) developed the widely known and used MEME algorithm. Given a set of sequences and a length ℓ, MEME views the input sequences as a sequence of (overlapping) ℓ-grams generated by a mixture model consisting of two components. The first component models the background, that means, those parts of the sequences not part of the motif. The parameters for this component are the probabilities of each character in the alphabet, that is, all letters are assumed to be independent and identically distributed (i.i.d.). The second component models the motif by specifying a separate character distribution for each position from 1 to ℓ. The information which ℓ-gram was generated by which model, that is, the positions of motif occurrences, is missing and seen as a hidden variable in the EM framework. The parameter estimation is then run until convergence using the EM algorithm.

Recent improvements of EM algorithms for motif discovery include the combination with Monte Carlo sampling (Jackson and Fitzgerald, 2007; Bi, 2009), the integration of false discovery rate control (Li et al., 2008), and the parallelized implementation on graphics processing units (Chen et al., 2008).

Gibbs Sampling

Gibbs sampling produces a series of samples from a multidimensional probability space by fixing all dimensions but one in each step and sampling the non-fixed dimension conditionally on the fixed ones. This is often much easier than sampling all dimensions at once from the joint distribution. The application of this paradigm to motif discovery has been proposed by Lawrence et al. (1993). In this case, the multidimensional data consists of the positions of motif occurrences in each input sequence. It is assumed that each sequence contains exactly one occurrence and random positions are chosen initially. In each iteration, one of the occurrence positions is resampled. To do this, all other occurrences are fixed and a PWM is constructed out of them. The new position is now sampled such that each window's sample probability is proportional to its probability as given by the PWM. That means the sum of PWM scores for all windows is normalized to one to give the sample distribution.

The initial contribution of Lawrence et al. (1993) has inspired many authors to work on improvements and various implementations. A similar algorithm, for instance, has

been implemented in the AlignACE program (Roth et al., 1998; Hughes et al., 2000). An adaptation of the idea to gapped alignments was given by Rocke and Tompa (1998). As further elaborated by Rocke (2000), the performance of this algorithm can be improved by using suffix trees. The BioProspector program by Liu et al. (2001) is able to handle Markovian background models and motifs consisting of two blocks separated by a gap. Thijs et al. (2002) discuss the use of Markovian background models as well. Shida (2006) proposes to modify the constructed PWM in each step such that strongly conserved columns are slightly weakened. It is argued that this is useful when the positions of differences of motif instances are distributed uniformly. In a study by Reddy et al. (2007), the robustness against the presence of sequences with no motif occurrence is found to increase when multiple runs are performed and only those positions found most often are retained. The GIMSAN software by Ng and Keich (2008) combines Gibbs sampling with motif significance assessment based on Gamma distributions whose parameters are fitted by running the Gibbs sampling algorithm on random sequences. Defrance and van Helden (2009) modify the sampling such that it explicitly targets the PWM's information content; that means the sampling distribution is chosen such that selecting positions that increase the information content is encouraged.

Random Projections

EM and Gibbs sampling methods have in common that they iteratively improve a candidate motif and therefore can get stuck in a local optimum. In fact, EM cannot even theoretically leave a local optimum. Thus, the attained motif strongly depends on the initial (usually random) choice of a candidate motif to be refined. To make an informed initialization, Buhler and Tompa (2002) put forward a *random projection* strategy for choosing good candidates. When searching for motif candidates with width ℓ, a number of $k < \ell$ column indices is chosen randomly. The selected column indices resemble a spaced seed (see Ma et al., 2002). Then, each ℓ-mer in the input sequences is translated into a k-mer by retaining the chosen columns and discarding the rest. This can be seen as a hash function mapping every ℓ-mer into one of 4^k buckets. For each bucket, the number of ℓ-mers that hash into it is determined. Each bucket for which this number exceeds a threshold is further investigated by using the ℓ-mers in it as an initial motif to be refined by local search. For details on choosing k and a threshold appropriately, refer to Buhler and Tompa (2002). Then, the steps of choosing a projection, that means, selecting columns, determining which buckets exceed the threshold, and running local search on them can be iterated and the overall best motif be reported. Regarding this procedure, Raphael et al. (2004) have noted that the strategy of uniformly sampling from the space of all projections can imply long waiting times until the *right* subset of columns is contained in the projection. They propose a modified sampling strategy that favors projections containing subsets not contained in the projections considered so far.

In general, random projections can often be used to speed up algorithms based on ℓ-mers. In the motif discovery algorithm given by Chin and Leung (2005), for example, each ℓ-mer in the input makes votes for all strings in its Hamming neighborhood. This approach is costly as it requires to enumerate these neighborhoods, but it becomes

feasible for larger values of ℓ when using random projections.

Other Methods

Many other data mining techniques have been applied to the problem, including neural networks (e.g. Heumann et al., 1994; Workman and Stormo, 2000; Liu et al., 2006), evolutionary algorithms (e.g. Fogel et al., 2004; Rahmann et al., 2009), particle swarm optimization (Chang et al., 2004), greedy algorithms (e.g. Stormo and Hartzell, 1989; Hertz and Stormo, 1999), and support vector machines (Schultheiss et al., 2009). We do not go into the details of all these heuristics but continue with a survey of exact methods.

1.5.3 Exact Algorithms

Exact motif discovery algorithms guarantee to return a motif with maximal score.

Analysis of k-mer composition

One approach is to analyze the frequency of all k-mers in the input sequences (for a reasonable value of k). As we discuss below, there are many different ways to count k-mers and to decide whether or not the obtained k-mer distribution indicates the presence of an overrepresented motif.

An early contribution was made by Waterman et al. (1984). Their idea is to simultaneously slide a length-ℓ window over all given sequences and examine the windows' k-mer compositions (for a k smaller than ℓ). Thus, their algorithm requires the input sequences to be (at least roughly) aligned and to have equal lengths. Although this simple approach suffices to detect signals with a fixed distance to some known sequence features such as transcription start sites (Galas et al., 1985), we are more interested in methods that do not require any kind of prior alignment.

To this end, Staden (1989) has proposed to analyze the ℓ-mer composition of the complete input sequences by constructing a data structure that he terms *fuzzy dictionary*. It contains, for each ℓ-mer, the number of its occurrences plus the number of occurrences of similar ℓ-mers; i.e., those ℓ-mers within a Hamming neighborhood. All ℓ-mers for which a similar one occurs more often are discarded. Of the retained ones, those with occurrence counts above a threshold are further analyzed by computing the information content of the derived PWM. This strategy does not require the input sequences to be aligned, but is limited to small pattern lengths. Furthermore, the method only guarantees to find motifs with a high number of occurrences; it does not guarantee to discover one with maximal information content.

Several other methods based on the enumeration of all ℓ-mers have been developed. Given a set of input sequences, Pesole et al. (1992) compare the number of sequences each ℓ-mer occurs in to the expectation with respect to a first order Markov model and compute a χ^2 statistic. The number of occurrences per sequence is assumed to be Poisson distributed. In the end, all patterns with a χ^2 value above a threshold are considered significant. Rather than approximating it, the probability that a sequence

contains one or more occurrences of an ℓ-mer with at most one mismatch is exactly computed by Tompa (1999) using finite automata. Based on that, exact z-scores, that is, deviations from expectation divided by standard deviations, are obtained directly. Shortly after that, Sinha and Tompa (2000) generalized this approach to allow limited use of IUPAC wildcard characters like a number of `N`s as spacers in the middle of motifs. Furthermore, their improved method takes all occurrences into account, including multiple occurrences in the same sequence, and computes exact z-scores. These are two of few prior articles on motif discovery that take the effects of self-overlaps into account. These algorithms are implemented in the YMF software (Sinha and Tompa, 2002, 2003).

To analyze yeast promoters, van Helden et al. (1998) also enumerate all ℓ-mers (for $\ell = 6$). They assume all length-ℓ windows to be independent and thus neglect overlaps. The significance of the total number of occurrences of each ℓ-mer is then computed by using a binomial distribution as a null model. This is approximately the same as using a Poisson distribution as the binomial distribution converges to the Poisson distribution when the number of trials goes to infinity (see Section 2.2). To correct for multiple testing, the p-value is then multiplied by the number of tested ℓ-mers. As van Helden et al. (2000) point out, this procedure can at once be adapted to the detection of pairs of short ℓ-mers separated by a fixed-width spacer. Another enumerative algorithm is proposed by Sze and Zhao (2006); it handles arbitrarily placed "don't care" characters (equivalent to IUPAC symbol `N`) and mismatches at the same time. Motif candidates are judged according to an E-value (refer to Sze and Zhao, 2006, for the precise definition). Although mismatches and don't care characters further the number of possible overlaps, the motif's overlapping structure is ignored when computing E-values.

Irredundant Motifs

When considering motifs of symbols from the input alphabet Σ plus a "don't care" character, a *maximal motif* is a motif that cannot be made more specific without losing occurrences. Research has been carried out on *irredundant motifs* which are maximal motifs that cannot be decomposed into multiple other maximal motifs. That means there does not exist a set of other maximal motifs that covers precisely the same sequence positions. Parida et al. (2000) seemingly proved that the cardinality of the set of irredundant motifs grows at most linearly with the length of the input sequence, but Pisanti et al. (2005) disproved this result by giving a counterexample. Even when a set of irredundant motifs has been extracted, for example by using the algorithm of Apostolico and Tagliacollo (2008), the problem of efficiently combining the resulting motifs in order to obtain the most statistically significant ones is, to our knowledge, unsolved.

Clustering Algorithms

Pevzner and Sze (2000) point out that motif discovery can be viewed as a clustering problem. They propose an algorithm called Winnower that builds a graph of all

length-ℓ windows in a set of input sequences. Two vertices are connected when the windows come from different sequences and their Hamming distance is below a threshold. Then, motifs are discovered by searching for cliques. Jensen et al. (2006) formulate the procedure generically: Given input sequences, a distance measure, and a length ℓ, compute the distance of all pairs of ℓ-mers in the input sequences and subject the resulting matrix to a clustering algorithm. On the one hand, this approach has the advantage of permitting the use of arbitrary distance measures and clustering paradigms. On the other hand, long inputs render this approach infeasible as the size of the matrix of distances grows quadratically with the input length. A related approach is taken by Fratkin et al. (2006) who cast motif discovery into a *maximum density subgraph problem* on a graph of ℓ-mers. Zaslavsky and Singh (2006) use a weighted graph where the edge weights are given by ℓ-mer similarity. Then, they apply *integer linear programming* to find a maximum weight clique with exactly one ℓ-mer from each input sequence.

Walking Suffix Trees

Among the first motif discovery algorithms using suffix trees were those of Brazma et al. (1998) and Sagot (1998). The former authors propose to construct a tree of putative motifs represented by IUPAC strings. That means that the tree is not built over the DNA alphabet, but over the IUPAC alphabet or a subset thereof. It is built lazily as described by Giegerich and Kurtz (1995), that is, a set of occurrence positions is stored within each node and only the needed nodes are actually constructed. Brazma et al. (1998) give several heuristic criteria to decide whether parts of the tree cannot contain interesting motifs and can therefore be omitted. They do not create a node, for instance, if the number of corresponding occurrences is below ten.

Instead of creating a tree of all patterns, Sagot (1998) uses the suffix tree of the input sequences and traverses it while searching for motifs. Despite this difference, the key technique is the same: for each visited node, it is decided whether the subtree below it can safely be skipped without missing relevant motifs. Or, speaking in terms of the string representing the motif, given a prefix, it is determined whether all motifs with this prefix can be skipped. Besides looking for exactly repeated patterns, the algorithm by Sagot (1998) is able to discover consensus strings. The input to the algorithm is a sequence, the number of allowed errors, and a quorum. All strings that occur (possibly overlapping) at least as often as the quorum with at most the given number of mismatches are returned by the algorithm. Allowing mismatches is achieved by branching in the suffix tree; instead of one node, a set of nodes can be active. The algorithms developed in this thesis use similar techniques which are described in more detail in Section 5.1. The main problem of Sagot's algorithm is the need to specify one global quorum. In general, the occurrence count is a bad measure of over-representation; it neither takes a background model nor a motif's degeneracy into account. Using a fixed quorum thus inevitably leads to a detection bias towards degenerate motifs and produces false positive hits that can often be explained by an appropriate background model.

Similar techniques to discover motifs while traversing a suffix tree are used in other

articles. Eskin and Pevzner (2002), for instance, combine the idea with the graph-based approach by Pevzner and Sze (2000). For each examined pattern prefix, it is determined whether the corresponding vertices in a (virtual) ℓ-mer graph can form a clique. If not, the pattern prefix is discarded. A speed-up of the initial algorithm by Sagot (1998) is reported by Pisanti et al. (2006). They propose to avoid duplicate computations: when it has been detected that a motif p cannot be extended to a certain length while meeting the quorum, this information is stored and reused when motifs having p as a substring are examined. The Weeder algorithm introduced by Pavesi et al. (2001) obtains a speed-up by imposing constraints on the placement of mismatches. A string is only considered to be a motif instance if, in all its prefixes, only a user-specified fraction of characters mismatch. This forbids, for example, to have all mismatches at the beginning, reducing the average size of the set of "active" suffix tree nodes during the search. Marsan and Sagot (2000) generalize the original algorithm to detect *structured motifs* which consist of several conserved blocks separated by spacers of different, possibly variable, lengths. A speed-up of this algorithm by augmenting the suffix tree with so called *box links* is reported by Carvalho et al. (2006). Although significance of found motifs is assessed by a χ^2 test, a fixed quorum is again used for initial motif extraction. Another approach to detect structured motifs is proposed by Federico et al. (2009). They outline an algorithm to combine several single motifs into structured motifs.

Based on prior work of Eskin (2004), it is shown by Leung and Chin (2005) that a branch-and-bound strategy on suffix trees can be used to extract a PWM with optimal likelihood. The idea is to partition the space of all theoretically possible PWM columns; then, a sequence of partitions corresponds to a set of PWMs. In other words, an alphabet is defined where each character is an infinite set of PWM columns. The partitioning is done such that, for each "character", the information content can be bounded. Now, the algorithm searches for sequences over that alphabet, i.e., sets of PWMs, that can potentially contain the optimal PWM. Large parts of the sequence tree can be pruned by using the bounds. The set of candidates is narrowed by iteratively refining the partitions.

Another way of using suffix trees to detect overrepresented words is described by Apostolico et al. (2000). They give an efficient algorithm to annotate each suffix tree node with expectation and variance of the corresponding string. This allows the computation of z-scores for all words represented by a suffix tree node. Moreover, the authors observe that z-scores are monotone in the sense that whenever appending a letter keeps the number of occurrences constant, the z-score increases. This implies that the words with maximal z-score must be explicitly represented by nodes in the suffix tree. In another article, Apostolico et al. (2003) further elaborate on such monotonicity properties of the z- and other scores. Nonetheless, the method has the drawback that it cannot handle mismatches or wildcard characters.

Conclusion

In summary, some existing algorithms meet some of the demands discussed in Section 1.2, but no algorithm fulfills all these requirements.

1.6 Organization of the Thesis

Before approaching Problems 1 and 2, we develop necessary mathematical and algorithmic techniques. In Chapter 2, fundamental concepts are introduced. This includes classical ones like (non)deterministic finite automata and hidden Markov models, but also new ones like *probabilistic arithmetic automata* (PAAs) to whose development the author has made contributions. The formal definition of PAAs and its application to pattern matching statistics have been previously published as

> Marschall and Rahmann (2008). *Probabilistic Arithmetic Automata and their Application to Pattern Matching Statistics*. Proceedings of CPM.

As an illustrative example on the utility of PAAs, Chapter 2 furthermore contains a digression on the analysis of pattern matching algorithms that is based on

> Marschall and Rahmann (2010a). *Exact Analysis of Horspool's and Sunday's Pattern Matching Algorithms with Probabilistic Arithmetic Automata*. Proceedings of LATA. An extended version has been submitted. A preprint is available from arXiv (Marschall and Rahmann, 2010c).

Further examples for applications of PAAs that are not included in this thesis can be found in

> Marschall, Herms, Kaltenbach, and Rahmann (submitted). *Probabilistic Arithmetic Automata and their Applications*. A preprint is available from arXiv (Marschall et al., 2010).

One result of Chapter 2 is that a motif's significance can be computed based on a deterministic finite automaton (DFA) accepting its instances. Algorithms for the efficient construction of minimal DFAs are therefore helpful and presented in Chapter 3 which is based on

> Marschall (2011). *Construction of Minimal Deterministic Finite Automata from Biological Motifs*. Theoretical Computer Science.

Clumps are maximal sets of overlapping occurrences of motifs. Hence, overlapping of motifs can be studied by studying clumps. When the distribution of clump sizes, that is, the number of occurrences in a clump, is known for a motif, motif statistics can be calculated based on fast and accurate compound Poisson approximations. Chapter 4 is thus devoted to clump statistics. Results on expected clump sizes have previously been published as

> Marschall and Rahmann (2010b). *Speeding up Exact Motif Discovery by Bounding the Expected Clump Size*. Proceedings of WABI.

Chapter 5 follows, where techniques from previous chapters are used to devise a motif discovery algorithm. The central idea of using a branch-and-bound approach to optimize the p-value is part of an article that has appeared as

> Marschall and Rahmann (2009). *Efficient Exact Motif Discovery*. Proceedings of ISMB.

The developed algorithms are applied to several sets of synthetic and biological sequences in Chapter 6. The final Chapter 7 contains a concluding discussion and an outlook to future work.

2 Preliminaries

In this chapter, we gather basic definitions and needed methodology. We begin with the classical definitions of automata in Section 2.1. In Section 2.2, we remind the reader of some probability distributions important in this thesis, namely binomial, Poisson, and compound Poisson distributions. Finite-memory text models are defined in Section 2.3. Useful extensions to classical automata are *probabilistic arithmetic automata* (PAAs) and *deterministic arithmetic automata* (DAA), which we define in Sections 2.4 and 2.5, respectively. As detailed in Section 2.6, these concepts allow us to connect deterministic finite automata (DFAs) accepting the instances of a motif to a finite-memory text model in order to obtain the distribution of the number of occurrences and compute the motif's statistical significance. Moreover, PAAs can be used in many applications beyond motif statistics. To get a flavor of what can be done using this technique, we analyze pattern matching algorithms in Section 2.7.

2.1 Finite Automata

In this section, we give the classical definitions of automata.

Definition 2.1 (Deterministic finite automaton (DFA))**.** A *deterministic finite automaton* is a tuple $(\mathcal{Q}, \Sigma, \delta, q_0, F)$, where $\mathcal{Q}$ is a finite set of states, Σ is a finite alphabet, $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is a *transition function*, $q_0 \in \mathcal{Q}$ is the *start state*, and $F \subset \mathcal{Q}$ is the set of *accepting states*. ◊

Definition 2.2 (Non-deterministic finite automaton (NFA))**.** A *non-deterministic finite automaton* is a tuple $(\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$, where $\mathcal{Q}$, Σ and F are defined as for the DFA above, $\Delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$ is the *non-deterministic transition function*, and $\mathcal{Q}_0 \subset \mathcal{Q}$ is a *set of start states*. ◊

Using a set of start states $\mathcal{Q}_0$ instead of only one start state is a notational convenience rather than a conceptual change: we can always transform the automaton to have only one start state by adding a new start state q_0 and defining its outgoing transitions by

$$(q_0, \sigma) \mapsto \bigcup_{q \in \mathcal{Q}_0} \Delta(q, \sigma)\,.$$

Another convenience is the extension of a DFA's transition function to strings (instead of single characters):

$$\hat{\delta} : \mathcal{Q} \times \Sigma^* \to \mathcal{Q} \tag{2.1}$$

$$(q, s) \mapsto \begin{cases} q & \text{if } s = \varepsilon\,, \\ \hat{\delta}\big(\delta(q, s[0]), s[1..]\big) & \text{otherwise}\,. \end{cases} \tag{2.2}$$

Note that $\delta(q, \sigma) = \hat{\delta}(q, \sigma)$ for all $q \in \mathcal{Q}$ and $\sigma \in \Sigma$. Therefore distinguishing between δ and $\hat{\delta}$ unnecessarily clutters notation. Hence, we omit the distinction and write δ to also refer to the extended function that maps pairs from $\mathcal{Q} \times \Sigma^*$ to states in $\mathcal{Q}$. Analogously, the transition function Δ of an NFA is extended in its second argument.

Sometimes it is useful to replace the set F with counts for each state.

Definition 2.3 (Counting DFA)**.** Let a DFA $(\mathcal{Q}, \Sigma, \delta, q_0, F)$ and values $\eta_q \in \mathbb{N}_0$ for each $q \in \mathcal{Q}$ be given. Then, the tuple $\big(\mathcal{Q}, \Sigma, \delta, q_0, (\eta_q)_{q \in \mathcal{Q}}\big)$ is called *counting DFA*. ◊

Example 2.4 (Aho-Corasick automaton as counting DFA)**.** For a given finite set of strings $\mathcal{S} \subset \Sigma^*$, an Aho-Corasick automaton (Aho and Corasick, 1975) can find all occurrences of strings from $\mathcal{S}$ in a given text. As $\mathcal{S}$ may contain strings that are suffixes of other strings in $\mathcal{S}$, it is possible that the occurrences of two different strings end at the same position. This is reflected in the Aho-Corasick automaton by an *output set* attached to each state. The automaton is constructed such that, when entering a state, we know that the strings in its output set end at the present position in the read text. An Aho-Corasick automaton can now be seen as a counting DFA over the same state space. We define the count η_q as the size of the output set of the state q. Then, the number of occurrence can be obtained by letting the counting DFA read a text and adding up the counts η_q of all visited states. △

Definition 2.5 (Language of a state)**.** The *language of an NFA state* $q \in \mathcal{Q}$ is given by

$$L_{NFA}(q) := \left\{ s \in \Sigma^* \,\middle|\, \Delta(q, s) \cap F \neq \emptyset \right\}.$$

Analogously, the *language of a DFA state* $q \in \mathcal{Q}$ is given by

$$L_{DFA}(q) := \left\{ s \in \Sigma^* \,\middle|\, \delta(q, s) \in F \right\}.$$

The language of a set of states $\mathcal{Q}' \subset \mathcal{Q}$ is defined as $L_{NFA}(\mathcal{Q}') := \bigcup_{q' \in \mathcal{Q}'} L_{NFA}(q')$ and $L_{DFA}(\mathcal{Q}') := \bigcup_{q' \in \mathcal{Q}'} L_{DFA}(q')$, respectively. ◊

Definition 2.6 (Language of a finite automaton)**.** The *language accepted by an NFA* $(\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ is defined as $L_{NFA}(\mathcal{Q}_0)$. The *language accepted by a DFA* $(\mathcal{Q}, \Sigma, \delta, q_0, F)$ is defined as $L_{DFA}(q_0)$. ◊

Let us briefly review the classical textbook construction of a DFA accepting the same language as a given NFA.

Lemma 2.7 (Subset construction; Rabin and Scott, 1959)**.** *Let* $M = (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ *be an NFA. Then,*

$$\left(2^{\mathcal{Q}},\ \Sigma,\ \delta,\ \mathcal{Q}_0,\ \left\{ \mathcal{Q}' \in 2^{\mathcal{Q}} \,\middle|\, \mathcal{Q}' \cap F \neq \emptyset \right\}\right)$$

with $\delta : (\mathcal{Q}', \sigma) \mapsto \bigcup_{q' \in \mathcal{Q}'} \Delta(q', \sigma)$*, is a DFA that accepts the same language as* M*.*

Proof. Omitted. See Rabin and Scott (1959) or Kozen (1999). □

Remark 2.8 (Languages of states in subset DFA)**.** Let $(\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ be an NFA. Applying the above subset construction yields a DFA whose states are elements of $2^{\mathcal{Q}}$ and thus sets of NFA states. As can be verified inductively, the subset construction ensures that $L_{DFA}(q) = L_{NFA}(q)$ for all $q \in 2^{\mathcal{Q}}$. △

When it is clear from the context whether L_{NFA} or L_{DFA} is meant or the distinction is unnecessary according to Remark 2.8, we omit the subscript.

Definition 2.9 (State accessibility)**.** Let an NFA $M = (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ be given. Formally, a state $q \in \mathcal{Q}$ is called *accessible* if there exists a string $s \in \Sigma^*$ and a start state $q_0 \in \mathcal{Q}_0$ such that $\Delta(q_0, s) = q$. Likewise, accessibility is defined for DFAs. ◊

The DFA that results from applying the subset construction to an NFA can have inaccessible states. These states can be removed from the DFA's state space without changing the accepted language. To ease notation, we write SUBSET-CONSTRUCTION(M) to denote the DFA resulting from the subset construction and subsequent removal of inaccessible states. In practice, we can use an algorithm that only generates the accessible states by performing a breadth-first search on the state space (see Navarro and Raffinot, 2002).

Two DFA states can be *equivalent*, meaning that their language is the same. Formally, two states q and q' of a DFA $(\mathcal{Q}, \Sigma, \delta, q_0, F)$ are *equivalent* if

$$\delta(q, s) \in F \quad \Longleftrightarrow \quad \delta(q', s) \in F$$

for all $s \in \Sigma^*$. This notion of equivalence can be used to characterize minimal DFAs, where a DFA is called minimal if there does not exist a DFA with fewer states accepting the same language.

Lemma 2.10. *A DFA is minimal if and only if its states are pairwise non-equivalent.*

Proof. See Chapters 13 and 15 in the textbook by Kozen (1999). □

In Chapter 3, we see how minimal DFAs can be constructed for different types of motifs.

2.2 Binomial, Poisson, and Compound Poisson Distributions

The *binomial distribution* gives the probabilities for the number of successes in a fixed number of trials assuming that all trials are independent and have equal success probabilities. If, for example, a coin is flipped a fixed number of times the number of observed heads follows a binomial distribution. Formally, it is given by

$$\mathcal{B}_{n,p} : k \mapsto \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k},$$

where n gives the number of trials and p the success probability.

When the number of trials grows to infinity but, at the same time, the expected number of events λ is kept fixed, we obtain a *Poisson distribution*. For example, the number of received phone calls in a given time interval is Poisson distributed (assuming that all callers choose the time of calling independently and uniformly). We can, in principle, receive a call at any point in time and thus the observed event has a continuum of opportunities to occur. Formally,

$$\mathcal{P}_\lambda(k) = \lim_{n \to \infty} \mathcal{B}_{n,\lambda/n}(k) \tag{2.3}$$

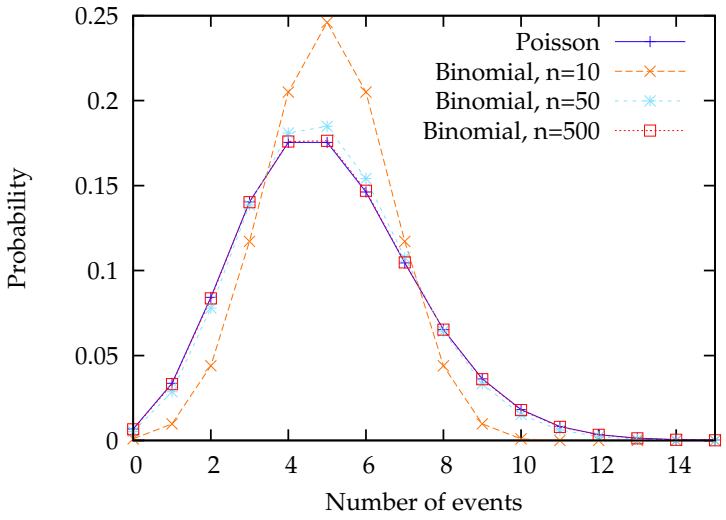


Figure 2.1: Comparison of different distributions with an expectation of five. As the number of trials is increased, the Binomial distribution approaches the Poisson distribution.

for all $k \in \mathbb{N}_0$. The resulting Poisson distribution $\mathcal{P}$ can be written explicitly as

$$\mathcal{P}_\lambda : k \mapsto e^{-\lambda} \cdot \frac{\lambda^k}{k!} .$$

When the number of trials is not infinite but sufficiently large, the Poisson distribution accurately approximates the binomial distribution. This behavior is illustrated in Figure 2.1. For $\lambda = 5$ and $n = 500$, both distributions are almost indiscernible. A Poisson distribution can be used, for instance, to model the number of lifetime wins in a lottery assuming that one plays each week for a given (long) timespan.

In many real processes, events are not independent. Let us consider the number of persons arriving at a carpark. Here, the events occur in so called *clumps* as each arriving car might contain multiple persons. While the number of cars that arrive in a fixed time interval may be Poisson distributed (at least if we ignore traffic lights, jams, etc.), the number of arriving persons is not. Such a scenario can be described by a *compound Poisson distribution*. Formally, a random variable C with this distribution can be written $C = \sum_{i=0}^{A} B_i$, where A is a Poisson-distributed random variable, all B_i are independent and identically distributed random variables, and all B_i are independent of A. The distribution of C is determined by the common distribution $\Psi = \mathcal{L}(B_i)$, which we call *clump size distribution*, and the *expected number of clumps* $\lambda = \mathbb{E}(A)$. The compound Poisson distribution $\mathcal{CP}_{\lambda,\Psi} = \mathcal{L}(C)$ is now given by

$$\mathcal{CP}_{\lambda,\Psi} : k \mapsto \sum_{i=0}^{\infty} \mathcal{P}_\lambda(i) \cdot \Psi^{*i}(k) .$$

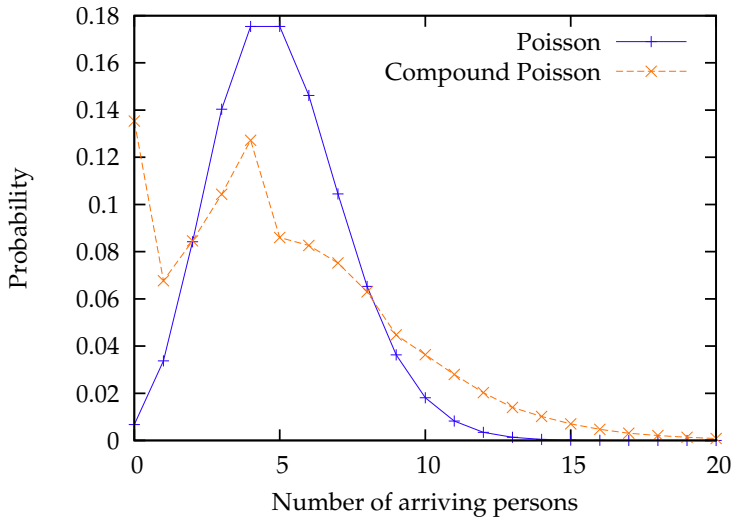


Figure 2.2: Comparison of Poisson distribution with an expectation of five to a compound Poisson distribution with the same expectation whose clump size is distributed uniformly between one and four.

Recall that Ψ^{*i} denotes the i-fold convolution of Ψ with itself. Resuming our car-park example, let us assume that the expected number of persons arriving in a fixed time interval (say, an hour) is five. Now we compare two scenarios. First, all persons come in their own car and their arrival times are hence independent; this corresponds to a Poisson process. Second, we assume that each car contains one to four persons with uniform probability. Therefore, each car contains an expected number of $\mathbb{E}(\Psi) = 2.5$ persons. Since

$$\mathbb{E}(\mathcal{CP}_{\lambda,\Psi}) = \lambda \cdot \mathbb{E}(\Psi)\,,$$

we must set the *expected number of clumps* λ to 2 to get a compound Poisson distribution with an expectation of five. The resulting distributions are plotted in Figure 2.2. Although their expectation is the same, the distributions differ considerably. An expected clump size larger than one leads to an increased variance.

2.3 Finite-Memory Text Models

In Section 1.4, text models have been formally introduced in Definition 1.4. As examples, we discussed uniform and i.i.d. text models. To analyze genomic sequences, however, such simple models are too coarse. *Markovian text models*, for instance, are more elaborate. In a Markovian text model of order r, the character distribution can depend on the last r characters. That means the random variables S_t are no longer (guaranteed to be) independent. *Character-emitting hidden Markov models* (HMMs) are even more sophisticated. They produce a random string by randomly walking the HMM's state space according to a transition function and emitting a character upon

entering a state. The emission is done with respect to a state-specific character distribution. In this section, we see that both Markovian text models and character-emitting hidden Markov models are special cases of *finite-memory text models* which we introduce formally. Similar text models are used by Kucherov et al. (2006), who call them probability transducers.

Definition 2.11 (Finite-memory text model)**.** A *finite-memory text model* is a tuple $(\mathcal{C}, c_0, \Sigma, \varphi)$, where $\mathcal{C}$ is a finite state space (called *context space*), $c_0 \in \mathcal{C}$ a start context, Σ an alphabet, and $\varphi : \mathcal{C} \times \Sigma \times \mathcal{C} \to [0,1]$ with $\sum_{\sigma\in\Sigma, c'\in\mathcal{C}} \varphi(c,\sigma,c') = 1$ for all $c \in \mathcal{C}$. At each time $t \in \mathbb{N}_0$, exactly one state $c \in \mathcal{C}$ is active and the random variable giving this active state is denoted C_t. A probability measure $\mathbb{P}$ is now induced by stipulating

$$\mathbb{P}(C_0 = c) = [\![c = c_0]\!] \tag{2.4}$$

for all $c \in \mathcal{C}$ and

$$\begin{aligned} &\mathbb{P}\big(S_t = \sigma, C_{t+1} = c_{t+1} \,\big|\, S_0 \cdots S_{t-1} = s, C_0 = c_0, \ldots, C_t = c_t\big) \\ &= \mathbb{P}\big(S_t = \sigma, C_{t+1} = c_{t+1} \,\big|\, C_t = c_t\big) \\ &= \varphi(c_t, \sigma, c_{t+1}) \end{aligned} \tag{2.5}$$

for all $t \in \mathbb{N}_0$, $\sigma \in \Sigma$, $s \in \Sigma^t$, and $(c_1, \ldots, c_{t+1}) \in \mathcal{C}^{t+1}$. ◊

Equation (2.5) illustrates that the model given by $(\mathcal{C}, c_0, \Sigma, \varphi)$ generates a random text by moving from context to context and emitting a character at each transition, where $\varphi(c, \sigma, c')$ is the probability of moving from context c to context c' and thereby generating the letter σ.

Lemma 2.12. *Equations* (2.4) *and* (2.5) *imply that*

$$\mathbb{P}(S_0 \cdots S_{t-1} = s, C_0 = c_0, \ldots, C_t = c_t) = \prod_{i=0}^{t-1} \varphi(c_i, s[i], c_{i+1}) \tag{2.6}$$

and

$$\mathbb{P}(S_0 \cdots S_t = s\sigma, C_{t+1} = c) = \sum_{c'\in\mathcal{C}} \mathbb{P}(S_0 \cdots S_{t-1} = s, C_t = c') \cdot \varphi(c', \sigma, c) \tag{2.7}$$

for all $t \in \mathbb{N}_0$, $s \in \Sigma^t$, $\sigma \in \Sigma$ and $c \in \mathcal{C}$.

Proof. Equation (2.6) is correct for $t = 0$ by Equation (2.4). For $t > 0$, Equation (2.6) follows inductively as

$$\begin{aligned} &\mathbb{P}\big(S_0 \cdots S_{t-1} = s, C_0 = c_0, \ldots, C_t = c_t\big) \\ &= \mathbb{P}\big(S_{t-1} = s[t-1], C_t = c_t \,\big|\, S_0 \cdots S_{t-2} = s[..t-2], C_0 = c_0, \ldots, C_{t-1} = c_{t-1}\big) \\ &\qquad \cdot \mathbb{P}\big(S_0 \cdots S_{t-2} = s[..t-2], C_0 = c_0, \ldots, C_{t-1} = c_{t-1}\big) \\ &\overset{\text{(i)}}{=} \varphi\big(c_{t-1}, s[t-1], c_t\big) \cdot \mathbb{P}\big(S_0 \cdots S_{t-2} = s[..t-2], C_0 = c_0, \ldots, C_{t-1} = c_{t-1}\big)\,, \end{aligned}$$

where (i) is correct due to Equation (2.5). Now we prove Equation (2.7).

$$\begin{aligned}
&\mathbb{P}(S_0 \cdots S_t = s\sigma, C_{t+1} = c) \\
&= \sum_{c' \in \mathcal{C}} \mathbb{P}(S_0 \cdots S_t = s\sigma, C_{t+1} = c, C_t = c') \\
&= \sum_{c' \in \mathcal{C}} \mathbb{P}(S_t = \sigma, C_{t+1} = c \,|\, S_0 \cdots S_{t-1} = s, C_t = c') \cdot \mathbb{P}(S_0 \cdots S_{t-1} = s, C_t = c') \\
&\overset{\text{(ii)}}{=} \sum_{c' \in \mathcal{C}} \varphi(c', \sigma, c) \cdot \mathbb{P}(S_0 \cdots S_{t-1} = s, C_t = c')\,,
\end{aligned}$$

where for (ii), we applied Equation (2.5). □

Remark 2.13. For all $s' \in \Sigma^*$, the probability $\mathbb{P}(S_0 \cdots S_{|s'|-1} = s')$ is obtained from Equation (2.6) through marginalization over all $c \in \mathcal{C}$. Therefore, a finite-memory text model consistently assigns probabilities to all finite strings and, hence, the existence of the probability measure $\mathbb{P}$ is guaranteed by Kolmogorov's existence theorem. △

In a slight abuse of nomenclature, we refer to a finite-memory text model simply as text model. It is always clear from the context whether the tuple $(\mathcal{C}, c_0, \Sigma, \varphi)$ or the probability measure it induces is meant.

Example 2.14 (I.i.d. text models)**.** To define a finite-memory text model equivalent to an i.i.d. model, we set $\mathcal{C} = \{\varepsilon\}$ and $\varphi(\varepsilon, \sigma, \varepsilon) = p_\sigma$ for each $\sigma \in \Sigma$, where p_σ is the occurrence probability of letter σ (and ε may be interpreted as an empty context). △

Next, we formally define Markovian text models and see how they can be given in the form of finite-memory text models.

Definition 2.15 (Markovian text model)**.** A text model $\mathbb{P}$ is called *Markovian of order* r if there exist constants $P_{\sigma|s}$ for all $\sigma \in \Sigma$ and $s \in \Sigma^*$ with $|s| \leq r$, such that

$$\begin{aligned}
\mathbb{P}\big(S_t = \sigma' \,\big|\, S_0 \cdots S_{t-1} = s'\big) &= \mathbb{P}\big(S_t = \sigma' \,\big|\, S_{t'} \cdots S_{t-1} = s'[t' \ldots t-1]\big) \\
&= P_{\sigma'|s'[t' \ldots t-1]}
\end{aligned} \tag{2.8}$$

with $t' := \max\{0, t-r\}$ for all $t \in \mathbb{N}_0$, $\sigma' \in \Sigma$, and $s' \in \Sigma^t$. ◊

Example 2.16 (Markovian finite-memory text models)**.** Let constants $P_{\sigma|s}$ for all $\sigma \in \Sigma$ and $s \in \Sigma^*$ with $|s| \leq r$ be given. For convenience, we also define a function $z_r : \Sigma^* \to \Sigma^*$ that truncates a string to length r, i.e. $z_r(s) := s[\max\{0, |s| - r\}..]$. Now we define a finite-memory text model that satisfies Equation (2.8) and, hence, is Markovian of order r. We set $\mathcal{C} := \bigcup_{i=0}^{r} \Sigma^i$, $c_0 := \varepsilon$, and

$$\varphi(c, \sigma, c') := \begin{cases} P_{\sigma|c} & \text{if } c' = z_r(c\sigma)\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

This definition implies that

$$C_t = c \quad \Longleftrightarrow \quad z_r(S_0 \cdots S_{t-1}) = c \tag{2.9}$$

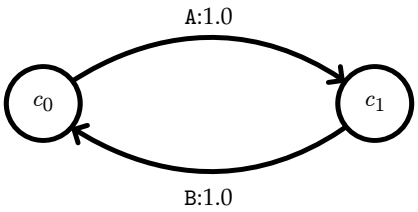


Figure 2.3: Illustration of a periodic text model with two states c_0 and c_1. The transition function is given by $\varphi(c_0, \mathtt{A}, c_1) = 1$, $\varphi(c_1, \mathtt{B}, c_0) = 1$, and zero otherwise.

and, thus,

$$\begin{aligned}\mathbb{P}\big(S_t = \sigma \,\big|\, S_0 \cdots S_{t-1} = s\big) &= \mathbb{P}\big(S_t = \sigma \,\big|\, S_0 \cdots S_{t-1} = s, C_t = z_r(s)\big) \\ &= \sum_{c' \in \mathcal{C}} \mathbb{P}\big(S_t = \sigma, C_{t+1} = c' \,\big|\, S_0 \cdots S_{t-1} = s, C_t = z_r(s)\big) \\ &\overset{(i)}{=} \sum_{c' \in \mathcal{C}} \varphi\big(z_r(s), \sigma, c'\big) = \varphi\big(z_r(s), \sigma, z_r(s\sigma)\big) = P_{\sigma|z_r(s)}\,,\end{aligned}$$

where (i) is implied by Equation (2.5). △

Remark 2.17. *Variable order Markov chains* as introduced by Schulz et al. (2008) can also be transformed into finite-memory text models. △

Although we do not formally introduce hidden Markov models (HMMs), we remark that finite-memory text models have the same expressive power as character-emitting HMMs. That means they allow us to construct the same probability distributions. For a given HMM, we can construct an equivalent text model by using the same state space (contexts) and setting $\varphi(c, \sigma, c') := T(c, c') \cdot \mu_{c'}(\sigma)$, where T and $\mu_{c'}$ are the HMM's transition function and emission distribution attached to state c', respectively. When, on the other hand, a text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ is given, we construct an equivalent HMM by using $\mathcal{C}^2$ as state space and setting

$$T\big((c_1, c_2), (c_1', c_2')\big) := \begin{cases} \sum_{\sigma \in \Sigma} \varphi(c_2, \sigma, c_2') & \text{if } c_2 = c_1', \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\mu_{(c_1, c_2)}(\sigma) := \varphi(c_1, \sigma, c_2).$$

In a finite-memory text model as introduced in Definition 2.11, the transition function φ can be chosen arbitrarily. In particular, periodic or otherwise degenerate text models are allowed. Figure 2.3 shows an example of a text model that produces the string ABABABA... with probability one. Furthermore, its state distribution does not converge to an equilibrium. It is periodic instead: we have $\mathbb{P}\left(C_{2t} = c_0\right) = 1$ and $\mathbb{P}\left(C_{2t+1} = c_1\right) = 1$ for all $t \in \mathbb{N}_0$. To exclude such pathological cases, we make another definition.

Definition 2.18 (Well-behaved finite-memory text model)**.** A finite-memory text model is called *well-behaved* if $\mathbb{P}(S_0 \cdots S_{|s|-1} = s) > 0$ for all $s \in \Sigma^*$ and its state distribution converges to an equilibrium, that is, the limit $\lim_{t\to\infty} \mathcal{L}(C_t)$ exists. In the remainder of this thesis, we assume all text models to be *well-behaved*. ◊

Remark 2.19 (Text model starting in equilibrium)**.** For many applications, it is reasonable to assume random texts to be generated by a model starting in equilibrium. Formally, this means that

$$\mathbb{P}(S_0 \cdots S_{|s|-1} = s) = \lim_{t\to\infty} \mathbb{P}(S_t \cdots S_{t+|s|-1} = s) \tag{2.10}$$

for all $s \in \Sigma^*$. Given a text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ and let $\alpha : \mathcal{C} \to [0,1]$ be its equilibrium state distribution. Then, it can be modified to attain this property by setting $\mathcal{C}' := \mathcal{C} \cup \{c_0'\}$, where c_0' is the new start state, and

$$\varphi' : (c, \sigma, c') \mapsto \begin{cases} \sum_{c''\in\mathcal{C}} \alpha(c'')\varphi(c'', \sigma, c') & \text{if } c = c_0' \text{ and } c' \in \mathcal{C}\,, \\ \varphi(c, \sigma, c') & \text{if } c, c' \in \mathcal{C}\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

Using Equation (2.6), the definition of φ', and that $\alpha(c) := \lim_{t\to\infty} \mathbb{P}(C_t = c)$ for all $c \in \mathcal{C}$, evaluation of both sides of Condition (2.10) yields

$$\lim_{t\to\infty} \sum_{c_t,\ldots,c_{t+|s|}\in\mathcal{C}} \mathbb{P}(C_t = c_t) \prod_{i=0}^{|s|-1} \varphi(c_{t+i}, s[i], c_{t+i+1})\,,$$

proving that Condition (2.10) is indeed satisfied for the newly defined text model. △

2.4 Probabilistic Arithmetic Automata

In many applications, processes can be modeled as chains of operations working on operands that are drawn probabilistically. As an example, let us consider a simple dice game. Suppose you have a bag containing three dice, a 6-faced, a 12-faced, and a 20-faced die. Now a die is drawn from the bag, rolled, and put back. This procedure is repeated n times. In the end one may, for example, be interested in the distribution of the maximum number observed. Many variants can be thought of, for instance, we might start with a value of 0 and each die might be associated with an operation, e.g. the spots seen on the 6-faced die might be subtracted from the current value and the spots on the 12-faced and 20-faced dice might be added. In addition to the distribution of values after n rolls, we can ask for the distribution of the waiting time for reaching a value above a given threshold.

We use a general formal framework, referred to as *probabilistic arithmetic automata (PAAs)*, to directly model such systems and answer the posed questions. In contrast to sampling strategies, this formalism allows us to compute the sought distributions *exactly*. Depending on the implementation, "exactly" can either mean "up to machine

precision" when floating point numbers are used or "mathematically exact" when rational arithmetic is used.

Recall that, in this thesis, we seek to develop an algorithm to discover a motif with optimal statistical significance. PAAs are introduced as they provide a framework to answer various questions regarding a motif's statistical properties, including its significance. This application of PAAs is further explored in Sections 2.6 and 4.3.

2.4.1 Definition

Markov additive processes have been studied in probability theory (see Cinlar, 1972a, and Cinlar, 1972b). In the discrete case, they find applications in bioinformatics, for example to model masses of protein fragments resulting from cleavage reactions as done by Kaltenbach (2006). The formal definition of PAAs (as found below) has been introduced by Marschall and Rahmann (2008). They can be seen as generalized discrete Markov additive processes as they are not restricted to additions but allow arbitrary operations. Applications of PAAs in computational biology have been explored by Herms (2009).

Definition 2.20 (Probabilistic arithmetic automaton)**.** A *probabilistic arithmetic automaton* P is a tuple

$$P = \left(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu = (\mu_q)_{q\in\mathcal{Q}}, \theta = (\theta_q)_{q\in\mathcal{Q}}\right),$$

where

- $\mathcal{Q}$ is a finite set of states,
- $q_0 \in \mathcal{Q}$ is called *start state*,
- $T : \mathcal{Q} \times \mathcal{Q} \to [0, 1]$ is a transition function with $\sum_{q'\in\mathcal{Q}} T(q, q') = 1$ for all $q \in \mathcal{Q}$, i.e. $\left(T(q, q')\right)_{q,q'\in\mathcal{Q}}$ is a stochastic matrix,
- $\mathcal{V}$ is a set called *value set*,
- $v_0 \in \mathcal{V}$ is called *start value*,
- $\mathcal{E}$ is a finite set called *emission set*,
- each $\mu_q : \mathcal{E} \to [0, 1]$ is an emission distribution associated with state q,
- each $\theta_q : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$ is an operation associated with state q.

$\Diamond$

We attach the following semantics: At first, the automaton is in its start state q_0, as for a classical deterministic finite automaton (DFA). In a DFA, the transitions are triggered by input symbols. In a PAA, the transitions are purely probabilistic; $T(q, q')$ gives the probability of going from state q to state q'. Note that the tuple $(\mathcal{Q}, T, \boldsymbol{\delta}_{q_0})$ defines a Markov chain on state set $\mathcal{Q}$ with transition matrix T, where the initial distribution $\boldsymbol{\delta}_{q_0}$ is the Dirac distribution assigning probability one to state q_0 and zero to all other states.

While going from state to state, a PAA performs a chain of calculations on a set of values $\mathcal{V}$. In the beginning, it starts with the value v_0. Whenever a state transition is

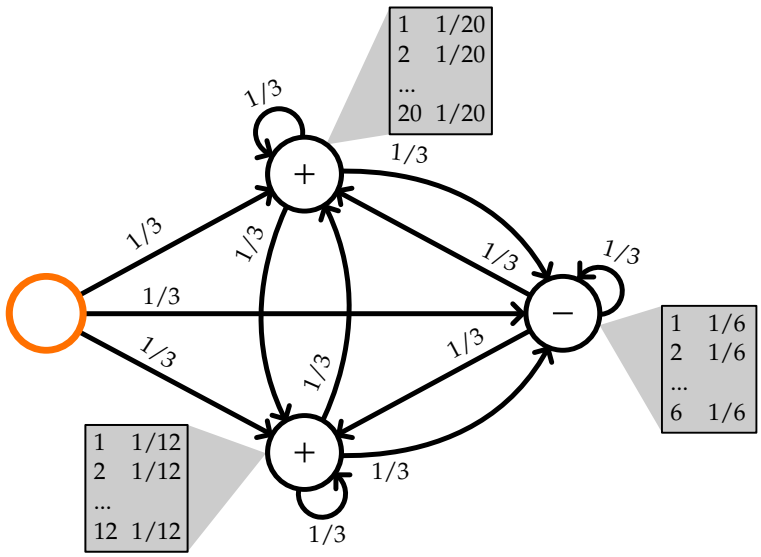


Figure 2.4: Illustration of a PAA for the dice example. Each of the three dice is represented by a state (circles). In each circle the associated operation is printed. Emission distributions are depicted as gray boxes. Each arrow stands for a possible state transition and is labeled with the transition propability. The start state is drawn in orange color. Its emission distribution and operation are irrelevant and thus omitted from the figure.

made, the entered state, say state q, generates an emission from $\mathcal{E}$ according to the distribution μ_q. The current value and this emission are then subject to the operation θ_q, resulting in the next value from the value set $\mathcal{V}$. Notice that the Markov chain $(\mathcal{Q}, T, \boldsymbol{\delta}_{q_0})$, together with the emission set $\mathcal{E}$ and the distributions $\mu = (\mu_q)_{q\in\mathcal{Q}}$, defines a hidden Markov model (HMM). In the context of HMMs, however, the focus usually rests on the sequence of emissions, whereas we are interested in the value resulting from a chain of operations on these emissions.

By introducing PAAs, we emphasize that many systems can naturally be modeled as chains of operations whose results are of interest. When compared to Markov chains, PAAs do not offer an increase in expressive power. In fact, from a theoretical point of view, every PAA might be seen as a Markov chain on the state space $\mathcal{Q} \times \mathcal{V}$. Thus, we advocate PAAs not because of their expressive power but for their merits as a modeling technique. As we see in Sections 2.6, 2.7, and 4.3, the framework lends itself to many applications and often allows simple and intuitive problem formulations. Before we formalize the introduced semantics in Definition 2.22, we come back to the dice example.

Example 2.21 (Dice)**.** We model each of the three dice as a PAA state. All transition probabilities equal $1/3$ and the emissions have uniform distributions over the number of faces of the respective die. If we are interested in the maximum, each state's operation is to take the maximum. In general, we can associate individual operations

with each state (each die), for instance: "sum up all numbers from the 12- and 20-faced dice and subtract the numbers seen on the 6-faced die". The value set is $\mathbb{Z}$, the start value is naturally $v_0 = 0$. The corresponding PAA is illustrated in Figure 2.4. △

Definition 2.22 (Stochastic processes induced by a PAA)**.** For a given PAA $P = (\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$, we define its *state process* $(Q_t^P)_{t\in\mathbb{N}_0}$ to be a Markov chain such that $Q_0^P \equiv q_0$ and

$$\begin{aligned}\mathbb{P}\left(Q_{t+1}^P = q_{t+1} \,\middle|\, Q_t^P = q_t, \ldots, Q_0^P = q_0\right) &= \mathbb{P}\left(Q_{t+1}^P = q_{t+1} \,\middle|\, Q_t^P = q_t\right) \\ &= T(q_t, q_{t+1})\end{aligned} \tag{2.11}$$

for all $q_0, \ldots q_{t+1} \in \mathcal{Q}$. Further, we define the *emission process* $(E_t^P)_{t\in\mathbb{N}_0}$ such that

$$\begin{aligned}&\mathbb{P}\left(E_t^P = e \,\middle|\, Q_0^P = q_0, \ldots, Q_t^P = q_t, E_0^P = e_0, \ldots, E_{t-1}^P = e_{t-1}\right) \\ &= \mathbb{P}\left(E_t^P = e \,\middle|\, Q_t^P = q\right) = \mu_q(e)\,,\end{aligned} \tag{2.12}$$

i.e. the current emission depends solely on the current state. Then, we use $(Q_t^P)_{t\in\mathbb{N}_0}$ and $(E_t^P)_{t\in\mathbb{N}_0}$ to define the *process of values* $(V_t^P)_{t\in\mathbb{N}_0}$ resulting from the performed operations:

$$V_0^P \equiv v_0 \quad \text{and} \quad V_t^P = \theta_{Q_t^P}\left(V_{t-1}^P, E_t^P\right)\,. \tag{2.13}$$

If the considered PAA is clear from the context, we omit the superscript P and write $(Q_t)_{t\in\mathbb{N}_0}$, $(E_t)_{t\in\mathbb{N}_0}$, and $(V_t)_{t\in\mathbb{N}_0}$, respectively. ◊

2.4.2 Computing State-Value Distributions

We describe two algorithms to compute the distribution values probabilistically computed by a PAA. Formally, we seek to calculate the distribution $\mathcal{L}(V_n)$ of the random variable V_n for a given n. The idea is to compute the joint distribution $\mathcal{L}(Q_n, V_n)$ and then to derive the sought distribution by marginalization over all states:

$$\mathbb{P}(V_n = v) = \sum_{q\in\mathcal{Q}} \mathbb{P}(Q_n = q, V_n = v)\,. \tag{2.14}$$

For the sake of a shorter notation, we define $f_t(q, v) := \mathbb{P}(Q_t = q, V_t = v)$ for $t \in \mathbb{N}_0$, $q \in \mathcal{Q}$, $v \in \mathcal{V}$.

Even when $\mathcal{V}$ is infinite, the range of V_t is finite for all t as it is a function of the states and emissions up to time t and these are finite sets. We define $\mathcal{V}_t := \mathrm{range}(V_t)$ and $\vartheta_n := \max_{0\le t\le n} |\mathcal{V}_t|$. By Definition 2.22, the value computed by a PAA is determined by the sequence of visited states and made emissions. Up to step n, there are at most $(|\mathcal{Q}|\cdot|\mathcal{E}|)^n$ such sequences that have non-zero probability and, hence, $\vartheta_n \le (|\mathcal{Q}|\cdot|\mathcal{E}|)^n$ for all $n \in \mathbb{N}_0$. Therefore, all actual computations are on finite sets. In many applications, ϑ_n grows only polynomially (even linearly) with n. Running times of algorithms are given in terms of ϑ_n.

Basic Algorithm

We discuss an algorithm to compute the distribution $f_n = \mathcal{L}(Q_n, V_n)$. A basic recurrence relation follows from Definitions 2.20 and 2.22.

Lemma 2.23 (State-value recurrence)**.** *For a given PAA, the state-value distribution can be computed by*

$$f_0(q,v) = \begin{cases} 1 & \text{if } q = q_0 \text{ and } v = v_0\,, \\ 0 & \text{otherwise}\,, \end{cases} \tag{2.15}$$

and

$$f_{t+1}(q,v) = \sum_{q' \in \mathcal{Q}} \sum_{(v',e) \in \theta_q^{-1}(v)} f_t(q',v') \cdot T(q',q) \cdot \mu_q(e) \tag{2.16}$$

for $t \geq 0$, where $\theta_q^{-1}(v)$ denotes the inverse image set of v under θ_q.

Proof. Equation (2.15) follows directly from (2.11) and (2.13). Let us verify Equation (2.16):

$$\begin{aligned}
f_{t+1}(q,v) &= \mathbb{P}\left(Q_{t+1} = q, V_{t+1} = v\right) \\
&= \sum_{q' \in \mathcal{Q}} \sum_{v' \in \mathcal{V}} \sum_{e \in \mathcal{E}} \mathbb{P}\left(Q_{t+1} = q, V_{t+1} = v, Q_t = q', V_t = v', E_{t+1} = e\right) \\
&= \sum_{q' \in \mathcal{Q}} \sum_{v' \in \mathcal{V}} \sum_{e \in \mathcal{E}} \mathbb{P}\left(Q_{t+1} = q, V_{t+1} = v, E_{t+1} = e \,\middle|\, Q_t = q', V_t = v'\right) \cdot f_t(q',v') \\
&= \sum_{q' \in \mathcal{Q}} \sum_{v' \in \mathcal{V}} \sum_{e \in \mathcal{E}} \mathbb{P}\left(V_{t+1} = v \,\middle|\, Q_{t+1} = q, E_{t+1} = e, Q_t = q', V_t = v'\right) \\
&\qquad \cdot \mathbb{P}\left(Q_{t+1} = q, E_{t+1} = e \,\middle|\, Q_t = q', V_t = v'\right) \cdot f_t(q',v') \\
&= \sum_{q' \in \mathcal{Q}} \sum_{v' \in \mathcal{V}} \sum_{e \in \mathcal{E}} [\![\theta_q(v',e) = v]\!] \\
&\qquad \cdot \mathbb{P}\left(Q_{t+1} = q, E_{t+1} = e \,\middle|\, Q_t = q', V_t = v'\right) \cdot f_t(q',v') \\
&= \sum_{q' \in \mathcal{Q}} \sum_{(v',e) \in \theta_q^{-1}(v)} \underbrace{\mathbb{P}\left(Q_{t+1} = q, E_{t+1} = e \,\middle|\, Q_t = q', V_t = v'\right)}_{(*)} \cdot f_t(q',v')\,.
\end{aligned}$$

We further evaluate the expression $(*)$:

$$\begin{aligned}
(*) \;&=\; \mathbb{P}\left(Q_{t+1} = q, E_{t+1} = e \,\middle|\, Q_t = q', V_t = v'\right) \\
&= \underbrace{\mathbb{P}\left(E_{t+1} = e \,\middle|\, Q_t = q', Q_{t+1} = q, V_t = v'\right)}_{\overset{(i)}{=} \mu_q(e)} \cdot \underbrace{\mathbb{P}\left(Q_{t+1} = q \,\middle|\, Q_t = q', V_t = v'\right)}_{\overset{(ii)}{=} T(q',q)}\,,
\end{aligned}$$

where (i) is true because of (2.12) and (2.13), and (ii) follows from the fact that $(Q_t)_{t \in \mathbb{N}_0}$ is a Markov chain. □

We start with the distribution f_0 and calculate the subsequent distributions by applying Equation (2.16) until we obtain the desired f_n. A straightforward implementation

Algorithm 2.1 Compute the value distribution of a PAA using a push strategy.

Input: PAA $P = (\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$, number of steps n
Output: Distribution $\mathcal{L}(V_n)$

PAA-VALUE-DISTRIBUTION(P, n)

```
1  initialize f_0(q, v) = [[q = q_0 and v = v_0]] for all q ∈ Q and v ∈ V_0
2  for t = 1 to n
3      initialize f_t(q, v) = 0 for all q ∈ Q and v ∈ V_t
4      for q ∈ Q and v ∈ V_{t-1}
5          for q' ∈ Q and e ∈ E
6              v' = θ_{q'}(v, e)
7              f_t(q', v') = f_t(q', v') + f_{t-1}(q, v) · T(q, q') · μ_{q'}(e)
8  return f_n
```

of Equation (2.16) results in a *pull strategy*; that means each entry in the table representing f_{t+1} is calculated by "pulling over" the required probabilities from table f_t. This approach makes it necessary to calculate θ^{-1} in a preprocessing step. In order to avoid this, we may implement a *push strategy*, meaning that we iterate over all entries in f_t rather than f_{t+1} and "push" the encountered summands over to the appropriate places in table f_{t+1}. In effect, we just change the order of summation. Algorithm 2.1 shows the push strategy in detail.

In the course of the computation, we have to store two distributions, f_t and f_{t+1}, at a time. Once f_{t+1} is calculated, f_t can be discarded. Since the table at time t has a size of $|\mathcal{Q}| \times |\mathcal{V}_t|$, the total space consumption is $\mathcal{O}(|\mathcal{Q}| \cdot \vartheta_n)$. Computing f_t from f_{t-1} takes $\mathcal{O}(|\mathcal{Q}| \cdot |\mathcal{V}_t| + |\mathcal{Q}|^2 \cdot |\mathcal{V}_{t-1}| \cdot |\mathcal{E}|)$ operations, as can be seen from Algorithm 2.1. We arrive at the following lemma.

Lemma 2.24. *Given a PAA $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$, the distribution of values $\mathcal{L}(V_n)$ can be computed using $\mathcal{O}(n \cdot |\mathcal{Q}|^2 \cdot \vartheta_n \cdot |\mathcal{E}|)$ operations and $\mathcal{O}(|\mathcal{Q}| \cdot \vartheta_n)$ space.*

Remark 2.25 (Number of operations and runtimes)**.** In Lemma 2.24 and in the following, runtimes of algorithms are reported in terms of the number of operations. This includes both arithmetic operations and the operations θ_q of a PAA. Usually, but not always, both types of operations can be performed in constant time. Using exact rational arithmetic, for instance, leads to operations that take more than constant time. As the operation θ_q can be chosen arbitrarily for every PAA state q, it might also take more than constant time. △

Doubling Technique

When n is large, executing PAA-VALUE-DISTRIBUTION(P, n) is slow. In this section, we present an alternative algorithm that can be favorable for large n. To derive this algorithm, we consider the conditional probability

$$U^{(t)}(q_1, q_2, v_1, v_2) := \mathbb{P}\left(Q_{t_0+t} = q_2, V_{t_0+t} = v_2 \,\middle|\, Q_{t_0} = q_1, V_{t_0} = v_1\right). \tag{2.17}$$

Note that $U^{(t)}$ does not depend on t_0, because transition as well as emission probabilities do not change over "time" (a property called *homogeneity*). Once $U^{(n)}$ is known, we can simply read off the desired distribution $\mathcal{L}(Q_n, V_n)$:

$$\mathbb{P}(Q_n = q, V_n = v) = U^{(n)}(q_0, q, v_0, v)\,. \tag{2.18}$$

The following lemma shows how $U^{(t)}$ can be computed.

Lemma 2.26. *Let $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ be a PAA and $(Q_t)_{t\in\mathbb{N}_0}$ and $(V_t)_{t\in\mathbb{N}_0}$ its state and value process, respectively. Then,*

$$U^{(1)}(q_1, q_2, v_1, v_2) = T(q_1, q_2) \cdot \sum_{\substack{e\in\mathcal{E}:\\ \theta_{q_2}(v_1,e)=v_2}} \mu_{q_2}(e) \tag{2.19}$$

and, for all $t_1 \in \mathbb{N}_0$ and $t_2 \in \mathbb{N}_0$,

$$U^{(t_1+t_2)}(q_1, q_2, v_1, v_2) = \sum_{q'\in\mathcal{Q}} \sum_{v'\in\mathcal{V}} U^{(t_1)}(q_1, q', v_1, v') \cdot U^{(t_2)}(q', q_2, v', v_2)\,. \tag{2.20}$$

Using these recurrences, the distribution of values $\mathcal{L}(V_n)$ can be computed using $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3 \cdot \vartheta_n^3)$ operations and $\mathcal{O}(|\mathcal{Q}|^2 \cdot \vartheta_n^2)$ space.

Proof. Equation (2.19) follows from Definition 2.22 and Equation (2.20) follows from the Chapman-Kolmogorov Equation for homogeneous Markov chains when the PAA is seen as a Markov chain with state space $\mathcal{Q} \times \mathcal{V}$. Computing $U^{(t_1+t_2)}$ from $U^{(t_1)}$ and $U^{(t_2)}$ takes $\mathcal{O}(|\mathcal{Q}|^3 \cdot \vartheta_n^3)$ operations, as follows from Equation (2.20). On the other hand, one step suffices to obtain $U^{(2t)}$ from $U^{(t)}$. Thus, we can compute all $U^{(2^b)}$ for $0 \le b < \lceil \log(n) \rceil$ in $\lceil \log(n) \rceil$ steps, which in turn can be combined into $U^{(n)}$ in at most $\lceil \log(n) \rceil$ steps. □

The doubling technique is asymptotically faster if ϑ_n is $o(\sqrt{n/\log n})$ and $\mathcal{Q}$ and $\mathcal{E}$ have constant size.

2.4.3 Waiting Times

Besides calculating the distribution of values after a fixed number of steps, we can ask for the distribution of the number of steps needed to reach a certain value or a certain state. Such *waiting time* problems play an important role in many applications, for example in the analysis of peptide fragments resulting from cleavage reaction and in the analysis of length distributions of 454 sequencing reads. Both examples are discussed by Marschall et al. (2010). A classical treatment of waiting time problems is given by Feller (1968). Applications to occurrence problems in texts are reviewed by Reinert et al. (2000).

Definition 2.27 (Waiting time for a value)**.** The *waiting time* for a set of target values $\mathcal{T} \subset \mathcal{V}$ is a random variable defined as $W_\mathcal{T} := \min\{t \in \mathbb{N}_0 \,|\, V_t \in \mathcal{T}\}$ if this set is not empty, and defined as infinity otherwise. ◊

While $\mathbb{P}(W_{\mathcal{T}} \geq t)$ may be greater than zero for all $t \in \mathbb{N}$, we are frequently only interested in the distribution up to a fixed time t_{max}.

Lemma 2.28. *Let $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ be a PAA and $\mathcal{T} \subset \mathcal{V}$. Then, the probabilities*

$$\mathcal{L}\big(W_{\mathcal{T}}\big)(0), \ldots, \mathcal{L}\big(W_{\mathcal{T}}\big)(t_{max})$$

can be computed using $\mathcal{O}\big(t_{max} \cdot |\mathcal{Q}|^2 \cdot |\mathcal{E}| \cdot \vartheta_{t_{max}}\big)$ operations and $\mathcal{O}\big(|\mathcal{Q}| \cdot \vartheta_{t_{max}}\big)$ space. Alternatively, this can be done using $\mathcal{O}\big(\log t_{max} \cdot |\mathcal{Q}|^3 \cdot (\vartheta_{t_{max}})^3\big)$ operations and $\mathcal{O}\big(|\mathcal{Q}|^2 \cdot (\vartheta_{t_{max}})^2\big)$ space.

Proof. We construct a modified PAA by defining a new value set $\mathcal{V}' := (\mathcal{V} \setminus \mathcal{T}) \cup \{\bullet, \circ\}$, assuming (without loss of generality) that $\bullet, \circ \notin \mathcal{V}$, and new operations

$$\theta'_q(v, e) := \begin{cases} \theta_q(v, e) & \text{if } v \notin \{\bullet, \circ\} \text{ and } \theta_q(v, e) \notin \mathcal{T}, \\ \bullet & \text{if } v \notin \{\bullet, \circ\} \text{ and } \theta_q(v, e) \in \mathcal{T}, \\ \circ & \text{if } v \in \{\bullet, \circ\}, \end{cases}$$

for all $q \in \mathcal{Q}$. Let V'_t be the modified value process. Using the modified PAA, the probability of waiting time t can be expressed as

$$\mathbb{P}(W_{\mathcal{T}} = t) = \mathbb{P}(V'_t = \bullet).$$

Runtime and space bounds follow from Lemmas 2.24 and 2.26. □

Besides waiting for a value (or a set of values), it can be interesting to consider the waiting time for a state (or a set of states). Since a PAA's state process does not depend on emission and value processes, the remainder of this section solely concerns the Markov chain $(\mathcal{Q}, T, \boldsymbol{\delta}_{q_0})$, which is part of the PAA $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$. Waiting times in Markov chains are a well-studied topic with particular interest in pattern occurrences (Reinert et al., 2000) and queuing theory (Brémaud, 1999).

For completeness, we briefly state the construction here.

Definition 2.29 (Waiting time for a state)**.** The *waiting* time for a set of target states $\mathcal{U} \subset \mathcal{Q}$ is a random variable defined as $W_{\mathcal{U}} := \min\{t \in \mathbb{N}_0 \,|\, Q_t \in \mathcal{U}\}$ if this set is not empty and defined as infinity otherwise. ◊

Lemma 2.30. *Let $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ be a PAA, $\alpha : \mathcal{Q} \to [0, 1]$ be a probability distribution on $\mathcal{Q}$, and $\mathcal{U} \subset \mathcal{Q}$ be a set of target states. Consider the Markov chain $(\mathcal{Q}, T, \alpha)$, let $(Q'_t)_{t \in \mathbb{N}_0}$ be its state process, and let $W'_{\mathcal{U}} := \min\{t \in \mathbb{N}_0 \,|\, Q'_t \in \mathcal{U}\}$ be the waiting time for states $\mathcal{U}$. Then,*

$$\mathcal{L}(W'_{\mathcal{U}})(0), \ldots, \mathcal{L}(W'_{\mathcal{U}})(t_{max})$$

can be computed using $\mathcal{O}(t_{max} \cdot |\mathcal{Q}|^2)$ operations and $\mathcal{O}(|\mathcal{Q}|)$ space, or using $\mathcal{O}(\log t_{max} \cdot |\mathcal{Q}|^3)$ operations and $\mathcal{O}(|\mathcal{Q}|^2)$ space using a doubling technique. If $\alpha = \boldsymbol{\delta}_{q_0}$, then $W_{\mathcal{U}} = W'_{\mathcal{U}}$.

Proof. As in the proof of Lemma 2.28, we introduce an aggregation state $\bullet$ to replace $\mathcal{U}$ and an absorbing state $\circ$ to "flush" $\bullet$. Then $\mathbb{P}(W'_{\mathcal{U}} = t) = \mathbb{P}(Q'_t = \bullet)$. □

When $\alpha = \boldsymbol{\delta}_{q_0}$, the above lemma yields the waiting time for the first event of reaching one of the states in $\mathcal{U}$. We further consider the waiting time of a return event

$$W_{\mathcal{U}}^{t_0} := \min\{t \in \mathbb{N} \mid Q_{t_0+t} \in \mathcal{U}\}.$$

If the Markov chain is aperiodic and irreducible, it has a unique stationary state distribution against which the PAA state distribution converges exponentially fast. We can then use Lemma 2.30 to compute

$$\lim_{t\to\infty} \mathbb{P}\big(W_{\mathcal{U}}^t = t' \;\big|\; Q_t \in \mathcal{U}\big) \tag{2.21}$$

for each $t' \in \mathbb{N}$ by choosing α in Lemma 2.30 as the stationary distribution restricted to $\mathcal{U}$.

2.5 Deterministic Arithmetic Automata

In this section, we discuss the construction of PAAs modeling the deterministic processing of random sequences. That means we consider a mechanism that processes sequences character by character and thereby deterministically computes a value for a given string. A pattern matching algorithm that computes the number of matches in a given sequence might serve as an example. We ask for the distribution of resulting values when such a *deterministic* computation is applied to *random* strings. To this end, we define *deterministic arithmetic automata* (DAAs) and combine them with a text model to obtain a PAA. In Section 2.6, we see how this technique can be employed to compute a significance score for a given motif.

2.5.1 Definition

As for an ordinary deterministic finite automaton (DFA), the state transitions of a DAA are triggered by characters, but, additionally, a DAA performs a computation while moving from state to state. Thus, the relation of DAAs to DFAs is similar to the relation of PAAs to Markov Chains. In both cases, the state space $\mathcal{Q}$ is augmented by a value space $\mathcal{V}$.

Definition 2.31 (Deterministic arithmetic automaton, DAA)**.** A *deterministic arithmetic automaton* is a tuple

$$D = \big(\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}}\big),$$

where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the start state, Σ is a finite alphabet, $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is a transition function, $\mathcal{V}$ is a set of values, $v_0 \in \mathcal{V}$ is called the start value, $\mathcal{E}$ is a finite set of emissions, $\eta_q \in \mathcal{E}$ is the emission associated with state q, and $\theta_q : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$ is a binary operation associated with state q. ◊

Informally, a DAA starts in state q_0 with value v_0 and reads a sequence of symbols from Σ. Being in state q with value v, upon reading $\sigma \in \Sigma$, the DAA performs a state

transition to $q' := \delta(q, \sigma)$ and updates the value to $v' := \theta_{q'}(v, \eta_{q'})$ using the operation and emission of the new state q'.

DAAs can be seen as DFAs over the state space $\mathcal{Q} \times \mathcal{V}$. Therefore, one could argue, it is not necessary not define them explicitly. However, for many applications the distinction between states and values is quite natural and modeling it explicitly leads to more intuitive models. Furthermore, they are a deterministic counterpart to PAAs and allow us to construct PAAs elegantly. In Section 2.7, for instance, we exemplify their utility for the analysis of pattern matching algorithms.

Definition 2.32 (Joint transition function of states and values)**.** The *joint transition function* of states and values of a given DAA $\big(\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}}\big)$ is defined by

$$\bar{\delta} : (\mathcal{Q} \times \mathcal{V}) \times \Sigma \to (\mathcal{Q} \times \mathcal{V}), \qquad \bar{\delta}\big((q, v), \sigma\big) := \big(\delta(q, \sigma)\,,\, \theta_{\delta(q,\sigma)}(v, \eta_{\delta(q,\sigma)})\big).$$

As done for DFAs in Equation (2.1) on Page 19, we extend the definitions of δ and $\bar{\delta}$ inductively from Σ to Σ^* in their second argument. Therefore, they are seen as functions $\delta : \mathcal{Q} \times \Sigma^* \to \mathcal{Q}$ and $\bar{\delta} : (\mathcal{Q} \times \mathcal{V}) \times \Sigma^* \to (\mathcal{Q} \times \mathcal{V})$, respectively. ◊

Definition 2.33 (Value computed by a DAA)**.** Let a DAA D be given such that $D = \big(\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}}\big)$ and let $\bar{\delta}$ be its joint transition function. When $\bar{\delta}\big((q_0, v_0), s\big) = (q, v)$ for a given $s \in \Sigma^*$ and some $q \in \mathcal{Q}$, we say that D computes value v for input s and define

$$\mathit{value}_D(s) := v\,.$$

◊

Example 2.34 (DAA constructed from a DFA)**.** Let a DFA $(\mathcal{Q}, \Sigma, \delta, q_0, F)$ be given. To obtain a DAA that counts how many times the DFA visits an accepting state when reading $s \in \Sigma^*$, let $\mathcal{E} := \{0, 1\}$ and define $\eta_q := [\![q \in F]\!]$. Further define $\mathcal{V} := \mathbb{N}$ and $v_0 := 0$, and let the operation in each state be the usual addition: $\theta_q(v, e) := v + e$ for all q. Then, $\mathit{value}_D(s)$ is the desired count. △

2.5.2 Constructing PAAs from DAAs and Text Models

The distribution of values resulting when a DAA processes random texts can be computed by constructing a PAA from a DAA and a text model. This construction is formalized in the next lemma.

Lemma 2.35 (DAA + Text model → PAA)**.** *Let a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ and a DAA $D = \big(\mathcal{Q}^D, q_0^D, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}^D}, (\theta_q^D)_{q\in\mathcal{Q}^D}\big)$ be given and define*

- *a state space $\mathcal{Q} := \mathcal{Q}^D \times \mathcal{C}$,*
- *a start state $q_0 := (q_0^D, c_0)$,*
- *transition probabilities*

$$T\big((q^D, c), (q'^D, c')\big) := \sum_{\sigma\in\Sigma:\, \delta(q^D,\sigma)=q'^D} \varphi(c, \sigma, c'), \tag{2.22}$$

- *(Dirac) emission distributions* $\mu_{(q^D,c)}(e) := [\![e = \eta_{q^D}]\!]$ *for all* $(q^D, c) \in \mathcal{Q}$,
- *operations* $\theta_{(q^D,c)}(v, e) := \theta^D_{q^D}(v, e)$ *for all* $(q^D, c) \in \mathcal{Q}$.

Then, $P = \big(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu = (\mu_q)_{q\in\mathcal{Q}}, \theta = (\theta_q)_{q\in\mathcal{Q}}\big)$ *is a PAA with*

$$\mathcal{L}(V_t) = \mathcal{L}\big(value_D(S_0 \dots S_{t-1})\big)$$

for all $t \in \mathbb{N}_0$*, where* $(S_t)_{t\in\mathbb{N}_0}$ *is a random text distributed according to the text model* $(\mathcal{C}, c_0, \Sigma, \varphi)$.

Proof. P is a PAA by Definition 2.20 on Page 28. As in Section 2.4.2, we define

$$f_t(q, v) := \mathbb{P}(Q_t = q, V_t = v)\,.$$

Then, the distribution $\mathcal{L}(V_t)$ is obtained by marginalization over all PAA states for each value v, i.e.

$$\mathcal{L}\big(V_t\big)(v) = \sum_{q\in\mathcal{Q}} f_t(q, v) = \sum_{q^D\in\mathcal{Q}^D} \sum_{c\in C} f_t\big((q^D, c), v\big)\,.$$

The probability that the text model generates a text $s \in \Sigma^t$ and after that is in state c such that the DAA D computes the value v for s and after reading s is in state q^D is given by

$$\sum_{s\in\Sigma^t} [\![\bar\delta\big((q_0^D, v_0), s\big) = (q^D, v)]\!] \cdot \mathbb{P}(S_0 \cdots S_{t-1} = s, C_t = c)\,,$$

where we have used Definition 2.33 to obtain the Iverson bracket. Therefore, we can prove that $\mathcal{L}(V_t) = \mathcal{L}\big(value_D(S_0 \dots S_{t-1})\big)$ by showing that

$$f_t\big((q^D, c), v\big) = \sum_{s\in\Sigma^t} [\![\bar\delta\big((q_0^D, v_0), s\big) = (q^D, v)]\!] \cdot \mathbb{P}(S_0 \cdots S_{t-1} = s, C_t = c) \tag{2.23}$$

for all $q^D \in \mathcal{Q}^D$, $c \in \mathcal{C}$, $v \in \mathcal{V}$, and $t \in \mathbb{N}_0$.

For $t = 0$, Equation (2.23) is correct by definitions of PAAs, DAAs and text models. For $t > 0$ we prove it inductively. Assume (2.23) to be correct for all t' with $0 \le t' < t$.

$$f_t\big(\underbrace{(q^D, c)}_{=:q}, v\big) \tag{2.24}$$

$$= \sum_{q'\in\mathcal{Q}} \sum_{(v',e)\in\theta_q^{-1}(v)} f_{t-1}(q', v') \cdot T(q', q) \cdot \mu_q(e) \tag{2.25}$$

$$= \sum_{q'\in\mathcal{Q}} \sum_{(v',e)\in\mathcal{V}\times\mathcal{E}} [\![\theta^D_{q^D}(v', e) = v]\!] \cdot f_{t-1}(q', v') \cdot T(q', q) \cdot [\![\eta_{q^D} = e]\!] \tag{2.26}$$

$$\begin{aligned} = \sum_{q'^D\in\mathcal{Q}^D} \sum_{c'\in\mathcal{C}} \sum_{(v',e)\in\mathcal{V}\times\mathcal{E}} & [\![\theta^D_{q^D}(v', e) = v]\!] \cdot [\![\eta_{q^D} = e]\!] \cdot f_{t-1}(q', v') \\ & \cdot \sum_{\sigma\in\Sigma} [\![\delta(q'^D, \sigma) = q^D]\!] \cdot \varphi(c', \sigma, c) \end{aligned} \tag{2.27}$$

$$
\begin{aligned}
&= \sum_{s\in\Sigma^{t-1}} \sum_{\sigma\in\Sigma} \sum_{q'^D\in\mathcal{Q}^D} \sum_{c'\in\mathcal{C}} \sum_{(v',e)\in\mathcal{V}\times\mathcal{E}} [\![\theta^D_{q^D}(v',e)=v]\!]\cdot[\![\eta_{q^D}=e]\!] \\
&\qquad \cdot [\![\delta(q'^D,\sigma)=q^D]\!]\cdot[\![\bar\delta\big((q_0^D,v_0),s\big)=(q'^D,v')]\!] \\
&\qquad \cdot \mathbb{P}(S_0\cdots S_{t-2}=s, C_{t-1}=c')\cdot\varphi(c',\sigma,c)
\end{aligned} \tag{2.28}
$$

$$
\begin{aligned}
&= \sum_{s\sigma\in\Sigma^{t}} \sum_{q'^D\in\mathcal{Q}^D} \sum_{(v',e)\in\mathcal{V}\times\mathcal{E}} [\![\theta^D_{q^D}(v',e)=v]\!]\cdot[\![\eta_{q^D}=e]\!]\cdot[\![\bar\delta\big((q_0^D,v_0),s\big)=(q'^D,v')]\!] \\
&\qquad \cdot [\![\delta(q'^D,\sigma)=q^D]\!]\cdot\mathbb{P}(S_0\cdots S_{t-1}=s\sigma, C_t=c)
\end{aligned} \tag{2.29}
$$

$$
= \sum_{s\sigma\in\Sigma^{t}} [\![\bar\delta\big((q_0^D,v_0),s\sigma\big)=(q^D,v)]\!]\cdot\mathbb{P}(S_0\cdots S_{t-1}=s\sigma, C_t=c) \tag{2.30}
$$

In the above derivation, step (2.24)→(2.25) follows from (2.16). Step (2.25)→(2.26) follows from the definitions of θ_q and μ_q. Step (2.26)→(2.27) uses the definitions of T and $\mathcal{Q}$ in Lemma 2.35. Step (2.27)→(2.28) uses the induction assumption. Step (2.28)→(2.29) uses Lemma 2.12 on Page 24. The final step (2.29)→(2.30) follows by combining the four Iverson brackets summed over q'^D and (v',e) into a single Iverson bracket. □

Remark 2.36. In the above construction, states having zero probability of being reached from q_0 may be omitted from $\mathcal{Q}$ and T. △

Lemma 2.37 (PAA from DAA; Construction time and space)**.**

1. *For a PAA constructed according to Lemma 2.35, the value distribution $\mathcal{L}(V_n)$, or the joint state-value distribution, can be computed with $\mathcal{O}(n\cdot|\mathcal{Q}^D|\cdot|\Sigma|\cdot|\mathcal{C}|^2\cdot\vartheta_n)$ operations using $\mathcal{O}(|\mathcal{Q}^D|\cdot|\mathcal{C}|\cdot\vartheta_n)$ space. The same statement holds for computing the waiting time distribution up to time n.*

2. *If for all $c\in\mathcal{C}$ and $\sigma\in\Sigma$, there exists at most one $c'\in\mathcal{C}$ such that $\varphi(c,\sigma,c')>0$, then the number of required operations is bounded by $\mathcal{O}(n\cdot|\mathcal{Q}^D|\cdot|\Sigma|\cdot|\mathcal{C}|\cdot\vartheta_n)$, saving a factor of $\mathcal{O}(|\mathcal{C}|)$.*

3. *Using the doubling technique, the distributions can be computed using $\mathcal{O}(\log n\cdot|\mathcal{Q}^D|^3\cdot|\mathcal{C}|^3\cdot\vartheta_n^3)$ operations and $\mathcal{O}(|\mathcal{Q}^D|^2\cdot|\mathcal{C}|^2\cdot\vartheta_n^2)$ space.*

Proof.

1. From Lemma 2.24, we obtain bounds for time and space complexity of $\mathcal{O}(n\cdot|\mathcal{Q}|^2\cdot\vartheta_n\cdot|\mathcal{E}|)$ and $\mathcal{O}(|\mathcal{Q}|\cdot\vartheta_n)$, respectively. By construction, $|\mathcal{Q}|\le|\mathcal{Q}^D|\cdot|\mathcal{C}|$. Recall that Lemma 2.24 is based on Algorithm 2.1. The loops in lines 2 and 4 together account for a factor of $\mathcal{O}(n\cdot|\mathcal{Q}|\cdot\vartheta_n)$ in the time complexity. A factor of $\mathcal{O}(|\mathcal{Q}|\cdot|\mathcal{E}|)$ is caused by the inner loop in line 5. However, since the constructed PAA has deterministic (i.e. Dirac distributed) emissions, we do not need to iterate over all $e\in\mathcal{E}$ and save a factor of $|\mathcal{E}|$. Furthermore, we only need to iterate over all states reachable in one step. For each $q\in\mathcal{Q}$, there exist at most $|\Sigma|\cdot|\mathcal{C}|$ such states by construction of the PAA. Therefore, the inner loop in line 5 can be modified to use $\mathcal{O}(|\Sigma|\cdot|\mathcal{C}|)$ operations, yielding the claimed runtime bound.

2. If for all $c \in \mathcal{C}$ and $\sigma \in \Sigma$, there exists at most one $c' \in \mathcal{C}$ such that $\varphi(c, \sigma, c') > 0$, then at most $|\Sigma|$ different states are reachable from each state $q \in \mathcal{Q}$. The claimed runtime follows by the same arguments as above.

3. Alternative time and space complexities for the doubling algorithm follow directly from Lemma 2.26.

□

2.6 Pattern Matching Statistics

In this section, we ask for the distribution of the number of occurrences of a given pattern in a random text. Recall that the statistical significance of a motif is, by definition, the tail probability of this distribution: Assume we observe k motif instances in a given sequence. The significance is now defined as the probability of seeing k or more instances in a random text of the same length.

Different motif models have been discussed in Chapter 1. For all of these models, a motif can be seen as a finite set of all strings that are valid motif instances. Thus, a motif may be expressed in the form of a DFA that accepts the respective string set. The question of an efficient construction of such DFAs for different kinds of motifs is addressed in detail in Chapter 3. Here, we assume a DFA to be given and show how the distribution of the number of occurrences can be calculated.

The topic of statistics of words on random texts has previously been studied extensively. An overview is provided in the book by Lothaire (2005). Its Chapter 6 ("Statistics on Words with Applications to Biological Sequences") is based on the overview article by Reinert et al. (2000). In many approaches, a generating function is derived for the sought quantity. Then, typically using symbolic Taylor expansion, the concrete values can be computed. This procedure is, for instance, described by Régnier (2000), who gives formulas for mean, variance and higher statistical moments of the exact occurrence count distribution. This approach has the advantage of additionally allowing asymptotic analysis. Her framework is general enough to admit Markovian text models as well as finite sets of patterns to be treated. Closely related is the approach of Nicodème et al. (2002), who present an algorithm to compute the distribution of the number of times a DFA visits an accepting state when reading a random text. This is equivalent to computing the distribution of the number of occurrences when no two strings in the language accepted by the DFA can end at the same position, that is, no string is a proper suffix of another string in the language. The field is reviewed by Lladser et al. (2008). They describe the relation between finite automata and Markov chains in terms of what they call *Markov chain embedding*. Related approaches to compute the exact p-values based on automata are developed by Boeva et al. (2007) and Nuel (2008). Another dynamic programming approach has been given by Zhang et al. (2007). It is used to compute exact p-values for position weight matrices describing transcription factor binding sites (TFBS). The basic idea of using automata to compute pattern statistics has been briefly described and used for enumeration-based motif discovery by Tompa (1999).

The approach developed here provides a unifying framework for the efficient computation of pattern matching statistics. In contrast to existing approaches, it is applicable to arbitrary finite-memory text models. That means even character-emitting HMMs can be used as background models.

Constructing DAAs from DFAs

To use a DFA for pattern matching, that is, to find all instances of a pattern in a (long) text, an automaton that accepts not only all strings matching the pattern, but all strings that have a suffix matching the pattern is needed. This is discussed in detail in Chapter 3. Here, we assume such an automaton to be given.

When a DFA reads a text, it is in an accepting state whenever (at least) one instance of the pattern ends. To count the number of times a DFA $(\mathcal{Q}, \Sigma, \delta, q_0, F)$ is in an accepting state, we generalize it to a counting DFA $(\mathcal{Q}, \Sigma, \delta, q_0, (\eta_q)_{q\in\mathcal{Q}})$ as introduced in Definition 2.3 by setting $\eta_q := [\![q \in F]\!]$ for all $q \in \mathcal{Q}$.

As discussed in Example 2.4, in an arbitrary finite set of strings, some strings can be proper suffixes of others. As a consequence, more than one match can end at a position in the text. For DFAs based on such pattern sets, we need to count more than one match when entering the respective accepting state. We achieve this by defining emissions $(\eta_q)_{q\in\mathcal{Q}}$, where $\eta_q \in \mathbb{N}_0$ gives the number of matches to be counted upon entering state q.

Based on this counting DFA, we now construct a DAA that sums up the emissions generated in each state and turn this DAA into a PAA using Lemma 2.35. For practical computations, it is often sufficient to truncate the summed emissions at a constant $M \in \mathbb{N}$. The proof of the following theorem contains the details of the DAA construction.

Theorem 2.38. *Let a counting DFA $(\mathcal{Q}, \Sigma, \delta, q_0, (\eta_q)_{q\in\mathcal{Q}})$ and a text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ be given, and let $(S_t)_{t\in\mathbb{N}_0}$ be a random text distributed according to that model. Then, the (truncated) distribution of accumulated counts*

$$\mathcal{L}\left(\min\left\{M\,,\,\sum_{i=0}^{n-1}\eta_{\delta(q_0,S_0\cdots S_i)}\right\}\right) \tag{2.31}$$

can be computed using $\mathcal{O}(n\cdot|\mathcal{Q}|\cdot|\Sigma|\cdot|\mathcal{C}|^2\cdot M)$ operations and $\mathcal{O}(|\mathcal{Q}|\cdot|\mathcal{C}|\cdot M)$ space. If for all $c\in\mathcal{C}$ and $\sigma\in\Sigma$, there exists at most one $c'\in\mathcal{C}$ such that $\varphi(c,\sigma,c')>0$, then the number of required operations is bounded by $\mathcal{O}(n\cdot|\mathcal{Q}|\cdot|\Sigma|\cdot|\mathcal{C}|\cdot M)$. Alternatively, (2.31) *can be computed with $\mathcal{O}(\log n\cdot|\mathcal{Q}|^3\cdot|\mathcal{C}|^3\cdot M^2)$ operations and $\mathcal{O}(|\mathcal{Q}|^2\cdot|\mathcal{C}|^2\cdot M)$ space.*

Proof. Given the counting DFA $(\mathcal{Q}, \Sigma, \delta, q_0, (\eta_q)_{q\in\mathcal{Q}})$, we use it to construct the DAA $D = \big(\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}}\big)$, where $\mathcal{V} := \{0,\dots,M\}$, $v_0 := 0$, $\mathcal{E} := \{\eta_q : q\in\mathcal{Q}\}$, and all operations are truncated additions:

$$\theta_q(v,e) := \min\{M, v+e\}$$

for all $q\in\mathcal{Q}$. Clearly, we have

$$\mathit{value}_D(s) = \min\left\{M\,,\,\sum_{i=0}^{n-1}\eta_{\delta(q_0,s[..i])}\right\}$$

for all $s \in \Sigma^*$.

We now apply Lemma 2.35 to the DAA and the text model in order to construct a PAA. The runtime and space bounds for the basic algorithm follow directly from Lemma 2.37. To obtain the bounds for the alternative doubling algorithm, namely $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3 \cdot |\mathcal{C}|^3 \cdot M^2)$ operations and $\mathcal{O}(|\mathcal{Q}|^2 \cdot |\mathcal{C}|^2 \cdot M)$ space, we exploit that the operations θ_q are (almost) additions in this case. Thus, $U^{(t)}(q_1, q_2, v_1, v_2) = U^{(t)}(q_1, q_2, v_3, v_4)$ if $v_1 \leq v_2 < M$, $v_3 \leq v_4 < M$ and $v_2 - v_1 = v_4 - v_3$. Thus, we can fix $v_1 = 0$ and thereby save a factor of $|\mathcal{V}| = M + 1$ in time and space. The special cases for $v_2 = M$ or $v_4 = M$ can be accommodated for in the same bounds. □

Remark 2.39 (DFA minimization)**.** Before turning the DFA into a PAA, one may wish to minimize it. Using an algorithm by Hopcroft (1971), a classical DFA can be minimized in $\mathcal{O}(|\mathcal{Q}| \log |\mathcal{Q}|)$ time for an alphabet of constant size, where $\mathcal{Q}$ is the set of states. It does so by iteratively refining a partition of the state set. In the beginning, all states are partitioned into two distinct sets: one containing the accepting states, and the other containing the non-accepting states. This partition is iteratively refined whenever a reason for non-equivalence of two states in the same set is found. Upon termination, the states are partitioned into sets of equivalent states. Refer to Knuutila (2001) for an in-depth explanation of Hopcroft's algorithm. He also gives a variant that runs in $\mathcal{O}(|\Sigma| \cdot |\mathcal{Q}| \log |\mathcal{Q}|)$ time when the alphabet size is not considered to be a constant. Hopcroft's algorithm can be adapted to minimize counting DFAs by using the partition induced by the different emissions as an initial partition, that means, states with the same emission are grouped together. For specific pattern classes, minimal DFAs can be constructed directly, that is, without the intermediate step of a non-minimal DFA. Algorithms for this are presented in Chapter 3. △

Example 2.40 (Significance of a motif)**.** Suppose we have observed the IUPAC pattern `ANAG` ten times in an stretch of DNA of length 100. Now we want to compute the significance of this event, that is, the probability that ten or more matches of `ANAG` are found in a random text of length 100. As a background model, we use a first order Markovian model estimated from the human genome as shown in Figure 2.5a. Note that the shown text model does not start in equilibrium. If necessary, this can be changed by adding a new start state according to Remark 2.19. The next step is the construction of a DFA that recognizes the motif. More precisely, we build a DFA that accepts all strings with a suffix matching the motif. This DFA is depicted in Figure 2.5b. Then, the DFA is translated into a DAA and subsequently into a PAA as formally described in the proof of Theorem 2.38. The resulting PAA is shown in Figure 2.5c. As the text model has four and the DFA has seven states, the PAA could have up to 28 ($= 4 \cdot 7$) states. The figure illustrates that in this case, only twelve states of those are reachable (unreachable states are not shown). Using Algorithm 2.1, we compute its state-value distribution $\mathcal{L}(Q_n, V_n)$ for $n = 100$, marginalize to obtain $\mathcal{L}(V_n)$, and read off the desired p-value which amounts to $4.3 \cdot 10^{-5}$. △

Example 2.41 (Mapping short reads to the genome)**.** Current DNA sequencing technologies produce reads at the scale of gigabases per day. Mapping reads efficiently

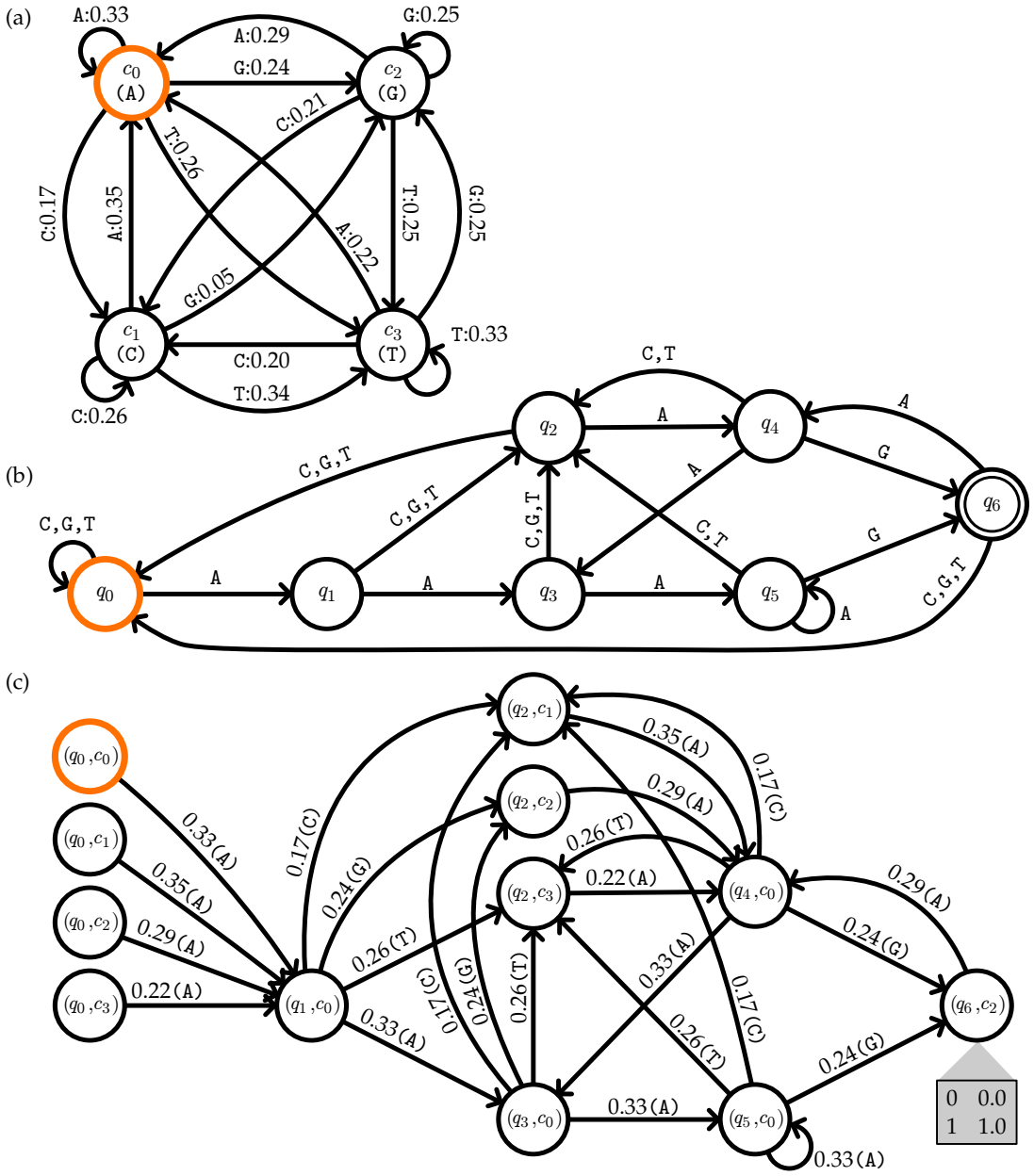


Figure 2.5: Construction of a PAA for pattern matching statistics over the alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Start states are drawn in orange. **(a)** Markovian text model of order one estimated from the human genome. **(b)** Minimal DFA accepting all strings whose suffix matches `ANAG`. **(c)** PAA resulting from applying Theorem 2.38. For clarity, edges with zero probability and those to states $(q_0, \cdot)$ have been omitted. All operations are additions (not drawn). Emission distributions are omitted for states that emit a zero with probability one. Characters near edges (in parentheses) are not part of the PAA and are given only to aid comparison to the DFA.

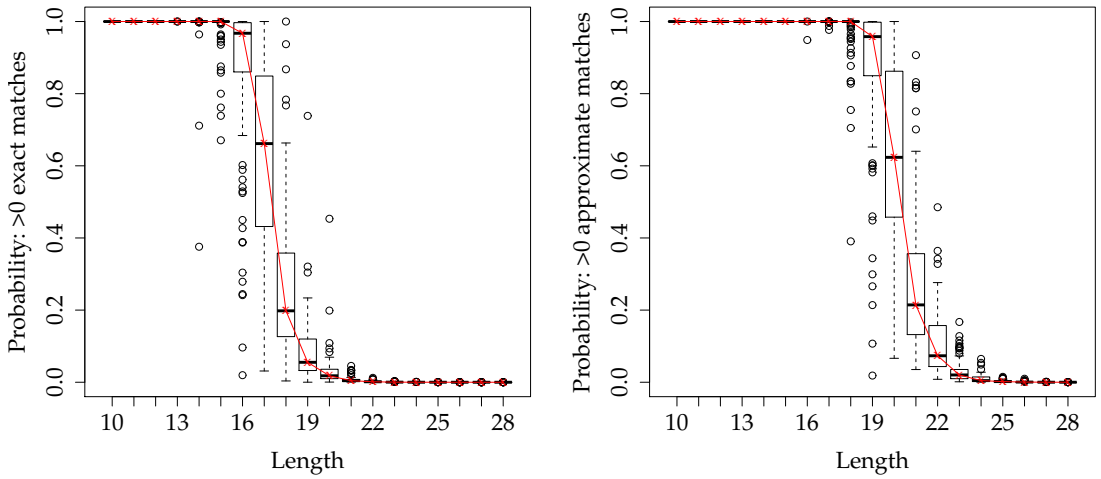


Figure 2.6: Both figures show boxplots of the probabilities that randomly sampled k-mers are contained at least once in a random genome (where k is varied on the x-axis). Red crosses mark mean probabilities. The experiments are further explained in Example 2.41. **Left:** only exact matches are counted. **Right:** up to one mismatch is allowed, i.e. all occurrences with Hamming distance of at most one are counted as matches.

and accurately to a reference genome is therefore an important yet difficult task. The question of whether a k-mer (for a given value of k) is likely to match the human genome just by chance is important for many seed-and-extend approaches to read mapping. For a given k-mer, this probability can be computed using PAAs. Here, the value space $\mathcal{V}$ is small as we only have to distinguish the two values "no match" and "one or more matches". On the other hand, the number of steps n is large as the human genome comprises more than three billion nucleotides. Therefore, the *doubling algorithm* introduced in Section 2.4.2 is advantageous. We use a second order Markovian background model and sample 100 k-mers for each k between 10 and 28. For these k-mers, the probability that a random text of the same length as the human genome contains at least one match (either exactly or with up to Hamming distance one) is computed with respect to the same text model. The results are shown in Figure 2.6. This figure illustrates that the probabilities indeed depend on the k-mer. For $k = 17$, for instance, the probability of finding one or more exact matches (left boxplot) is particularly diverse and ranges from 0.031 (for `AAGCGCATCGTGAAAGA`) to 0.999 (for `AAAATCATAAATAAAAA`). Furthermore, the plots show that there is a rather sharp transition from high to low probabilities: when considering exact matches (left boxplot), choosing $k = 15$ leads to a high probability for random matches, while for $k = 21$, random matches are unlikely. △

Remark 2.42 (Waiting time for pattern occurrences)**.** The waiting time for the first occurrence of a pattern equals the waiting time for a state that emits a match. Therefore,

its distribution can directly be computed by applying Lemma 2.30 for

$$\mathcal{U} := \left\{ (q^D, c) \in \mathcal{Q} \,\middle|\, \eta_{q^D} > 0 \right\}.$$

The (asymptotic) waiting time for a subsequent occurrence as defined by Equation (2.21) can be computed by choosing α in Lemma 2.30 to be the equilibrium state distribution restricted to $\mathcal{U}$. △

2.7 Excursus: Analysis of Pattern Matching Algorithms

The basic pattern matching problem is to find all exact occurrences of a *pattern* $p \in \Sigma^*$ in a (long) *text* $s \in \Sigma^*$ with few character accesses. Many algorithms have been invented to solve this problem. As a further example for the use of DAAs and PAAs, we compute the distribution of the number of character accesses needed by an arbitrary window-based algorithm to search for a fixed pattern in a random text.

Let $X_n^{A,p}$ be a random variable giving the number of character accesses made by algorithm A to search for the pattern p in the random text $S_0 \cdots S_{n-1}$. Despite all research on pattern matching algorithms, a procedure to exactly compute the distribution $\mathcal{L}\big(X_n^{A,p}\big)$ has, to our knowledge, not been given before. However, some related questions have been answered in the literature. Baeza-Yates et al. (1990) and Baeza-Yates and Régnier (1992), for example, analyze the expected value of $X_n^{A,p}$ for Horspool's algorithm. Mahmoud et al. (1997) further show that for Horspool's algorithm, $X_n^{A,p}$ is asymptotically normally distributed for i.i.d. texts, and Smythe (2001) extends this result to Markovian text models. Tsai (2006) devises a method to compute mean and variance of these distributions.

In contrast to these results that are special to Horspool's algorithm and tied to special types of text models, PAAs can be used to analyze arbitrary window-based pattern matching algorithms with respect to arbitrary finite-memory text models. After a brief survey of pattern matching algorithms in Section 2.7.1, we give a generic method to construct a DAA that counts the number of character accesses incurred by a given algorithm in Section 2.7.2. In Section 2.7.3, we apply the techniques developed in Sections 2.4 and 2.5 to combine this DAA with a text model and compute the sought distribution of the number of character accesses. In Section 2.7.4, methods to compare two algorithms are introduced. We conclude this digression with Section 2.7.5, where we study three pattern matching algorithms in the non-asymptotic regime (short patterns, medium-length texts).

2.7.1 Pattern Matching Algorithms

Let n be the text length and ℓ be the pattern length. The well-known Knuth-Morris-Pratt algorithm (Knuth et al., 1977) reads each text character exactly once from left to right. After preprocessing the pattern in $\Theta(\ell)$ time, it hence needs exactly n character accesses for any text of length n. In contrast, the algorithms by Boyer and Moore (1977), Horspool (1980), Sunday (1990), Crochemore et al. (1994, Backward DAWG Matching, BDM), and Allauzen et al. (2001, Backward Oracle Matching, BOM) move a length-ℓ

search window across the text and first compare its *last* character to the last character of the pattern. This often allows moving the search window by more than one position. At best, a shift of ℓ positions can be made if the last window character does not occur in the pattern at all. This leads to n/ℓ character accesses in the best case, but ℓn character accesses in the worst case. As done in the Boyer-Moore algorithm, the worst case can often be improved to $\Theta(\ell + n)$, but this makes the code more complicated and seldom provides a speed-up in practice.

For practical pattern matching applications, the most important algorithms are Horspool's algorithm, BDM, and BOM, depending on alphabet size, text length and pattern length. An experimental map is provided by Navarro and Raffinot (2002, Section 2.5). BDM is often implemented as Backward Nondeterministic DAWG Matching (BNDM), via a non-deterministic automaton that is simulated in a bit-parallel fashion. To measure costs incurred by an algorithm, we count the number of character accesses. We do not treat BDM and BNDM separately as they are indistinguishable in terms of text character accesses. In the following, we summarize the three algorithms. More details can be found in the book by Navarro and Raffinot (2002, Chapter 2).

All three algorithms have the following properties: They maintain a search window w of length $\ell = |p|$ that is initially placed at position 0 in the text s so that its rightmost character is at position $t = \ell - 1$. The right window position t increases in the course of the algorithm; we always have $w = s[(t - \ell + 1) \dots t]$. The algorithms look at the characters in each window from right to left, and thus compare the reversed window with the reversed pattern. For Horspool's algorithm, variants with different comparison orders are possible, but the rightmost character is always compared first.

The number of character accesses done by an algorithm are determined by the following two functions that are different for the three algorithms. For a pattern $p \in \Sigma^\ell$, each window $w \in \Sigma^\ell$ determines

1. its cost $\xi^{A,p}(w)$, e.g., the number of text character accesses required to analyze this window, and

2. its shift, written $\mathit{shift}^{A,p}(w)$, which is the number of characters the window is advanced after it has been examined.

If the algorithm referred to is clear from the context, we omit the superscript A. In the following, we review Horspool's algorithm, B(N)DM, and BOM and give their cost function and their shift function.

Horspool's Algorithm

First, the rightmost characters of window and pattern are compared. That means $\sigma := w[\ell - 1] = s[t]$ is compared with $p[\ell - 1]$. If they match, the remaining $\ell - 1$ characters are compared until either the first mismatch is found or an entire match has been verified. This comparison can happen right-to-left, left-to-right, or in an arbitrary order that may depend on p. In our analysis, we focus on the right-to-left case, but the modifications for the other cases are straightforward. Therefore, the cost of window w

Algorithm 2.2 Horspool's algorithm to compute the number of occurrences of p in s.

Input: Pattern $p \in \Sigma^*$, text $s \in \Sigma^*$
Output: Number of occurrences (*occ*) and number of incurred character accesses (*cost*)

HORSPOOL-SEARCH(s, p)

```
1  precompute shift-by-lastchar^p(σ) for all σ ∈ Σ
2  (occ, cost) = (0, 0)
3  t = ℓ − 1
4  while t < |s|
5      i = 0
6      while i < ℓ
7          cost = cost + 1
8          if s[t − i] ≠ p[(ℓ − 1) − i]
9              break
10         i = i + 1
11     if i = ℓ
12         occ = occ + 1
13     t = t + shift-by-lastchar^p(s[t])
14 return (occ, cost)
```

is

$$\xi^p(w) = \begin{cases} \ell & \text{if } p = w, \\ \min\left\{ i \in \{1, \ldots, \ell\} \,\middle|\, p[\ell - i] \neq w[\ell - i] \right\} & \text{otherwise.} \end{cases}$$

In any case, the rightmost window character σ is used to determine how far the window can be shifted for the next iteration. Thus, we define a function *shift-by-lastchar*^{p} that maps characters to the corresponding shift. It ensures that no match can be missed. It does so by moving the window such that σ becomes aligned to the rightmost σ in p (not considering the last position). If σ does not occur in p (or only at the last position), it is safe to shift by ℓ positions. Formally, we define

$$\begin{aligned} \textit{right}^p(\sigma) &:= \max\left[\left\{ i \in \{0, \ldots, \ell - 2\} \,\middle|\, p[i] = \sigma \right\} \cup \{-1\}\right], \\ \textit{shift-by-lastchar}^p(\sigma) &:= (\ell - 1) - \textit{right}^p(\sigma), \\ \textit{shift}^p(w) &:= \textit{shift-by-lastchar}^p(w[\ell - 1]). \end{aligned}$$

For concreteness, we state Horspool's algorithm and how we count text character accesses as pseudocode in Algorithm 2.2. Note that after a shift, even when we know that σ now matches its corresponding pattern character, the corresponding position is compared again and counts as a text access. Otherwise the additional bookkeeping would make the algorithm more complicated; this is not worth the effort in practice. We do not count the character access $s[t]$ in Line 13 as we can memorize $s[t]$ as soon as it has been read in Line 8.

The main advantage of this algorithm is its simplicity. Horspool's algorithm does not require any advanced data structure and can be implemented in a few lines of code.

Backward (Nondeterministic) DAWG Matching, B(N)DM

The main idea of the BDM algorithm is to use a suffix automaton, which is a *directed acyclic word graph* (DAWG), to determine the shift. A DAWG is a deterministic finite automaton that accepts all suffixes of the reversed pattern (including the empty suffix) and enters a special state q_{fail} if (and only if) a string has been read that is not a substring of the reversed pattern $\overleftarrow{p}$.

We process each window from right to left. As long as the state q_{fail} has not been reached, we have read a substring of the reversed pattern. If we are in an accepting state, we have even found a suffix of the reversed pattern (i.e., a prefix of p). Whenever this happens before we have read ℓ characters, the last such event marks the next potential window start that may contain a match with p, and hence determines the shift. When we are in an accepting state after reading ℓ characters, we have found a match, but this does not influence the shift.

So, $\xi^p(w)$ is the number of characters read when entering q_{fail}, or ℓ if $p = w$. Let $I^p(w)$ be a subset of $\{0, \ldots, \ell - 1\}$ such that $i \in I^p(w)$ if and only if the suffix automaton of $\overleftarrow{p}$ is in an accepting state after reading i characters of w. Then,

$$\mathit{shift}^p(w) = \min\left\{\ell - i \,\middle|\, i \in I^p(w)\right\}.$$

The set $I^p(w)$ is never empty as the suffix automaton accepts the empty string and, thus, $0 \in I^p(w)$ for all windows w.

The advantage of BDM are long shifts, but its main disadvantage is the necessary construction of the suffix automaton, which is possible in $\mathcal{O}(\ell)$ time via the suffix tree of the reversed pattern, but too expensive in practice to compete with other algorithms unless the search text is extremely long.

Constructing a nondeterministic finite automaton (NFA) instead of the deterministic suffix automaton is much simpler. However, processing a text character then does not take constant, but $\mathcal{O}(\ell)$ time. However, the NFA can be efficiently simulated with bit-parallel operations such that processing a text character takes $\mathcal{O}(\ell/W)$ time, where W is the machine word size. For many patterns in practice, this is as good as $\mathcal{O}(1)$. The resulting algorithm is called BNDM. In terms of text character accesses, BDM and BNDM are equivalent as they have the same shift and cost functions.

Backward Oracle Matching, BOM

BOM is similar to BDM, but the suffix automaton of the reversed pattern is replaced by a simpler deterministic automaton, the *factor oracle* (rarely also called *suffix oracle*). It accepts all factors of the reversed pattern but may also accept more strings. Tolerating that the automaton accepts additional strings allows a simpler construction.

The cost and shift functions are defined as for BDM, but based on the oracle. We refer to Navarro and Raffinot (2002) for the construction details and further properties of the oracle. By construction, BOM never gives a longer shift than B(N)DM for any window w. The main advantage of BOM over BDM is reduced space usage and preprocessing time as the factor oracle only has $\ell + 1$ states and can be constructed faster than a suffix automaton.

2.7.2 Constructing DAAs

For a given algorithm A with known shift and cost functions and a pattern $p \in \Sigma^\ell$, we construct a DAA that, upon reading a text $s \in \Sigma^*$, computes the total cost, defined as the sum of costs over all examined windows. Which windows are examined depends on the shift values of previously examined windows. Extending notation, we write $\xi^{A,p}(s)$ for the total cost incurred on $s \in \Sigma^*$.

Definition 2.43 (Cost-computing DAA)**.** Let a window-based pattern matching algorithm A and a pattern $p \in \Sigma^\ell$ be given and the algorithm's shift and cost functions *shift* $^{A,p} : \Sigma^\ell \to \{1, \ldots, \ell\}$ and $\xi^{A,p} : \Sigma^\ell \to \mathbb{N}$ be known. We define *CostDAA*$(A, p) = \big(\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}}\big)$ by

- $\mathcal{Q} := \Sigma^\ell \times \{0, \ldots, \ell\}$,
- $q_0 := (p, \ell)$,
- $\delta : \big((w, x), \sigma\big) \mapsto \begin{cases} (w[1..]\sigma,\ x - 1) & \text{if } x > 0, \\ (w[1..]\sigma,\ \textit{shift}^{\,A,p}(w) - 1) & \text{if } x = 0, \end{cases}$
- $\mathcal{V} := \mathbb{N}$,
- $v_0 := 0$,
- $\mathcal{E} := \{1, \ldots, \ell\}$,
- $\eta_{(w,x)} := \begin{cases} 0 & \text{if } x > 0, \\ \xi^{A,p}(w) & \text{if } x = 0, \end{cases}$
- $\theta_q : (v, e) \mapsto v + e$ for all $q \in \mathcal{Q}$.

$\Diamond$

Informally, the state $q = (w, x)$ means that the last ℓ read characters spell w and that x more characters need to be read to get to the end of the current window. For the start state (p, ℓ), the component p is arbitrary, as we need to read ℓ characters to reach the end of the first window. The value accumulates the cost of examined windows. Therefore, the operation is a simple addition in each state, and the emission of state (w, x) specifies the cost to add. Consequently, the emission is zero if the state does not correspond to an examined window ($x > 0$), and the emission equals the window cost $\xi^p(w)$ if $x = 0$. The transition function δ specifies how to move from one state to the next when reading the next text character $\sigma \in \Sigma$: In any case, the window content is updated by forgetting the first character and appending the read σ. If the end of the current window has not been reached ($x > 0$), the counter x is decremented.

Otherwise, the window's shift value is used to compute the number of characters till the next window aligns.

Theorem 2.44. *Let a window-based pattern matching algorithm A and a pattern $p \in \Sigma^\ell$ be given and $D := \mathit{CostDAA}(A, p)$ be constructed according to Definition 2.43. Then,*

$$\mathit{value}_D(s) = \xi^{A,p}(s)$$

for all $s \in \Sigma^*$.

Proof. The total cost $\xi^{A,p}(s)$ can be written as the sum of costs of all processed windows:

$$\xi^{A,p}(s) = \sum_{i \in I_s} \xi^{A,p}(s[i-\ell+1 \ldots i])\,,$$

where I_s is the set of indices giving the processed windows, i.e. $I_s \subset \{\ell-1, \ldots, |s|-1\}$ such that

$$i \in I_s \quad :\Longleftrightarrow \quad i = \ell - 1 \quad \text{or} \quad \exists j \in I_s : i = j + \mathit{shift}^{A,p}(s[j-\ell+1 \ldots j]).$$

We have to prove that the DAA computes this value for $s \in \Sigma^*$.

Let (w_i, x_i) be the DAA state active after reading $s[..i]$. Observe that the transition function δ ensures that the w_i-component of (w_i, x_i) reflects the rightmost length-ℓ window of $s[..i]$, which can immediately be verified inductively. Thus, the emission on reading the last character $s[i]$ of $s[..i]$ with $i \geq \ell - 1$ is, by definition of $\eta_{(w_i,x_i)}$, either $\xi^{A,p}(s[i-\ell+1 \ldots i])$ or zero, depending on the second component of (w_i, x_i). As the operation is an addition for all states, we get

$$\mathit{value}_D(s) = \sum_{i \in I'_s} \xi^{A,p}(s[i-\ell+1 \ldots i])$$

for

$$I'_s := \left\{ i \in \{0, \ldots, |s|-1\} \,\middle|\, x_i = 0 \right\}.$$

It remains to be shown that $I_s = I'_s$. By δ, we have $x_{i+1} = x_i - 1$ if $x_{i+1} > 0$ and $x_{i+1} = \mathit{shift}^{A,p}(w_i) - 1$ if $x_{i+1} = 0$. As $q_0 = (p, \ell)$, it follows that $\ell - 1 \in I'_s$. Using $w_i = s[i-\ell+1 \ldots i]$ for $i \geq \ell - 1$, we conclude that whenever $x_i = 0$, it follows that $x_j = 0$ for $j = i + \mathit{shift}^{A,p}(s[i-\ell+1 \ldots i])$ and that $x_{j'} > 0$ for $i < j' < j$. Hence we obtain that $i \in I'_s$ implies that $i + \mathit{shift}^{A,p}(s[i-\ell+1 \ldots i]) \in I'_s$ and $i + k \notin I'_s$ for $0 < k < \mathit{shift}^{A,p}(s[i-\ell+1 \ldots i])$, which completes the proof. □

DAA Minimization

The size of the constructed DAA's state space depends exponentially on the pattern length, making the application for long patterns infeasible. However, depending on the particular circumstances (i.e., algorithm and pattern analyzed), the constructed DAA can often be substantially reduced by applying a modified version of Hopcroft's algorithm for DFA minimization (Hopcroft, 1971) which is explained in Remark 2.39.

The algorithm can straightforwardly be adapted to minimize DAAs by choosing the initial state set partition appropriately. In our case, all DAA states are associated with the same operation. The only differences in the behavior of states thus stem from different emissions. Therefore, Hopcroft's algorithm can be initialized by the partition induced by the emissions and then continued as usual. As exemplified in Section 2.7.5, this leads to a considerable reduction of the number of states.

2.7.3 Computing Cost Distributions

Having constructed a DAA that computes the cost, in our case the number of character accesses caused by an algorithm, we can make use of the theory developed in Section 2.5 and directly arrive at the following theorem.

Theorem 2.45. *Let a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$, a window-based pattern matching algorithm A, a pattern $p \in \Sigma^\ell$, and the functions shift ^{A,p} and $\xi^{A,p}$ be given. Furthermore, let $\mathcal{Q}^D$ be the state set (possibly after minimization) of the corresponding cost-computing DAA as given in Definition 2.43. Then, the cost distribution $\mathcal{L}(X_n^{A,p})$ can be computed using $\mathcal{O}(n^2 \cdot \ell \cdot |\mathcal{Q}^D| \cdot |\Sigma| \cdot |\mathcal{C}|^2)$ operations and $\mathcal{O}(|\mathcal{Q}^D| \cdot |\mathcal{C}| \cdot n \cdot \ell)$ space. Since $|\mathcal{Q}^D|$ is bounded by $\mathcal{O}(\ell \cdot |\Sigma|^\ell)$, the computation uses $\mathcal{O}(n^2 \cdot \ell^2 \cdot |\Sigma|^{\ell+1} \cdot |\mathcal{C}|^2)$ operations and $\mathcal{O}(\ell^2 \cdot |\Sigma|^\ell \cdot |\mathcal{C}| \cdot n)$ space. If for all $c \in \mathcal{C}$ and $\sigma \in \Sigma$, there exists at most one $c' \in \mathcal{C}$ such that $\varphi(c, \sigma, c') > 0$, a factor of $|\mathcal{C}|$ can be dropped from the runtime bounds.*

Proof. The claim follows by observing that $\vartheta_n \in \mathcal{O}(n\ell)$ and combining Theorem 2.44, Lemma 2.35, and Lemma 2.37. □

Applying DAA minimization before the PAA construction results in considerable speed-ups in practice. Alternatively, algorithm-dependent construction schemes may be used to construct small automata. Tsai (2006), for instance, gives algorithms to compute the asymptotic mean and variance of the number of comparisons used by Horspool's algorithm; for that, he constructs a Markov chain with $\mathcal{O}(\ell^2)$ states. His construction can immediately be adapted to construct a DAA with $\mathcal{O}(\ell^2)$ states. However, it cannot be easily transferred to other algorithms and we are not aware of any similar results for BOM or B(N)DM. It remains an open question whether there exists an algorithm to construct the minimal automaton directly in general, i.e. using only $\mathcal{O}(|\mathcal{Q}^D_{min}|)$ time.

2.7.4 Comparing Algorithms

Computing the cost distribution for two algorithms allows us to compare their performance characteristics. One natural question, however, cannot be answered by comparing these two (one-dimensional) distributions: What is the probability that algorithm A needs more text accesses than algorithm B to scan the same random text? The answer will depend on the correlation of algorithm performances: Do the same instances lead to long runtimes for both algorithms or are there instances that are easy for one algorithm but difficult for the other? This section answers these questions by constructing a PAA to compute the distribution of *cost differences* of two algorithms.

That means we calculate the probability that algorithm A_1 needs v text accesses *more* than algorithm A_2 for all $v \in \mathbb{Z}$.

We start by giving a general construction of a DAA that computes the difference of the sum of emission of two given DAAs.

Definition 2.46 (Difference DAA)**.** Let $D^1 = \left(\mathcal{Q}^1, q_0^1, \Sigma, \delta^1, \mathcal{V}^1, v_0^1, \mathcal{E}^1, (\eta_q^1)_{q\in\mathcal{Q}^1}, (\theta_q^1)_{q\in\mathcal{Q}^1}\right)$ and $D^2 = \left(\mathcal{Q}^2, q_0^2, \Sigma, \delta^2, \mathcal{V}^2, v_0^2, \mathcal{E}^2, (\eta_q^2)_{q\in\mathcal{Q}^2}, (\theta_q^2)_{q\in\mathcal{Q}^2}\right)$ be DAAs given over the same alphabet Σ with $\mathcal{V}^1 = \mathcal{V}^2 = \mathbb{N}$, $v_0^1 = v_0^2 = 0$, $\mathcal{E}^1, \mathcal{E}^2 \subset \mathbb{N}$, and all operations are additions of the previous value and the current emission. The *difference DAA* is defined as

$$\mathit{DiffDAA}(D^1, D^2) := (\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}})$$

where

- $\mathcal{Q} := \mathcal{Q}^1 \times \mathcal{Q}^2$ and $q_0 := (q_0^1, q_0^2)$,
- $\mathcal{V} := \mathbb{Z}$ and $v_0 := 0$,
- $\mathcal{E} := \mathcal{E}^1 \times \mathcal{E}^2$ and $\eta_{(q^1,q^2)} := \left(\eta_{q^1}^1, \eta_{q^2}^2\right)$,
- $\delta : \left((q^1, q^2), \sigma\right) \mapsto \left(\delta^1(q^1, \sigma), \delta^2(q^2, \sigma)\right)$,
- $\theta_q : \left(v, (e^1, e^2)\right) \mapsto v + e^1 - e^2$.

$\Diamond$

Lemma 2.47. *Let D^1 and D^2 be DAAs meeting the criteria given in Definition 2.46 and $D := \mathit{DiffDAA}(D^1, D^2)$. Then,*

$$\mathit{value}_D(s) = \mathit{value}_{D^1}(s) - \mathit{value}_{D^2}(s)$$

for all $s \in \Sigma^*$.

Proof. Follows directly from Definition 2.46. □

Lemma 2.47 can now be applied to the DAAs constructed according to Definition 2.43 for the analysis of two algorithms. Since the construction in Definition 2.46 builds the product of both state spaces, it is advisable to minimize both DAAs before generating the product. Furthermore, in an implementation, only reachable states of the product automaton need to be constructed. Before being used to build a PAA (by applying Lemma 2.35), the difference DAA should again be minimized.

At most $\ell(n - \ell + 1)$ character accesses can result from scanning a text of length n for a pattern of length ℓ. Thus, the difference of costs for two algorithms lies in the range $\{-\ell(n - \ell + 1), \ldots, \ell(n - \ell + 1)\}$ and, hence, $\vartheta_n \in \mathcal{O}(n \cdot \ell)$.

2.7.5 Case Studies

Above, we described three practically relevant algorithms, namely Horspool's algorithm, backward oracle matching (BOM), and backward (non)-deterministic DAWG matching (B(N)DM). Now, we compare the distributions of running time costs of these algorithms for several patterns. Figure 2.7 shows these distributions for the

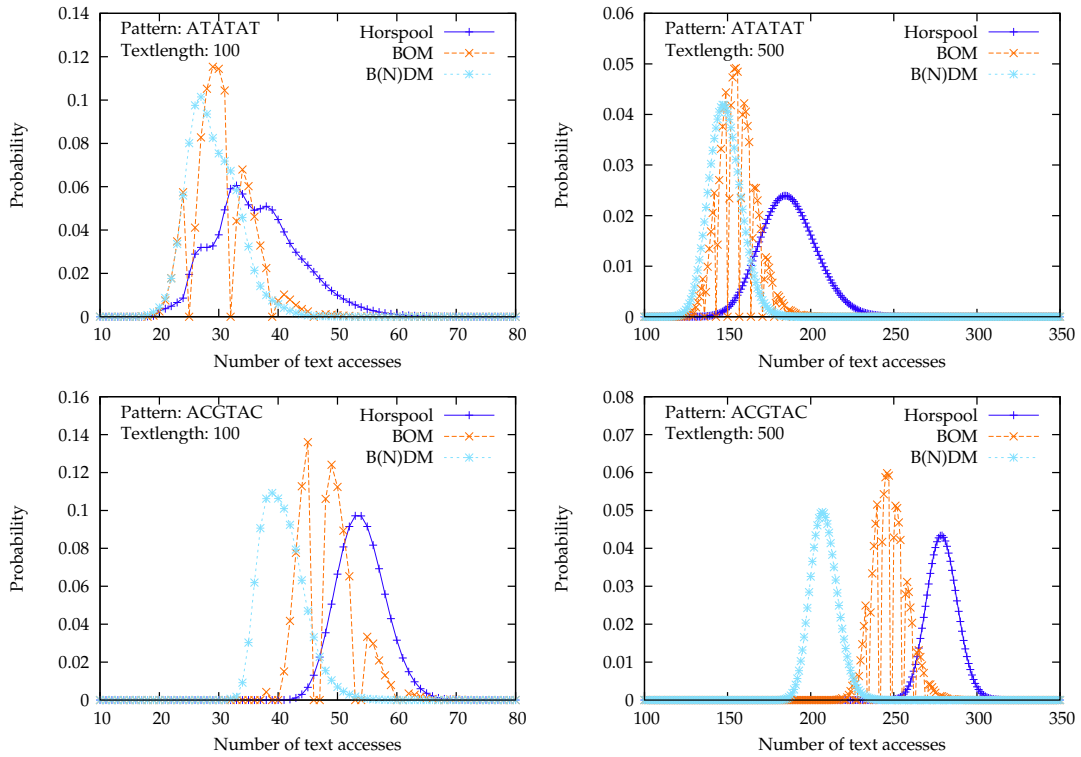


Figure 2.7: Exact distributions of character access counts for patterns `ATATAT` (top) and `ACGTAC` (bottom) for text length 100 (left) and text length 500 (right). An i.i.d. text model with uniform character distribution is used.

patterns `ATATAT` and `ACGTAC` for text lengths 100 and 500 under a uniform i.i.d. model on the DNA alphabet {`A`,`C`,`G`,`T`}. For text length 500, the distributions for Horspool's algorithm and B(N)DM resemble the shape of normal distributions. In fact, for Horspool's algorithm, Smythe (2001) has proved that the distribution is asymptotically normal. For smaller text lengths (e.g. 100, as shown in left column of Figure 2.7), the distributions are less smooth than for longer texts. It is remarkable that for BOM we find zero probabilities with a fixed period. In all examples shown in Figure 2.7, this period equals seven.

The probability that one pattern matching algorithm is faster than another depends on the pattern. Using the technique introduced in Section 2.7.4, we can quantify the strength of this effect. Figure 2.8 shows distributions of cost *differences* for different patterns and algorithms. That means the probability that the first algorithm is faster is represented by the area under the curve left of zero. For the pattern `CGAAAA`, for example, there is a 55.6% probability that Horspool's algorithm needs fewer character accesses than B(N)DM in uniform i.i.d. texts of length 100, while for `ACGTAC`, the probability is only 0.18%.

Worth noting and perhaps surprising is the fact that there is a non-zero probability

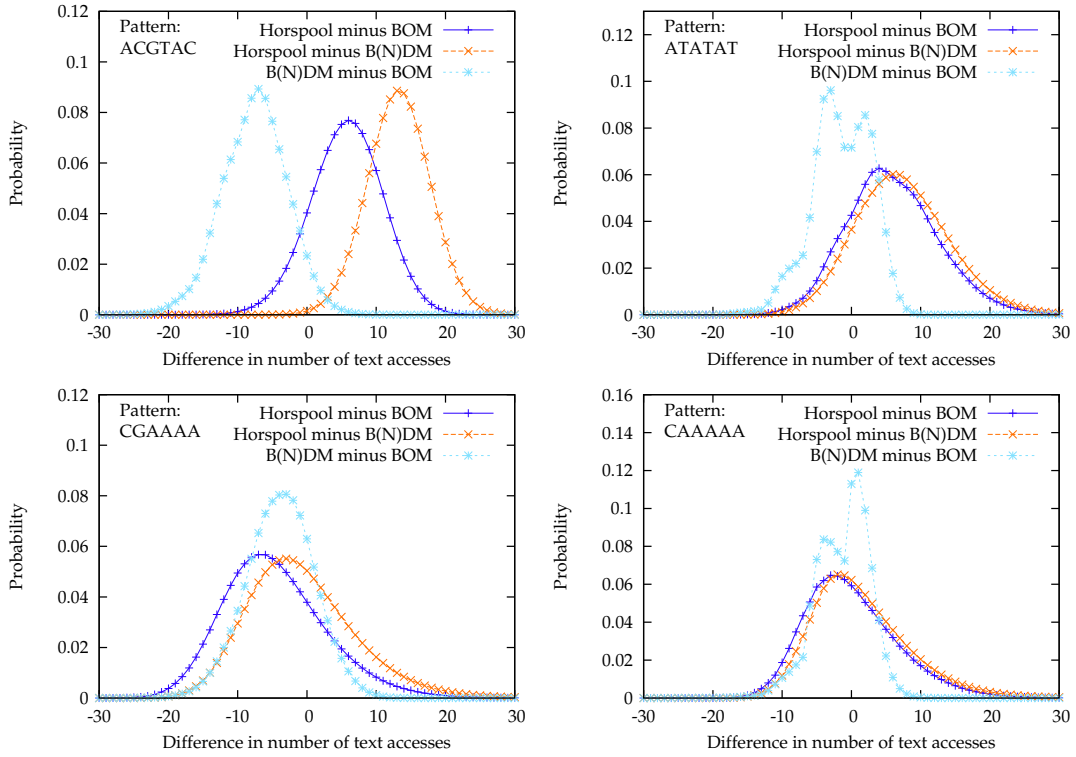


Figure 2.8: Exact distributions of differences in character access counts for different patterns using a uniform character distribution as text model and random texts of lengths 100.

of BOM being faster than B(N)DM although $shift^{B(N)DM,p}(w) \geq shift^{BOM,p}(w)$ for all window contents w. The explanation is that a shorter (say, first) shift for BOM leads to a different window content than for B(N)DM for the second window, which may have a larger shift value. This effect depends on the pattern: for the pattern `CAAAAA`, there is a 48.2% probability that BOM performs better than B(N)DM, while it is 6.2% for `ACGTAC`, again on texts of length 100. These results show that the algorithms' performance is indeed non-negligibly influenced by the pattern.

To assess the effect of DAA minimization before constructing PAAs, we constructed minimized DAAs for all 21 840 patterns of lengths two to seven over $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. The minimum, average, and maximum state counts are shown in Table 2.9. For length six, Figure 2.10 contains a detailed histogram of worst-case state counts of minimal automata. It may be conjectured that the minimal state counts grow only polynomially with ℓ for all of these algorithms, as has been previously proved for Horspool's algorithm by Tsai (2006).

Table 2.9: Comparison of DAA sizes for all patterns of length ℓ over $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$.

ℓ	States unminimized $\lvert\Sigma\rvert^\ell \cdot (\ell + 1)$	States minimized (min./avg./max.) Horspool	BOM	B(N)DM
2	48	4 / 4.8 / 5	4 / 4.0 / 4	4 / 4.8 / 5
3	256	7 / 8.3 / 9	7 / 8.3 / 9	7 / 9.6 / 10
4	1 280	11 / 14.3 / 15	11 / 15.6 / 18	11 / 17.0 / 19
5	6 144	16 / 23.6 / 25	16 / 26.5 / 30	16 / 27.9 / 31
6	28 672	22 / 37.0 / 39	22 / 41.8 / 47	22 / 42.8 / 48
7	131 072	29 / 55.2 / 58	29 / 62.4 / 70	29 / 62.6 / 70

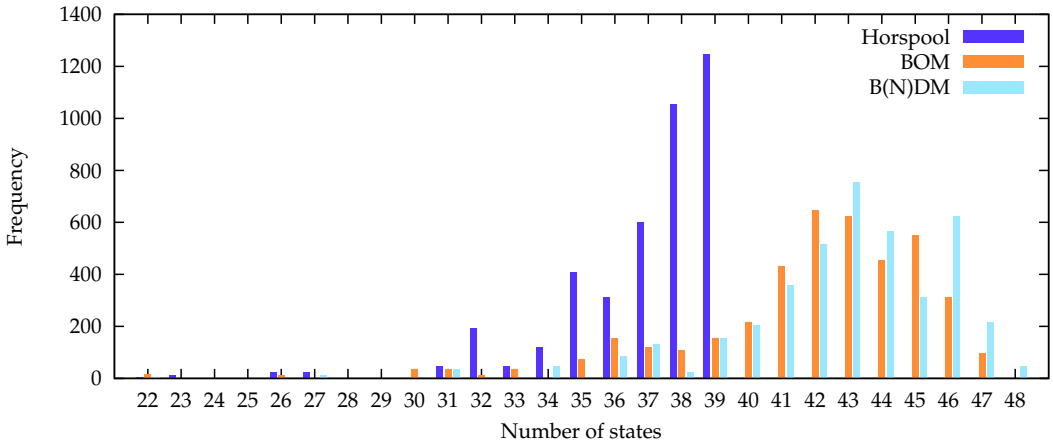


Figure 2.10: Histogram on number of states of minimal DAAs over all 4 096 patterns of length six over $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$.

Runtimes

Computing the distributions shown in Figure 2.7 took 0.3 to 0.6 seconds for each distribution. Distributions of differences as in Figure 2.8 were computed in 14 to 36 seconds.

3 Direct Construction of Minimal DFAs

The statistical properties of a motif are reflected by a DFA accepting all of its instances. This connection is made explicit in Sections 2.6 and 4.3. In all introduced algorithms, the runtime depends at least linearly on the number of DFA states, in some cases the dependence is even cubic. Therefore, it is advisable to minimize DFAs before processing them further. In this chapter, we raise the question whether there are more direct ways to obtain the minimal DFA for a given motif than constructing a non-minimal DFA and subsequently minimizing it. Ultimately, the goal is to find algorithms whose runtime depends linearly on the number of states of the minimal DFA, which would be optimal. Although automata theory has been subject to extensive research for decades, not much attention has been given to this particular topic.

In Section 3.1, we introduce a general approach to the direct construction of minimal DFAs. Although the concept is quite simple and seemingly restrictive, we show that it is strong enough to cover many motifs found in computational biology. In Sections 3.2 and 3.3, we give construction schemes for (sets of) generalized strings and consensus strings with a Hamming neighborhood, respectively.

3.1 Simple NFAs

Our goal is to identify a class of NFAs for which the classical subset construction, written SUBSET-CONSTRUCTION$(\cdot)$ as introduced on Page 21, yields a minimal DFA. To this end, we define *simple NFAs*.

Definition 3.1 (Simple non-deterministic finite automaton)**.** Let $M = (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ be an NFA. M is called *simple* if all states are accessible and the languages $L(q)$ of all states $q \in \mathcal{Q}$ are non-empty and pairwise disjoint. $\lozenge$

A given automaton can easily be modified such that all states have a non-empty language and are accessible: if there is a state q with an empty language, that is, $L(q) = \emptyset$, we can safely remove q from $\mathcal{Q}$ without changing the accepted language. Likewise, all inaccessible states can be removed without changing the accepted language.

Theorem 3.2 (Minimality of DFA constructed from simple NFA)**.** *Let a simple NFA $M_n = (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ be given. Then, the DFA*

$$M_d = \big(\mathcal{Q}_d, \Sigma, \delta, q_0, F_d\big) = \text{SUBSET-CONSTRUCTION}(M_n)$$

is minimal.

Proof. Recall that $\mathcal{Q}_d \subset 2^{\mathcal{Q}}$ and $q_0 = \mathcal{Q}_0$ by definition of SUBSET-CONSTRUCTION. Further recall that $L_{NFA}(q) = L_{DFA}(q)$ for all $q \in \mathcal{Q}_d$ (see Remark 2.8) and that the subscript

is therefore omitted. Let $\mathcal{Q}', \mathcal{Q}'' \in \mathcal{Q}_d$ be two distinct DFA states. By Lemma 2.10, we have to show that $\mathcal{Q}'$ and $\mathcal{Q}''$ are not equivalent, or more formally

$$L(\mathcal{Q}') = \bigcup_{q' \in \mathcal{Q}'} L(q') \neq \bigcup_{q'' \in \mathcal{Q}''} L(q'') = L(\mathcal{Q}'') \,. \tag{3.1}$$

Without loss of generality, assume that $\mathcal{Q}' \setminus \mathcal{Q}'' \neq \emptyset$ and let $q \in \mathcal{Q}' \setminus \mathcal{Q}''$. By Definition 3.1, $L(q) \cap L(q'') = \emptyset$ for all $q'' \in \mathcal{Q}''$ and thus $L(q) \cap L(\mathcal{Q}'') = \emptyset$. But, by choice of q, $L(q) \subset L(\mathcal{Q}')$ and, by Definition 3.1, $L(q) \neq \emptyset$. Hence, it follows that $L(\mathcal{Q}') \neq L(\mathcal{Q}'')$. □

An Alternative Proof

We give an alternative proof of Theorem 3.2 by means of the theory developed by van Glabbeek and Ploeger (2008). They consider five different variants of the classical subset construction. Each variant is characterized by an operation $f : 2^{\mathcal{Q}} \to 2^{\mathcal{Q}}$, where $\mathcal{Q}$ is the state space of an NFA. When a new DFA state is produced in the course of the subset construction, it is subjected to the operation f before being added to the final automaton. In one variant, they define f to be the closure operation

$$\mathit{close}_{\sqsubseteq} : \mathcal{Q}' \mapsto \left\{ q \in \mathcal{Q} \,\middle|\, L(q) \subseteq L(\mathcal{Q}') \right\}$$

and show that the subset construction endowed with this operation directly produces minimal DFAs. Theorem 3.2 now follows from the definition of simple NFAs: as all sets $L(q)$ for $q \in \mathcal{Q}$ are pairwise disjoint, $\mathit{close}_{\sqsubseteq}(\mathcal{Q}') = \mathcal{Q}'$ for each $\mathcal{Q}' \subseteq \mathcal{Q}$ and, thus, the classical subset construction yields a minimal DFA.

Note that the language inclusion problem required to be solved for the $\mathit{close}_{\sqsubseteq}$-operation is in general hard to compute. According to van Glabbeek and Ploeger (2008), it is PSPACE-complete.

Self-Transitions of Start States

In most practical settings like pattern search or pattern statistics, we are given a certain type of pattern and need to construct an automaton that accepts all strings with a suffix matching this pattern, rather than an automaton that accepts only the strings that match the pattern (see Section 2.6). For instance, if the pattern is the single string `ABA` and we want to compute the distribution of the number of occurrences in a random text, we need to build an automaton accepting all strings whose last three letters are `ABA`. For NFAs, we can easily obtain such an automaton once we have constructed an NFA accepting all strings that match the pattern. All we need to do is to modify the transition function Δ by adding self-transitions to all start states

$$\Delta_{\circlearrowleft} : (q, \sigma) \mapsto \begin{cases} \{q\} \cup \Delta(q, \sigma) & \text{if } q \in \mathcal{Q}_0 \,, \\ \Delta(q, \sigma) & \text{otherwise} \,. \end{cases} \tag{3.2}$$

Throughout this thesis, the subscript "$\circlearrowleft$" refers to this modification of a transition function. Furthermore, we use the notation $L_{\circlearrowleft}(q)$ to refer to the language of the state q

with respect to the modified NFA $(\mathcal{Q}, \Sigma, \Delta_{\circlearrowleft}, \mathcal{Q}_0, F)$. Lemma 3.4 characterizes those simple NFAs that remain simple under this modification. Before we can state it, we need to define *one-way start states*.

Definition 3.3 (One-way start states)**.** For an NFA $(\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$, a start state $q_0 \in \mathcal{Q}_0$ is called *one-way start state* if there do not exist $\sigma \in \Sigma$ and $q \in \mathcal{Q} \setminus \{q_0\}$ such that $q_0 \in \Delta(q, \sigma)$, i.e. if it cannot be reached from any state other than itself. ◊

Lemma 3.4. *Let $M = (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ be a simple NFA. The modified automaton $M_{\circlearrowleft} := (\mathcal{Q}, \Sigma, \Delta_{\circlearrowleft}, \mathcal{Q}_0, F)$ is simple if and only if, in M, all start states $q_0 \in \mathcal{Q}_0$ are one-way start states.*

To prove this, we need an auxiliary lemma.

Lemma 3.5. *Let $M = (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ be a simple NFA such that all its start states $q_0 \in \mathcal{Q}_0$ are one-way start states. If $L_{\circlearrowleft}(q) \setminus L(q) \neq \emptyset$ for a $q \in \mathcal{Q}$, then $q \in \mathcal{Q}_0$ and, for all $s \in L_{\circlearrowleft}(q) \setminus L(q)$, there exists a $k \in \mathbb{N}$ such that $s[k..] \in L(q)$.*

Proof. If $q \in \mathcal{Q} \setminus \mathcal{Q}_0$, then no start state can be reached from q as all start states are one-way start states. Thus, $\Delta_{\circlearrowleft}(q', \sigma) = \Delta(q', \sigma)$ for all states q' reachable from q and all $\sigma \in \Sigma$. Hence, $L_{\circlearrowleft}(q) = L(q)$, which implies that $L_{\circlearrowleft}(q) \setminus L(q) = \emptyset$. Therefore, $q \in \mathcal{Q}_0$.

It remains to be shown that there exists a $k \in \mathbb{N}$ such that $s[k..] \in L(q)$. If $q \in F$, this is true as for $k := |s|$ we get $s[k..] = \varepsilon \in L(q)$. If $q \notin F$, then there exists a sequence of states $q_1, \ldots, q_{|s|} \in \mathcal{Q}$ such that $q_1 \in \Delta_{\circlearrowleft}(q, s[0])$, $q_i \in \Delta_{\circlearrowleft}(q_{i-1}, s[i-1])$ for $1 < i < |s|$, and $q_{|s|} \in F$. Now, setting $k := \min\left\{i \in \{0, \ldots, |s|-1\} \,|\, q_{i+1} \neq q\right\}$ implies that $s[k..] \in L(q)$ as all states $q_{k+1}, \ldots q_{|s|}$ cannot be start states and thus their transition function remains unchanged under Equation (3.2). □

Proof of Lemma 3.4. "$\Longrightarrow$": Suppose $M_{\circlearrowleft}$ is simple and there exist $\sigma \in \Sigma$, $q_0 \in \mathcal{Q}_0$, and $q \in \mathcal{Q}$ with $q_0 \neq q$ such that $q_0 \in \Delta(q, \sigma)$. Thus, $\sigma s \in L(q)$ for all $s \in L(q_0)$. Because of the added self-transition, we also have $\sigma s \in L_{\circlearrowleft}(q_0)$ and, thus, $L_{\circlearrowleft}(q_0)$ and $L_{\circlearrowleft}(q)$ are not disjoint, contradicting the assumption that $M_{\circlearrowleft}$ is simple.

"$\Longleftarrow$": Now, we assume that all start states are one-way start states. The properties that all states are accessible and have non-empty languages cannot get lost by adding the additional self-transitions. Therefore, we only need to verify that $L_{\circlearrowleft}(q)$ and $L_{\circlearrowleft}(q')$ are disjoint for all distinct $q, q' \in \mathcal{Q}$. For the sake of contradiction, we assume there exist distinct $q, q' \in \mathcal{Q}$ violating this condition. We choose $s \in L_{\circlearrowleft}(q) \cap L_{\circlearrowleft}(q')$ such that $s \in L_{\circlearrowleft}(q) \setminus L(q)$; if that is not possible, it becomes possible after swapping q and q', because $L(q'') \subseteq L_{\circlearrowleft}(q'')$ for all $q'' \in \mathcal{Q}$ and $L(q) \cap L(q') = \emptyset$. We have to distinguish two cases.

Case 1 ($s \in L(q')$): By Lemma 3.5, q is a start state and there exists a $k \in \mathbb{N}$ such that $s[k..] \in L(q)$. Since all $L(q'')$ for $q'' \in \mathcal{Q}$ are disjoint, it follows that $s[k..] \notin L(q'')$ for all $q'' \in \mathcal{Q} \setminus \{q\}$. As $s \in L(q')$, we thus conclude that $q \in \Delta(q', s[..k-1])$, which contradicts the assumption that q is a one-way start state.

Case 2 ($s \notin L(q')$): By Lemma 3.5, q, q' are start states and there exist $k, k' \in \mathbb{N}$ such that $s[k..] \in L(q)$ and $s[k'..] \in L(q')$. If $k = k'$, then $s[k..] \in L(q) \cap L(q') \neq \emptyset$, contradicting the simpleness of M. We assume, without loss of generality, that $k < k'$.

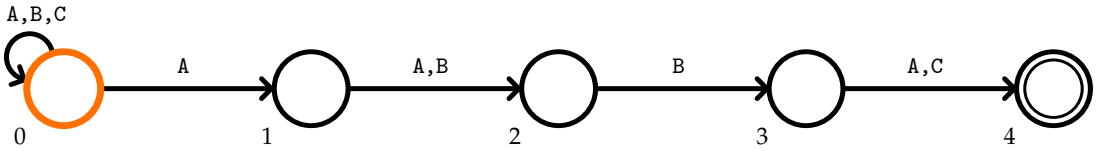


Figure 3.1: Example of a simple NFA (with self-transition added to the start state) constructed from the generalized string $\{\mathtt{A}\}\{\mathtt{A},\mathtt{B}\}\{\mathtt{B}\}\{\mathtt{A},\mathtt{C}\}$ over the alphabet $\Sigma = \{\mathtt{A},\mathtt{B},\mathtt{C}\}$. The start state is drawn in orange and the accepting state is represented by two concentric circles.

Since $s[k'..] \in L(q')$ and $s[k'..] \notin L(q'')$ for all $q'' \in Q \setminus \{q'\}$, we conclude that $q' \in \Delta(q, s[k \dots k'-1])$, contradicting the assumption that q' is a one-way start state. □

3.2 Application to Generalized Strings

In this section, we show that generalized strings and sets of generalized strings admit the construction of simple NFAs. Obviously, a single string is a special case of a set of strings. To aid understandability, we nonetheless start with the easier case of one single string.

3.2.1 Single Generalized Strings

For a generalized string g, an NFA accepting all strings that match g can easily be constructed by connecting the state set $Q = \{0, \dots, |g|\}$ with the transition function

$$\Delta : (q, \sigma) \mapsto \begin{cases} \{q+1\} & \text{if } q < |g| \text{ and } \sigma \in g[q]\,, \\ \emptyset & \text{otherwise}\,. \end{cases}$$

Setting $Q_0 = \{0\}$ and $F = \{|g|\}$ completes the construction of the NFA $(Q, \Sigma, \Delta, Q_0, F)$. For brevity, we write $\mathrm{NFA}(g)$ to denote the automaton created from a generalized string g using the above construction.

Lemma 3.6. *Let g be a generalized string. Then $M_g := \mathrm{NFA}(g)$ is a simple NFA.*

Proof. Clearly, all states $i \in Q$ are accessible and have non-empty languages. M_g admits only transitions from a state i to its successor state $i+1$; only the last state in this chain is an accepting state. Thus, for each state $i \in Q$, the lengths of all accepted strings $s \in L(i)$ equal $|g| - i$. Hence, for two different states i and j, accepted strings have different lengths. Thus, all $L(i)$ must be pairwise disjoint (for $i \in Q$). □

As discussed in Section 3.1, we often need to add a self-transition to the start state. This modification is defined formally in Equation (3.2). We write $\mathrm{NFA}_{\circlearrowleft}(g)$ to refer to the resulting automaton. See Figure 3.1 for an example. Combining Theorem 3.2, Lemma 3.6, and Lemma 3.4, we arrive at the following corollary:

Corollary 3.7. *Let g be a generalized string and $M'_g := \mathrm{NFA}_{\circlearrowleft}(g)$ the corresponding NFA. Then,* SUBSET-CONSTRUCTION(M'_g) *is a minimal DFA.*

3.2.2 Sets of Generalized Strings

In this section, we generalize the above results to finite sets of generalized strings of equal length. Speaking formally, we assume a length ℓ and $G \subset \mathcal{G}^\ell$ to be given and seek to construct a simple NFA that accepts all strings that have a suffix matching a $g \in G$. As above, we first construct an automaton that accepts all strings matching a $g \in G$ and, in a second step, add self-transitions to the start states $\mathcal{Q}_0$.

Remark 3.8 (NFAs with $|F| > 1$ are not simple)**.** One way of building an NFA that accepts a set $G \subset \mathcal{G}^\ell$ is to construct a trie (whose edges are labeled with generalized characters from $\mathcal{G}$) containing all $g \in G$ and using every node as an NFA state. The transition function is directly given by the tries' edge labels. For $|G| > 1$, however, the resulting NFA is *not* simple, because it has more than one accepting state: the languages of all accepting states contain the empty string and, hence, cannot be disjoint. △

The automaton we build is organized level-wise with $\ell+1$ levels. Transitions are only possible between states in adjacent levels and only in one direction (which we choose to call *downwards*). The bottom level contains just one state which is the single accepting state. All states in the top level are start states. As before for a single generalized string, two states q' and q'' in different levels are obviously "language-disjoint", meaning that $L(q') \cap L(q'') = \emptyset$. But here, we possibly need more than one state in a level, which entails the problem of ensuring language-disjointness for states in the same level. We achieve this by using a state space induced by a special parent-child relation between states in adjacent levels. Before we formally construct state space and automaton, the reader may have a look at the example in Figure 3.2.

Let us begin with the formal specification of a suitable state space $\mathcal{Q}$. We choose $\mathcal{Q}$ to be a special subset of $\bar{\mathcal{Q}} := 2^G \times \{0, \ldots, \ell\}$ with the following semantics in mind: to be in state $q = (H, k)$ means that the last k characters read match the first k positions of a $g \in H$. For the definition of $\mathcal{Q}$, we need the function $\textsc{Parent} : \bar{\mathcal{Q}} \times \Sigma \to \bar{\mathcal{Q}} \cup \{\bot\}$ given by

$$\textsc{Parent} : \big((H,k),\sigma\big) \mapsto \begin{cases} \big(\{h \in H \mid \sigma \in h[k-1]\},\, k-1\big) & \text{if } k > 0\,, \\ \bot & \text{otherwise}\,. \end{cases} \tag{3.3}$$

We say that $\textsc{Parent}(q,\sigma)$ is a parent of q under the character σ. The special symbol $\bot$ is used to indicate that a state is in the top level and therefore does not have any parents. The PARENT mapping induces a hierarchy of $\ell + 1$ levels of states:

$$\mathcal{Q}_\ell := \{(G,\ell)\}\,, \tag{3.4}$$

$$\mathcal{Q}_i := \Big\{(H,i') \in \bar{\mathcal{Q}} \;\Big|\; H \neq \emptyset,\ \exists q \in \mathcal{Q}_{i+1}, \sigma \in \Sigma : \textsc{Parent}(q,\sigma) = (H,i')\Big\}\,, \tag{3.5}$$

for $0 \leq i < \ell$. Finally, we write the state space as

$$\mathcal{Q} := \mathcal{Q}_0 \cup \ldots \cup \mathcal{Q}_\ell\,. \tag{3.6}$$

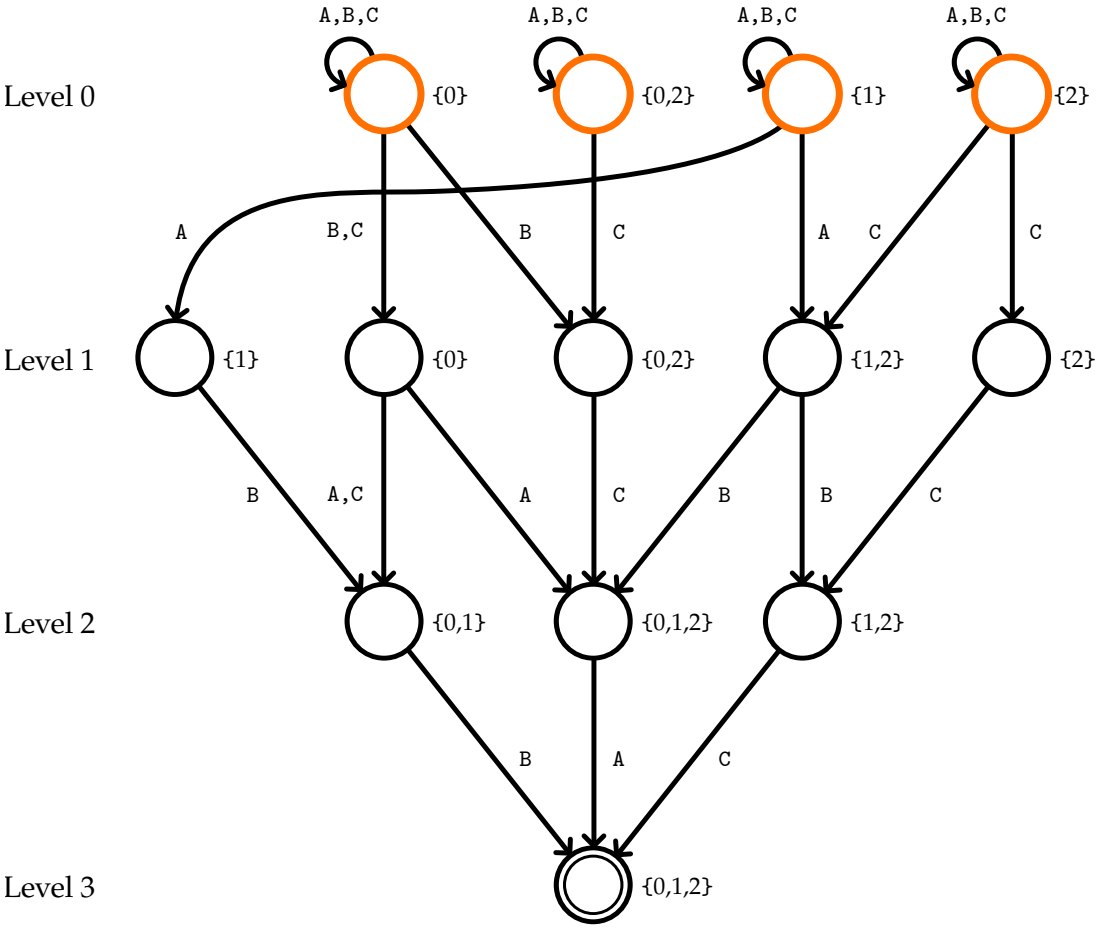


Figure 3.2: Example of a simple NFA constructed from the three generalized strings 0:{B, C}{A, C}{A, B}, 1:{A}{B}{A, B, C}, and 2:{C}{B, C}{A, C} over the alphabet $\Sigma = \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$. Each state is annotated with the set of generalized strings that are "active" in this state (each generalized string is represented by its index 0, 1, or 2). The start states are drawn in orange and the accepting state is represented by two concentric circles.

The PARENT mapping also induces a transition function Δ:

$$\Delta : \big((H,k),\sigma\big) \mapsto \begin{cases} \big\{q \in \mathcal{Q}_{k+1} \,\big|\, \textsc{Parent}(q,\sigma) = (H,k)\big\} & \text{if } k < \ell\,, \\ \emptyset & \text{otherwise}\,. \end{cases} \tag{3.7}$$

To complete the construction, we set

$$F := \mathcal{Q}_\ell = \{(G,\ell)\}$$

and obtain $\mathrm{NFA}(G) := (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$. The next lemma states that an NFA constructed in this way accepts exactly the language given by G.

Lemma 3.9. *Let a length $\ell \in \mathbb{N}$ and a set of generalized strings $G \subset \mathcal{G}^\ell$ be given and set $(\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F) = \mathrm{NFA}(G)$. Then,*

$$\exists q \in \mathcal{Q}_0 : \Delta(q, s) \cap F \neq \emptyset \quad \Longleftrightarrow \quad \exists g \in G : s \lhd g\,,$$

for all $s \in \Sigma^*$.

Proof. We start with the forward direction "$\Longrightarrow$". If $s \in \Sigma^*$ is accepted by $\mathrm{NFA}(G)$, then there exists a sequence of states $q_0, \ldots, q_{|s|}$ such that $q_0 \in \mathcal{Q}_0$, $q_{|s|} \in F$, and $q_i \in \Delta(q_{i-1}, s[i-1])$ for $0 < i \leq |s|$. It follows from Equation (3.7) that $q_{i-1} = \textsc{Parent}(q_i, s[i-1])$. Hence, Equation (3.3) implies that $H_0 \subset \ldots \subset H_{|s|}$, where $(H_i, k_i) := q_i$. Furthermore, by Equation (3.5), H_0 is non-empty. Inductively applying (3.3) now yields that $s \lhd h$ for all $h \in H_0$, which proves the forward direction.

Let us prove the backward direction "$\Longleftarrow$". Let $g \in G$, such that $s \lhd g$. Consider the sequence of states $q'_0, \ldots, q'_{|s|}$ with $(H'_i, k'_i) := q'_i$ given by $q'_{|s|} := (G, \ell)$ and $q'_{i-1} := \textsc{Parent}(q'_i, s[i-1])$ for $0 < i \leq |s|$. From $s \lhd g$ and Equation (3.3) it follows that $g \in H'_i$ for $0 \leq i \leq |s|$. Thus, each H'_i is non-empty and by Equations (3.4) and (3.5) we get $q'_i \in \mathcal{Q}_i$ for $0 \leq i \leq |s|$, implying that q'_0 is a start state. From Equation (3.7) we conclude that $\Delta(q'_0, s) = q'_{|s|}$ which proves the claim as $q'_{|s|} \in \mathcal{Q}_\ell = F$. □

In analogy to Lemma 3.6, we verify that $\mathrm{NFA}(G)$ is indeed a simple NFA.

Lemma 3.10. *Let $\ell \in \mathbb{N}$ and $G \subset \mathcal{G}^\ell$. Then, $M_G := \mathrm{NFA}(G)$ is a simple NFA.*

Proof. The level-wise construction directly implies that all states are accessible and have non-empty languages. States with empty $L(q)$ cannot be generated by Equation (3.5).

It remains to be shown that for all distinct $q, q' \in \mathcal{Q}$ the sets $L(q)$ and $L(q')$ are disjoint. By construction, this is clearly true if q and q' are in different levels. Hence, it suffices to show that

$$L(q) \cap L(q') = \emptyset \text{ for all } q, q' \in \mathcal{Q}_i \text{ with } q \neq q' \tag{3.8}$$

for all i with $0 \leq i \leq \ell$. We prove this by induction on i, starting with $i = \ell$. For $i = \ell$, Condition (3.8) is fulfilled as $|\mathcal{Q}_\ell| = 1$. Assume that (3.8) holds for $i > 0$. For the sake of contradiction, we further assume there exist distinct $q, q' \in \mathcal{Q}_{i-1}$, such that $L(q) \cap L(q') \neq \emptyset$. Let $s \in L(q) \cap L(q')$; it follows that $\Delta(q, s) \in F$. There must exist a state $r \in \mathcal{Q}_i$ such that $\Delta(r, s[1..]) \in F$. As, by the induction hypothesis, Condition (3.8) holds for i, we conclude that the state r is unique. By Equation (3.7), transitions from q and q' can only lead to states in $\mathcal{Q}_i$ implying that $r \in \Delta(q, s[0])$ and $r \in \Delta(q', s[0])$. Applying (3.7) again, we get $q = \textsc{Parent}(r, s[0]) = q'$ and, thus, $q = q'$. □

In Section 3.2.1, we added an initial self-transition to the constructed NFA and obtained an automaton that finds all occurrences of the generalized string in a given text. Now we repeat this step by transforming $\mathrm{NFA}(G)$ using Equation (3.2). Again, we refer to the resulting modified automaton by $\mathrm{NFA}_\circlearrowleft(G)$. Algorithm 3.1 shows how the construction can be implemented. For $|G| = 1$ we obtain the same automaton as constructed in Section 3.2.1. Formally, we get the following theorem.

Algorithm 3.1 Construction of a simple NFA as formalized in Equations (3.4) and (3.5).

Input: Alphabet Σ, length ℓ, and set of generalized strings $G \subset \mathcal{G}_\Sigma^\ell$
Output: Simple NFA accepting all strings $\{st \mid s,t \in \Sigma^*, \exists g \in G : t \lhd g\}$

BUILD-SIMPLE-NFA(Σ, ℓ, G)

```
1  Δ = empty transition map
2  Q_ℓ = {(G, ℓ)}
3  for k = ℓ − 1 downto 0
4      Q_k = ∅
5      for (H', k + 1) ∈ Q_{k+1}, σ ∈ Σ
6          H = {h ∈ H' | σ ∈ h[k]}
7          if H ≠ ∅
8              Q_k = Q_k ∪ {(H, k)}
9              Δ.add-transition(((H, k), σ) ↦ (H', k + 1))
10 for q ∈ Q_0, σ ∈ Σ
11     Δ.add-transition((q, σ) ↦ q)
12 Q = Q_0 ∪ ... ∪ Q_ℓ
13 F = Q_ℓ
14 return (Q, Σ, Δ, Q_0, F)
```

Theorem 3.11. *Let a length $\ell \in \mathbb{N}$ and a set of generalized strings $G \subset \mathcal{G}^\ell$ be given. Then, a minimal automaton accepting the set $\{st \mid s,t \in \Sigma^*, \exists g \in G : t \lhd g\}$ can be constructed in $\mathcal{O}(2^{|G|} \cdot \ell \cdot |\Sigma| \cdot |G| + m)$ time, where m is the size of the minimal DFA.*

Proof. Consider $\text{NFA}_{\circlearrowleft}(G)$. It accepts the language $\{st \mid s,t \in \Sigma^*, \exists g \in G : t \lhd g\}$ by Lemma 3.9 and Equation (3.2). Lemma 3.10 and Lemma 3.4 show that it is indeed a simple NFA. By Theorem 3.2, the resulting DFA is minimal. The runtime follows by analyzing Algorithm 3.1. The loop in Line 3 builds ℓ levels. Each level contains at most $2^{|G|}$ states and thus the inner loop in Line 5 is executed $\mathcal{O}(2^{|G|} \cdot |\Sigma|)$ times for each level, where execution of Line 6 takes $\mathcal{O}(|G|)$ time and the other steps can be performed in constant time. All in all, Algorithm 3.1 takes $\mathcal{O}(2^{|G|} \cdot \ell \cdot |\Sigma| \cdot |G|)$ time. The subsequent subset construction can be implemented to run in $\mathcal{O}(m)$ time leading to the claimed overall runtime. □

Remark 3.12 (Size of minimal DFA)**.** Theorem 3.11 does not give a bound for the minimal DFA's size m. Let us choose an arbitrary set $\mathcal{W} \subset \Sigma^\ell$. On the one hand, the size of the minimal DFA accepting this set is bounded by $\mathcal{O}(\Sigma^\ell)$ as this is a bound for the number of states of an Aho-Corasick automaton (Aho and Corasick, 1975) accepting $\mathcal{W}$. On the other hand, the minimal DFA accepting an arbitrary set $\mathcal{W} \subset \Sigma^\ell$ has a size of $\Omega(\Sigma^{(\ell-1)/2})$ in the worst-case. This can be shown by a counting argument as follows. There are at most $|\mathcal{Q}| \cdot 2^{|\mathcal{Q}|} \cdot |\mathcal{Q}|^{|\mathcal{Q}| \cdot |\Sigma|}$ different DFA's with $|\mathcal{Q}|$ states since we can choose $|\mathcal{Q}|$ different states as start states, can choose the set of accepting states F in $2^{|\mathcal{Q}|}$ different ways, and define at most $|\mathcal{Q}|^{|\mathcal{Q}| \cdot |\Sigma|}$ different transition functions. Therefore, there are at most $|\mathcal{Q}|^2 \cdot 2^{|\mathcal{Q}|} \cdot |\mathcal{Q}|^{|\mathcal{Q}| \cdot |\Sigma|}$ automata with $|\mathcal{Q}|$ states or less. Comparing this

number to the number of possible different word sets $\mathcal{W} \subset \Sigma^\ell$ yields the claimed lower bound:

$$\begin{aligned}
&|\mathcal{Q}|^2 \cdot 2^{|\mathcal{Q}|} \cdot |\mathcal{Q}|^{|\mathcal{Q}|\cdot|\Sigma|} \geq 2^{\left(|\Sigma|^\ell\right)} \\
\implies &2 \cdot \log|\mathcal{Q}| + |\mathcal{Q}| \cdot \log 2 + |\mathcal{Q}| \cdot |\Sigma| \cdot \log|\mathcal{Q}| \geq |\Sigma|^\ell \cdot \log 2 \\
\implies &3|\mathcal{Q}|^2 \geq |\Sigma|^{\ell-1} \\
\implies &|\mathcal{Q}| \in \Omega\left(|\Sigma|^{\frac{\ell-1}{2}}\right) .
\end{aligned}$$

This shows that the worst-case size of the minimal DFA accepting a subset of Σ^ℓ grows exponentially in ℓ. However, there are less than $2^{\ell \cdot k \cdot |\Sigma|}$ subsets of Σ^ℓ that can be expressed as a set of generalized strings $G \subset \mathcal{G}^\ell$ with $|G| \leq k$. Whether or not there exists a bound for the size of the minimal DFA accepting G that is not exponential in ℓ, remains an open question. △

3.3 Application to Consensus Strings with a Hamming Neighborhood

Another type of motif commonly used in computational biology is a consensus string along with a distance threshold. Here, we assume that a (generalized) string s and a distance threshold d_{max} are given and want to compute the minimal DFA that accepts all strings with a Hamming distance to s of at most d_{max}, where the Hamming distance between a string s and a generalized string g of the same length is defined as

$$d(s, g) := \left|\left\{ i \in \{0, \ldots, |s| - 1\} \;\middle|\; s[i] \notin g[i] \right\}\right| .$$

We construct a simple NFA accepting a generalized string and its Hamming neighborhood. The construction is similar to the one given in the textbook by Navarro and Raffinot (2002). Interestingly, the resulting NFA turns out to be simple.

The basic idea for the construction is to use a two-dimensional grid as a state space, where we advance into one dimension whenever a valid character has been read and into the other dimension for each mismatch. Figure 3.3 illustrates an NFA built in this way. Formally the state space is defined by

$$\mathcal{Q} := \left\{ (e, k) \in \{0, \ldots, d_{max}\} \times \{0, \ldots, |g|\} \;\middle|\; |g| - k - e \geq 0 \right\} \tag{3.9}$$

with the following semantics: state (e, k) accepts all strings of length $|g| - k$ that match the respective suffix of g with exactly e errors. The condition $|g| - k - e \geq 0$ states that the number of errors e cannot be larger than $|g| - k$, which is the number of characters left. We define the transition function to obey these semantics:

$$\Delta : (e, k) \times \sigma \mapsto \begin{cases} z(e, k+1) & \text{if } \sigma \in g[k] , \\ z(e-1, k+1) & \text{otherwise} , \end{cases} \tag{3.10}$$

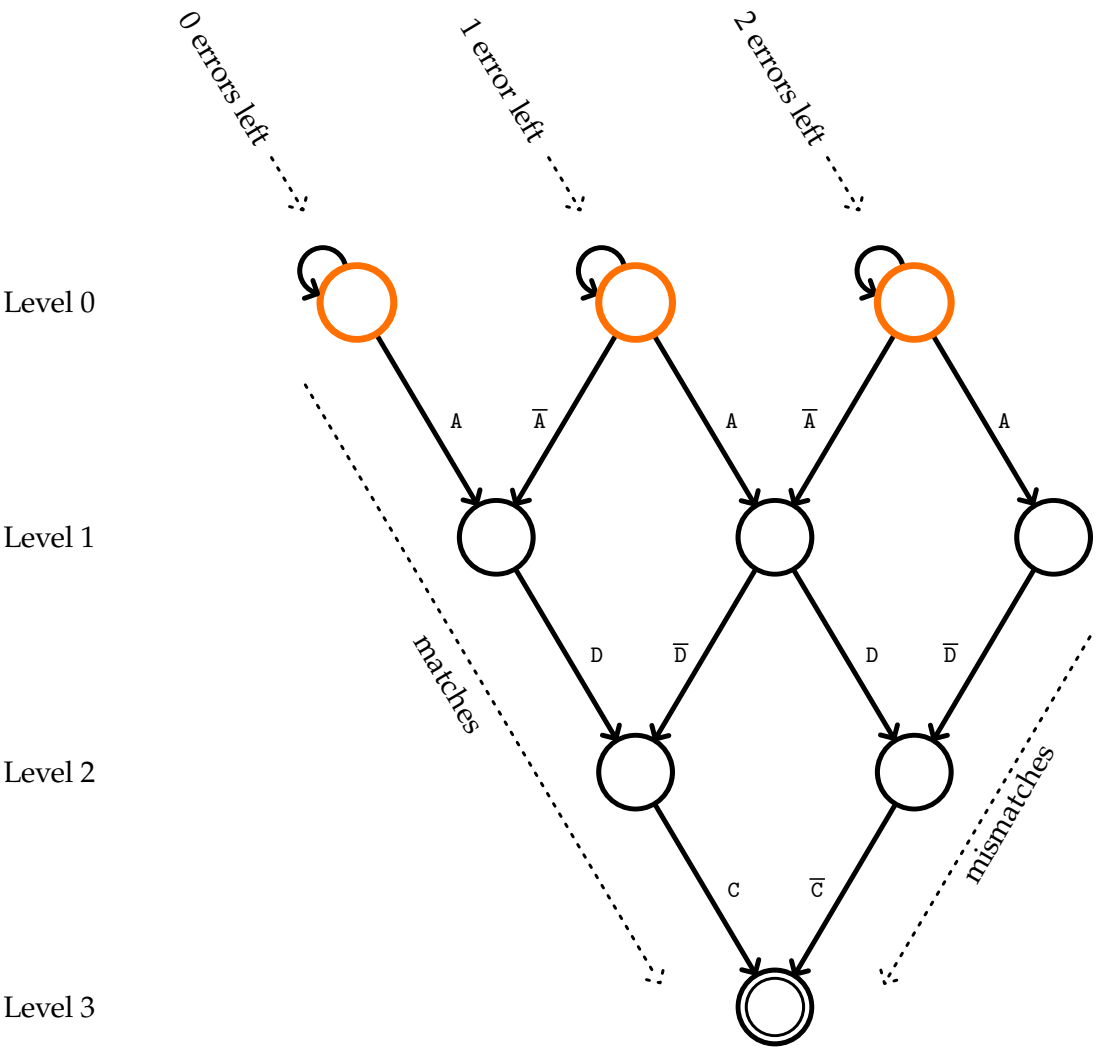


Figure 3.3: Example of a simple NFA over the alphabet $\Sigma = \{\texttt{A},\texttt{B},\texttt{C},\texttt{D}\}$ accepting the consensus ADC and all strings within a Hamming distance of two or less. Characters with bars stand for the complement with respect to Σ, e.g. $\overline{\texttt{A}}$ stands for B, C, or D. The start states are drawn in orange and the accepting state is represented by two concentric circles.

where the function $z : \mathbb{Z} \times \mathbb{Z} \to 2^{\mathcal{Q}}$ returns the empty set whenever we "fall off the grid". More precisely,

$$z : (e,k) \mapsto \begin{cases} \{(e,k)\} & \text{if } (e,k) \in \mathcal{Q}\,, \\ \emptyset & \text{otherwise}\,. \end{cases} \tag{3.11}$$

As before, the topmost level constitutes the start states, i.e. $\mathcal{Q}_0 := \{(e,k) \in \mathcal{Q} \mid k = 0\}$, and the bottommost level contains only the single accepting state, i.e. $F := \{(0,|g|)\}$.

We write $\mathrm{NFA}(g, d_{max}) := (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ to denote the NFA constructed in this way. Again, we use the notation $\mathrm{NFA}_{\circlearrowleft}(g, d_{max}) := (\mathcal{Q}, \Sigma, \Delta_{\circlearrowleft}, \mathcal{Q}_0, F)$ to refer to the automaton with self-transitions added to the start states. Note that for $d_{max} = 0$, the resulting automaton is isomorphic to the one constructed from a single generalized string in Section 3.2.1.

In order to prove that the construction is correct and produces simple NFAs, we use the following Lemma on the states' languages.

Lemma 3.13. *Let $g \in \mathcal{G}^*_\Sigma$, $d_{max} \in \mathbb{N}_0$ and $M = \mathrm{NFA}(g, d_{max}) = (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$. Then, the language of state (e, k) is characterized by*

$$L\big((e,k)\big) = \Big\{ s \in \Sigma^{|g|-k} \;\Big|\; d\big(s, g[k..]\big) = e \Big\}$$

for all $(e, k) \in \mathcal{Q}$.

Proof. We start with the direction "$\subseteq$". By construction of Δ and F, we have $L\big((e,k)\big) \subseteq \Sigma^{|g|-k}$. Let $s \in L\big((e,k)\big)$, then $\Delta\big((e,k), s\big) = (0, |g|)$. That means that, in the course of $|s|$ state transitions, the first component of the state changes from e to zero. As we see from Equation (3.10), the only change possible in the first component is a decrease by one, which happens if and only if the read character is a mismatch. Thus, it follows that $d\big(s, g[k..]\big) = e$.

Now we prove the direction "$\supseteq$". Let $s \in \Sigma^{|g|-k}$ and $d\big(s, g[k..]\big) = e$. That means there are exactly e indices $a_0, \ldots, a_{e-1}$ such that $s[a_i] \notin g[k + a_i]$ for $0 \leq i < e$. Provided that all states exist and thus the z function never returns $\emptyset$, we apply the first case of (3.10) exactly $|s| - e$ times and the second case exactly e times, ending in state $(0, |g|)$ as claimed. The only thing left to verify is that z indeed never returns $\emptyset$. Note that, by (3.10), the term $|g| - k - e$ cannot increase. Since it reaches zero after $|s|$ steps, it cannot have been smaller than zero at any time. Hence, by Equation (3.9), all intermediate states exist and, thus, the first case of Equation (3.11) is applied for all state transitions. □

Using this lemma, the construction's correctness is easily verified:

Lemma 3.14. *Let $g \in \mathcal{G}^*_\Sigma$, $d_{max} \in \mathbb{N}_0$ and $M = \mathrm{NFA}(g, d_{max}) = (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$. Then, M accepts exactly the strings $\{s \in \Sigma^{|g|} \mid d(s, g) \leq d_{max}\}$.*

Proof. By definition, the automaton M accepts the strings $L(\mathcal{Q}_0)$. By construction of $\mathcal{Q}_0$ and Lemma 3.13, we obtain

$$L(\mathcal{Q}_0) = \bigcup_{e=0}^{\min\{d_{max}, |g|\}} L\big((e,0)\big) = \bigcup_{e=0}^{\min\{d_{max}, |g|\}} \big\{ s \in \Sigma^{|g|} \;\big|\; d(s,g) = e \big\}.$$

□

Lemma 3.15. *Let $g \in \mathcal{G}^*_\Sigma$, $d_{max} \in \mathbb{N}_0$. Then, $\mathrm{NFA}(g, d_{max}) = (\mathcal{Q}, \Sigma, \Delta, \mathcal{Q}_0, F)$ is a simple NFA.*

Proof. By construction, all states are accessible and have non-empty languages. The disjointness of $L(q)$ and $L(q')$ for distinct $q, q' \in \mathcal{Q}$ follows immediately from Lemma 3.13. □

In analogy to Sections 3.2.1 and 3.2.2, we can now add self-transitions to the start states to obtain $\text{NFA}_{\circlearrowleft}(g, d_{max})$ and arrive at the following theorem.

Theorem 3.16. *Let a generalized string $g \in \mathcal{G}^*$ and an error threshold d_{max} be given. Then, a minimal automaton accepting the set $\{st \mid s \in \Sigma^*, t \in \Sigma^{|g|}, d(t, g) \leq d_{max}\}$ can be constructed in $\mathcal{O}(|g| \cdot d_{max} + m)$ time, where m is the size of the minimal DFA.*

Proof. $\text{NFA}(g, d_{max})$ is simple by Lemma 3.15. Furthermore, it fulfills the conditions of Lemma 3.4. Thus, $\text{NFA}_{\circlearrowleft}(g, d_{max})$ is also simple and, by Lemma 3.14 and Equation (3.2), accepts the language $\{st \mid s \in \Sigma^*, t \in \Sigma^{|g|}, d(t, g) \leq d_{max}\}$. The subset construction directly yields the minimal DFA by Theorem 3.2. The claimed runtime follows directly. □

4 Clump Statistics

The main difficulty in computing a motif's occurrence count distribution lies in the need to take self-overlaps into account. The stronger a motif's tendency to overlap itself is, the larger is the variance of this distribution. As discussed in Section 2.6, it can exactly be computed based on the DFA accepting the motif as the DFA implicitly encodes the structure of self-overlaps. Here, we approximate it by a compound Poisson distribution (see Section 2.2). For this, we need the notion of *clumps of words*.

Definition 4.1 (Clump)**.** Given a sequence $s \in \Sigma^*$ and a pattern p, a *clump* is a maximal set of overlapping occurrences of p in s. The number of occurrences it contains is called *size of a clump*. ◊

Example 4.2. Let $p := \texttt{ACA}$ and $s := \texttt{G}\underline{\texttt{ACACA}}\texttt{TT}\underline{\texttt{ACA}}\texttt{AA}$. Then, s contains three occurrences of p in two clumps (underlined). The first clump has size two while the second has size one. △

Using Definition 4.1, we can uniquely decompose the set of all occurrences of a pattern in a text into clumps. Furthermore, we can approximate the distribution of the *number* of clumps in a random text by a Poisson distribution. Comparing this approximation to the exact distribution of the number of clumps, Stefanov et al. (2007) conclude that it is accurate for rare words, justifying this approach. We now obtain a compound Poisson approximation by computing the *expected number of clumps* and the *distribution of clump sizes*.

The resulting approximation has two advantages compared to the exact distribution. First, the time to calculate it does not depend on the text length, leading to a speed-up in practice. And second, it allows us to establish bounds on the p-value that can be exploited in branch-and-bound motif discovery algorithms as we see in Chapter 5.

For their utility and accuracy, compound Poisson approximations have been discussed by multiple other authors, including Schbath (1995), Waterman (1995), and Roquain and Schbath (2007). As new contributions, we derive an exact yet simple formula for the expected clump size and an algorithm to compute the exact clump size distribution. Both results hold for arbitrary finite-memory text models.

In Section 4.1, we give several examples of overlapping patterns and demonstrate the effects of clumping on motif statistics. A formula for the expected clump size is derived in Section 4.2 and an algorithm to compute the complete distribution of clump sizes is developed in Section 4.3. The chapter is concluded with an evaluation of the accuracy of compound Poisson approximations in Section 4.4.

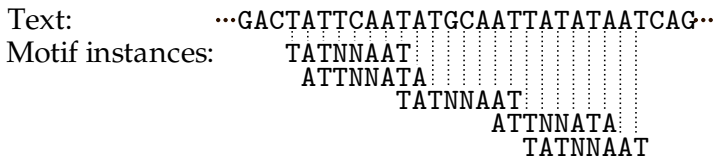


Figure 4.1: Example of a text containing a clump (of size five) of the motif TATNNAAT when considered jointly with its reverse complement.

4.1 Effects of Overlaps

As a first example, consider a random text of length 10 000 over $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ with respect to the uniform text model. The expected number of occurrences for the two patterns AAAAAAAAAA and AAAAAAAAAC computes to

$$\left(\frac{1}{4}\right)^{10} \cdot (10\,000 - 10 + 1) \approx 0.0095$$

in both cases. The probabilities of observing 10 or more occurrences, however, differ greatly and amount to $2.982 \cdot 10^{-8}$ and $1.546 \cdot 10^{-27}$, respectively. The different behavior is reflected in the expected clump sizes that equal $4/3$ and 1, respectively. When computing the compound Poisson approximations of these p-values, we obtain $2.986 \cdot 10^{-8}$ and $1.685 \cdot 10^{-27}$, respectively. This example shows that the effect of clumping can have a substantial influence on pattern statistics. Furthermore, it illustrates that clumping can be taken into account by using a compound Poisson approximation.

Keeping track of possible overlaps gets more complicated when, on the one hand, generalized strings and, on the other hand, both strands of DNA are considered. Recall that DNA is a double-stranded molecule and each nucleotide forms a bond with its complement on the opposite strand: C pairs with G and A pairs with T. Suppose now we are interested in the IUPAC motif TATNNAAT and want to count matches on both strands. In effect, that means we count all instances of TATNNAAT *and* ATTNNATA on the forward strand, where the second motif is the *reverse complement* of the first. Together, they can (self-)overlap in multiple ways, which is illustrated in Figure 4.1. Therefore, the expected clump size of the joint motif is larger than that of the forward motif alone. With respect to the uniform text model, the expected clump sizes amount to 1.041 and 1.001, respectively. The complete clump size distribution for both cases is shown in the left part of Figure 4.2.

We call a motif that equals its reverse complement *palindromic*. For example, ATANNTAT is palindromic, which strongly affects its statistical properties. Whenever a match on the forward strand is found, another match is found at the same position on the backward strand. This means that the probability of observing an odd clump size is zero. The right part of Figure 4.2 visualizes this behavior.

The clump size distribution of a motif influences not only the probability of observing many motif instances, but also the probability of observing no instances at all. This fact might be counter-intuitive on first sight. It can be explained by means of the

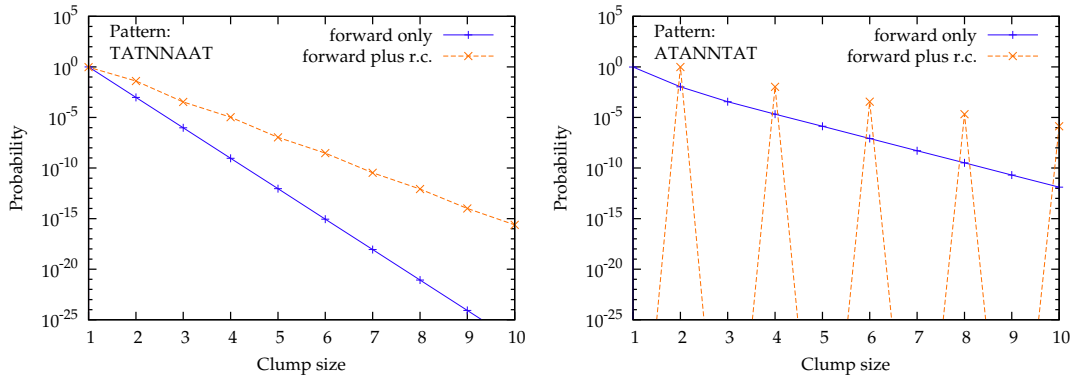


Figure 4.2: Clump size distributions of the motifs `TATNNAAT` (left) and `ATANNTAT` (right) with respect to the uniform text model. Both figures show the distribution for the motif alone (labeled "forward only") and for the motif considered jointly with its reverse complement (labeled "forward plus r.c."). The second distribution in the right figure contains zero probabilities. As there is no zero at the (logarithmically scaled) y-axis, they are indicated by lines ending at the bottom of the plot.

two motifs `TATNNAAT` and `ATANNTAT` discussed above. We consider them jointly with their reverse complement, corresponding to the orange curves in Figure 4.2. For both (joint) motifs the expected number of occurrences with respect to the uniform text model (and to any other i.i.d. text model) is the same as both are composed of the same characters. In contrast, the expected clump sizes are different and amount to 1.041 and 2.131, respectively. The expected *number* of clumps is now given by

$$\text{expected number of clumps} = \frac{\text{expected number of occurrences}}{\text{expected clump size}}\,. \tag{4.1}$$

Obviously, these quantities differ for the two motifs. Knowing that the number of clumps is accurately approximated by a Poisson distribution, we conclude that the probabilities of observing zero occurrences (and therefore zero clumps) must also differ. In these two cases, the (exact) probabilities of finding no motif occurrences in a random text of length 1 000 are 0.627 and 0.787, respectively. Indeed, the probability of observing no matches is higher for the motif with larger expected clump size. When using the compound Poisson approximation instead of the exact probabilities, we get 0.628 and 0.787, respectively. Again, the compound Poisson approximation accurately reflects the effects of clumping.

The discussed probabilities of finding no match are important in the context of motif discovery as formalized in Problem 2 on Page 10. Suppose we have examined a set S of 50 genomic regions (promotor regions, for instance) of length 1 000 and determined that each of the two motifs is contained in 35 of these regions. The p-values for these

events are now given as the tail probabilities of binomial distributions:

$$pvalue_{seqs}\left(\{\{\texttt{TATNNAAT}, \texttt{ATTNNATA}\}\}, \mathcal{S}\right) = \sum_{k=35}^{50} \mathcal{B}_{50,(1-0.628)}(k) \approx 2.6 \cdot 10^{-6},$$

$$pvalue_{seqs}\left(\{\{\texttt{ATANNTAT}, \texttt{ATANNTAT}\}\}, \mathcal{S}\right) = \sum_{k=35}^{50} \mathcal{B}_{50,(1-0.787)}(k) \approx 2.2 \cdot 10^{-13}.$$

In this case, the fact that the expected clump sizes differ leads to a large difference in p-values. The effect is particularly strong because one motif is palindromic while the other is not, but, in principle, it is present for any two motifs with different expected clump size. If this seems counter-intuitive, note that for a palindromic motif, knowing that it does not occur on one strand implies that is does not occur on the other strand either. For a non-palindromic motif, there is still a chance that it occurs on the complementary strand.

4.2 Expected Clump Size

In this section, we derive a formula for the expected clump size. We assume a motif of length ℓ to be given and that the (non-empty) set $\mathcal{W} \subset \Sigma^\ell$ is the set of all words that match the motif, that is, the set of all possible motif instances. For the considerations in this chapter, it is irrelevant whether $\mathcal{W}$ is given through a generalized string, a set of generalized strings, a consensus string and its Hamming neighborhood, a PWM with a threshold, or arbitrary choice. The only assumptions we make is that all $w \in \mathcal{W}$ have the same length ℓ.

The set of words $\mathcal{W} \subset \Sigma^\ell$ can be chosen such that a clump, once begun, cannot end. A trivial example for that is $\mathcal{W} = \Sigma^\ell$, but we can also construct sets $\mathcal{W} \subsetneq \Sigma^\ell$ with this property. For instance, let $\Sigma = \{\texttt{A}, \texttt{B}\}$ and $\mathcal{W} = \{\texttt{AAA}, \texttt{AAB}, \texttt{ABA}, \texttt{BAB}, \texttt{BBA}, \texttt{BBB}\}$ and observe that for all $w \in \mathcal{W}$ there do not exist $\sigma_1, \sigma_2 \in \Sigma$ such that $w[1]w[2]\sigma_1 \notin \mathcal{W}$ and $w[2]\sigma_1\sigma_2 \notin \mathcal{W}$. This notion of degeneracy is formalized in the following definition.

Definition 4.3 (Degenerate word sets)**.** A set $\mathcal{W} \subset \Sigma^\ell$ is called *degenerate* if $occ_{\mathcal{W}}(s) \geq 1$ for all $s \in \Sigma^{2\ell-2}$. ◊

If $\mathcal{W}$ is degenerate in this sense, then every window of length $2\ell - 2$ of any string contains at least one occurrence of $\mathcal{W}$. These occurrences in two consecutive windows must overlap by choice of the length $2\ell - 2$. Hence, a clump cannot end. If, conversely, $\mathcal{W}$ is not degenerate, then there exists a string $f \in \Sigma^{2\ell-2}$ that does not contain a word from $\mathcal{W}$. Therefore, appending f to any $w \in \mathcal{W}$ ends a clump.

Remark 4.4 (Degenerate generalized strings)**.** When $\mathcal{W}$ is the set of all words that match a generalized string $p \in \mathcal{G}$, it is degenerate if and only if $p[k] = \Sigma$ for all $k \in \{0, \ldots, |p| - 1\}$: if there is a k with $p[k] \neq \Sigma$, then a string f consisting exclusively of characters from $\Sigma \setminus p[k]$ cannot contain an instance of $\mathcal{W}$ and, hence, $\mathcal{W}$ is not degenerate. △

Definition 4.5 (Clump process)**.** Let $(S_t)_{t\in\mathbb{N}_0}$ be a random text distributed according to a finite-memory text model and $\mathcal{W} \subset \Sigma^\ell$ be a set of words that is neither empty nor degenerate. The *clump process* $(Z_i^{\mathcal{W}})_{i\in\mathbb{N}_0}$ is a sequence of random variables $Z_i^{\mathcal{W}}$, where each $Z_i^{\mathcal{W}}$ gives the size of the i-th clump of words $\mathcal{W}$ in the text $(S_t)_{t\in\mathbb{N}_0}$. If no i-th clump exists, we set $Z_i^{\mathcal{W}} := 0$; this happens with zero probability for infinite texts because we assume all text models to be well-behaved according to Definition 2.18. ◊

Definition 4.6 (Expected clump size)**.** Let a non-empty word set $\mathcal{W} \subset \Sigma^\ell$ be given. If $\mathcal{W}$ is not degenerate, the limit

$$\psi_{\mathcal{W}} := \lim_{i\to\infty} \mathbb{E}\left(Z_i^{\mathcal{W}}\right) \tag{4.2}$$

is called *the expected clump **size** of* $\mathcal{W}$, otherwise we define $\psi_{\mathcal{W}} := \infty$. ◊

Note that, by Definition 2.18, the distribution of text model states $\mathcal{L}(C_t)$ converges to an equilibrium when t goes to infinity. Therefore, the distributions of clump sizes $\mathcal{L}\big(Z_i^{\mathcal{W}}\big)$ and, thus, the expected clump sizes $\mathbb{E}\big(Z_i^{\mathcal{W}}\big)$ converge as well. Therefore, $\psi_{\mathcal{W}}$ as given in Definition 4.6 is well-defined. In the following, we omit the superscript $\mathcal{W}$ as the set $\mathcal{W}$ is fixed throughout this chapter (although it can be chosen arbitrarily).

Recall that we write $S_i^\ell = S_i \cdots S_{i+\ell-1}$ to refer to the length-ℓ substring of the random text $(S_t)_{t\in\mathbb{N}_0}$ starting at position i. To analyze clumps with respect to finite-memory text models, it is not sufficient to look at strings S_t^ℓ only, but we simultaneously need to keep track of the text model state. To shorten notation, we define

$$X_t := (S_t^\ell, C_t)\,,$$

so that $X_t = (w, c)$ means that word $w \in \Sigma^\ell$ starts at position t in S, and before generating its first letter, we are in context c, i.e., $S_t^\ell = w$ and $C_t = c$. We say that *word w occurs in context c at position t*. Furthermore, we define

$$\mathcal{X} := \Big\{(w,c) \in \mathcal{W}\times\mathcal{C} \,:\, \lim_{t\to\infty} \mathbb{P}\big(X_t = (w,c)\big) > 0\Big\}\,.$$

Restricting attention to $\mathcal{X}$ is sufficient as pairs (w, c) with zero occurrence probability do not contribute to the expected clump size ψ. To derive a formula for ψ, we need several definitions.

Definition 4.7 (Overlap probability function)**.** The *overlap probability function* $\kappa : \mathcal{X} \times \mathcal{X} \to [0, 1]$ is defined by

$$\kappa\big((w_1,c_1),(w_2,c_2)\big) := \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_{t+i} = (w_2,c_2), S_{t+1}^\ell, \ldots, S_{t+i-1}^\ell \notin \mathcal{W} \,\big|\, X_t = (w_1,c_1)\big).$$

◊

By definition of finite-memory text models, the involved conditional probabilities do not depend on t. Intuitively, $\kappa\big((w_1, c_1), (w_2, c_2)\big)$ is the probability that, having seen w_1 in context c_1, there follows another word from $\mathcal{W}$ in the same clump and that the *next* such word is w_2 in context c_2.

Example 4.8 (Overlap probability function)**.** Let $\Sigma = \{\mathtt{A}, \mathtt{B}\}$ and consider an i.i.d. text model, that means a finite-memory text model with only one context c_0. Let the character probabilities be given by $p_{\mathtt{A}} = 0.1$, $p_{\mathtt{B}} = 0.9$. Let further $\mathcal{W} := \{\mathtt{AAA}, \mathtt{AAB}, \mathtt{ABA}\}$. We obtain the values $\kappa\big((\mathtt{AAA}, c_0), (\mathtt{AAA}, c_0)\big) = 0.1$, $\kappa\big((\mathtt{AAA}, c_0), (\mathtt{AAB}, c_0)\big) = 0.9$, and $\kappa\big((\mathtt{AAA}, c_0), (\mathtt{ABA}, c_0)\big) = 0$. The last probability is zero as ABA cannot right-overlap AAA without first creating an occurrence of AAB. △

It is useful to view κ as a matrix.

Definition 4.9 (Overlap matrix)**.** Given a bijective mapping $\iota : \mathcal{X} \to \{0, \ldots, |\mathcal{X}| - 1\}$, the matrix $K = \big(k_{\iota(w,c),\iota(w',c')}\big) \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$ with $k_{\iota(w,c),\iota(w',c')} := \kappa\big((w,c),(w',c')\big)$ is called *overlap matrix*. As all results in this thesis hold independent of the choice of ι, we omit ι and use pairs $(w,c) \in \mathcal{X}$ as indices directly. ◊

Whether or not $\mathcal{W}$ is degenerate in the sense of Definition 4.3 is directly reflected in the overlap matrix K.

Lemma 4.10 (Overlap matrix for degenerate word sets)**.** *Let $\mathcal{W} \subset \Sigma^\ell$ and a finite-memory text model be given. As always, assume the text model to be well-behaved according to Definition 2.18. Further, let K be the corresponding overlap matrix given by Definition 4.9. If $\mathcal{W}$ is non-degenerate, then* $\operatorname{sprad}(K) < 1$*, where* $\operatorname{sprad}(K)$ *is the spectral radius of K as introduced in Definition A.5 on Page 144.*

Proof. Let $|w,c\rangle \in \mathbb{R}^{|\mathcal{X}|}$ be a vector such that the component with index (w,c) is one and all others are zero. The set of all $|w,c\rangle$ for $(w,c) \in \mathcal{X}$ is an orthonormal basis of $\mathbb{R}^{|\mathcal{X}|}$. By definition of K, the matrix element $\langle w_1, c_1|K^n|w_2, c_2\rangle$ for $(w_1,c_1),(w_2,c_2) \in \mathcal{X}$ and $n \in \mathbb{N}$ is the probability that, starting from an occurrence of w_1 in context c_1, a clump contains at least n more occurrences of words from $\mathcal{W}$ and the n-th of these occurrences is w_2 in context c_2. If $\mathcal{W}$ is non-degenerate, then $\lim_{n\to\infty}\langle w_1, c_1|K^n|w_2, c_2\rangle = 0$ for all $(w_1,c_1),(w_2,c_2) \in \mathcal{X}$. This is the case because there exists a string $f \in \Sigma^{2\ell-2}$ that ends the clump and a well-behaved text model guarantees that the probability that f occurs is non-zero for every position. Since the set of all $|w,c\rangle$ for $(w,c) \in \mathcal{X}$ is a basis, it follows that $\lim_{n\to\infty} K^n|x\rangle = |0\rangle$ for all $|x\rangle \in \mathbb{R}^{|\mathcal{X}|}$. Now we choose $|x\rangle$ to be an eigenvector with eigenvalue λ, i.e. $K|x\rangle = \lambda|x\rangle$. As

$$\lim_{n\to\infty} K^n|x\rangle = \lim_{n\to\infty} \lambda^n|x\rangle = |0\rangle\,,$$

we conclude that $|\lambda| < 1$. □

Definition 4.11 (Word and clump start distributions)**.** The *word distribution vector*, denoted $|p\rangle = \big(p_{(w,c)}\big) \in \mathbb{R}^{|\mathcal{X}|}$, is given by

$$p_{(w,c)} := \lim_{t\to\infty} \mathbb{P}\left(X_t = (w,c) \,\middle|\, S_t^\ell \in \mathcal{W}\right).$$

It is the equilibrium probability to see word w in context c, given that a word from $\mathcal{W}$ is seen. The *clump start distribution vector* $\big|p^{\mathit{start}}\big\rangle = \Big(p^{\mathit{start}}_{(w,c)}\Big) \in \mathbb{R}^{|\mathcal{X}|}$ is given by

$$p^{\mathit{start}}_{(w,c)} := \lim_{t\to\infty} \mathbb{P}\left(X_t = (w,c) \,\middle|\, S_t^\ell \in \mathcal{W}, S_{t-1}^\ell, \ldots, S_{t-\ell+1}^\ell \notin \mathcal{W}\right). \tag{4.3}$$

It is the equilibrium probability to see word w in context c, given that a word from $\mathcal{W}$ is seen and starts a clump of such words. ◊

Using the definitions made above, we are now ready to state the main theorem of this section. It gives a simple formula for the expected clump size.

Theorem 4.12 (Expected clump size)**.** *Let a non-empty pattern set $\mathcal{W} \subset \Sigma^\ell$ and a finite-memory text model be given. The expected clump size of $\mathcal{W}$ with respect to the text model is finite if and only if $\mathcal{W}$ is non-degenerate. In this case, it is given by*

$$\psi \;=\; \big\langle p^{\mathit{start}}\big|(\mathbf{1}-K)^{-1}\big|1\big\rangle \;=\; \frac{1}{1-\langle p|K|1\rangle}\,,$$

where $\big|p^{\mathit{start}}\big\rangle$, $|p\rangle$ and K are defined according to Definitions 4.11 and 4.9, respectively.

Needed definitions and facts from matrix theory are given in Appendix A.3. To prove the theorem, we need additional definitions and an auxiliary lemma.

Definition 4.13 (Clump end vector)**.** The *clump end vector*, denoted $|f\rangle = \big(f_{(w,c)}\big) \in \mathbb{R}^m$, is given by

$$f_{(w,c)} := \mathbb{P}\big(S^\ell_{t+1},\ldots,S^\ell_{t+\ell-1} \notin \mathcal{W} \,|\, X_t = (w,c)\big),$$

which is the conditional probability that, when seeing word w in context c, no further word from $\mathcal{W}$ follows in the same clump. Here $f_{(w,c)}$ does not depend on t due to conditioning on $C_t = c$. ◊

We express f in terms of the overlap matrix:

$$f_{(w,c)} = 1 - \sum_{(w',c')\in\mathcal{X}} k_{(w,c),(w',c')} \quad \text{or} \quad |f\rangle = (\mathbf{1}-K)|1\rangle. \tag{4.4}$$

Definition 4.14 (Backward overlap function and matrix)**.** The *backward overlap function* $\overleftarrow{\kappa} : \mathcal{X}\times\mathcal{X} \to [0,1]$ is defined by

$$\overleftarrow{\kappa}\big((w_1,c_1),(w_2,c_2)\big) := \\ \lim_{t\to\infty} \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_{t-i} = (w_2,c_2), S^\ell_{t-i+1},\ldots,S^\ell_{t-1} \notin \mathcal{W} \,|\, X_t = (w_1,c_1)\big).$$

◊

Intuitively, $\overleftarrow{\kappa}\big((w_1,c_1),(w_2,c_2)\big)$ is the equilibrium probability that, observing w_1 in context c_1, there exists a preceding word from $\mathcal{W}$ in the same clump and that the immediately preceding such word is w_2 in context c_2. Note the symmetry to Definition 4.7; however, the conditional probability may depend on t, so we take the equilibrium limit. The *backward overlap matrix* $\overleftarrow{K}$ is defined accordingly.

Lemma 4.15. *For all $(w_1,c_1),(w_2,c_2) \in \mathcal{X}$,*

$$p_{(w_1,c_1)} k_{(w_1,c_1),(w_2,c_2)} = p_{(w_2,c_2)}\, \overleftarrow{k}_{(w_2,c_2),(w_1,c_1)}.$$

This is a "detailed balance" between p, K and $\overleftarrow{K}$: The equilibrium probability of seeing a word-context pair (w_1, c_1) followed by (w_2, c_2) in the same clump equals the equilibrium probability of (w_2, c_2) preceded by (w_1, c_1).

Proof. The claim obviously holds if $p_{(w_1,c_1)} = 0$ or $p_{(w_2,c_2)} = 0$. Otherwise,

$$\begin{aligned}
p_{(w_1,c_1)} &\overleftarrow{\kappa}\big((w_1,c_1),(w_2,c_2)\big) \\
&= \lim_{t\to\infty} \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_{t-i} = (w_2,c_2), S^\ell_{t-i+1}, \dots, S^\ell_{t-1} \notin \mathcal{W}, X_t = (w_1,c_1)\big) \\
&= \lim_{t\to\infty} \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_t = (w_2,c_2), \underbrace{S^\ell_{t+1}, \dots, S^\ell_{t+i-1} \notin \mathcal{W}}_{=:\bigstar}, X_{t+i} = (w_1,c_1)\big) \\
&= \lim_{t\to\infty} \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_{t+i} = (w_1,c_1), \bigstar \,\big|\, X_t = (w_2,c_2)\big)\mathbb{P}(X_t = (w_2,c_2)) \\
&= p_{(w_2,c_2)}\kappa\big((w_2,c_2),(w_1,c_1)\big)\,.
\end{aligned}$$

□

Proof of Theorem 4.12. The forward direction is true by Definition 4.6: If the expected clump size is finite, then $\mathcal{W}$ is not degenerate.

Now, let $\mathcal{W}$ be non-degenerate. Every clump can be uniquely decomposed into a sequence of overlapping occurrences of words from $\mathcal{W}$: A clump of size z starts with a word-context pair $x_1 = (w_1, c_1) \in \mathcal{X}$, and makes $z-1$ transitions to following word-context pairs $x_j = (w_j, c_j)$. The transition probabilities are given by the corresponding entries of K. The clump ends with a word-context pair $x_z = (w_z, c_z)$. In equilibrium, we get

$$\lim_{i\to\infty} \mathbb{P}(Z_i = z) = \sum_{x_1} \cdots \sum_{x_z} p^{\mathit{start}}_{x_1} k_{x_1,x_2} \cdots k_{x_{z-1},x_z} f_{x_z} = \langle p^{\mathit{start}} | K^{z-1} | f \rangle\,,$$

and therefore

$$\begin{aligned}
\psi = \lim_{i\to\infty} E(Z_i) &= \sum_{z=1}^{\infty} z \cdot \langle p^{\mathit{start}} | K^{z-1} | f \rangle = \langle p^{\mathit{start}} | \Big(\sum_{z=1}^{\infty} z K^{z-1} \Big) | f \rangle \\
&\overset{\text{(i)}}{=} \langle p^{\mathit{start}} | \left(\mathbf{1} - K\right)^{-2} | f \rangle \\
&\overset{\text{(ii)}}{=} \langle p^{\mathit{start}} | \left(\mathbf{1} - K\right)^{-1} | 1 \rangle\,,
\end{aligned} \tag{4.5}$$

where (i) holds due to Lemma A.7 as the spectral radius of K is smaller than one by Lemma 4.10 and (ii) follows from Equation (4.4). We now rewrite $|p^{\mathit{start}}\rangle$:

$$\begin{aligned}
p^{\mathit{start}}_{(w,c)} &= \lim_{t\to\infty} \mathbb{P}\left(X_t = (w,c) \,\middle|\, S^\ell_t \in \mathcal{W}, S^\ell_{t-1}, \dots, S^\ell_{t-\ell+1} \notin \mathcal{W}\right) \\
&= \lim_{t\to\infty} \frac{\mathbb{P}\left(X_t = (w,c),\, S^\ell_{t-1}, \dots, S^\ell_{t-\ell+1} \notin \mathcal{W}\right)}{\mathbb{P}\left(S^\ell_t \in \mathcal{W},\, S^\ell_{t-1}, \dots, S^\ell_{t-\ell+1} \notin \mathcal{W}\right)}
\end{aligned}$$

$$= \lim_{t\to\infty} \frac{\mathbb{P}\left(S^\ell_{t-1},\dots,S^\ell_{t-\ell+1} \notin \mathcal{W} \,\middle|\, X_t=(w,c)\right)\mathbb{P}\left(X_t=(w,c)\right)}{\sum_{(w_1,c_1)} \mathbb{P}\left(S^\ell_{t-1},\dots,S^\ell_{t-\ell+1} \notin \mathcal{W} \,\middle|\, X_t=(w_1,c_1)\right)\mathbb{P}\left(X_t=(w_1,c_1)\right)}$$

$$= \frac{\left(1-\sum_{(w',c')} \overleftarrow{k}_{(w,c),(w',c')}\right)\cdot p_{(w,c)}}{\sum_{(w_1,c_1)}\left(1-\sum_{(w_2,c_2)} \overleftarrow{k}_{(w_1,c_1),(w_2,c_2)}\right)\cdot p_{(w_1,c_1)}} \tag{4.6}$$

$$\overset{\text{(iii)}}{=} \frac{p_{(w,c)} - \sum_{(w',c')} p_{(w',c')} k_{(w',c'),(w,c)}}{\sum_{(w_1,c_1)}\left(p_{(w_1,c_1)} - \sum_{(w_2,c_2)} p_{(w_2,c_2)} k_{(w_2,c_2),(w_1,c_1)}\right)}, \tag{4.7}$$

where (iii) follows from Lemma 4.15. Thus

$$\langle p^{start}| = \frac{\langle p|(\mathbf{1}-K)}{\langle p|(\mathbf{1}-K)|1\rangle} = \frac{\langle p|(\mathbf{1}-K)}{1-\langle p|K|1\rangle}.$$

The proof is completed by combining the above expression with (4.5) and noting that $\langle p|1\rangle = 1$, since p is a probability distribution. □

Weighted Patterns

When a motif is considered jointly with its reverse complement, the set $\mathcal{W}$ may need to be extended to a multiset: When a motif is palindromic, it equals its reverse complement and must therefore be counted twice. See Section 4.1 for an example. We represent such a multiset by augmenting $\mathcal{W}$ with a weight function $\nu : \mathcal{W} \to \mathbb{N}$, where $\nu(w)$ gives the multiplicity of the word $w \in \mathcal{W}$.

Definition 4.16 (Weight vector and expected clump weight)**.** We define the *weight vector* $|v\rangle \in \mathbb{R}^{|\mathcal{X}|}$ by $v_{(w,c)} := \nu(w)$ for $(w,c) \in \mathcal{X}$. For a given weight vector $|v\rangle$, the random variable Y_i denotes the weight of the i-th clump, i.e. the sum of the weights of the words forming the clump. The *expected clump weight* is defined as $\psi^v := \lim_{i\to\infty} \mathbb{E}(Y_i)$. ◊

Theorem 4.17 (Expected clump weight)**.** *Given a non-degenerate word set $\mathcal{W} \subset \Sigma^\ell$ and a weight vector $|v\rangle$, the expected clump weight is*

$$\psi^v = \langle p|v\rangle \cdot \psi = \frac{\langle p|v\rangle}{1-\langle p|K|1\rangle} < \infty.$$

The idea of the following proof is to combine a scaling and a homogeneity argument: The (asymptotic) expected number of occurrences of $\mathcal{W}$ per text character, say $\mu = \langle p|1\rangle \cdot \mu$, changes to $\langle p|v\rangle \cdot \mu$ when weights are assigned, as p is the equilibrium distribution restricted to $\mathcal{W}$. In equilibrium, all clumps have the same stochastic properties, and the distribution of words from $\mathcal{W}$ in clumps is p as well, because by definition words only occur in clumps.

Proof. Let N_t be the random variable giving the number of clumps that start before position t. Asymptotically, the expected number of pattern occurrences and the number

of clumps is proportional to the text length. Therefore,

$$\begin{aligned}
0 < \lim_{t\to\infty} \frac{1}{t}\mathbb{E}\left(\sum_{i=0}^{N_t} Y_i\right) &= \lim_{t\to\infty} \frac{1}{t}\mathbb{E}\left(\sum_{t'=0}^{t-1}\sum_{w\in\mathcal{W}} [\![S_{t'}^{\ell} = w]\!] \cdot \nu(w)\right) \\
&= \lim_{t\to\infty} \frac{1}{t}\sum_{t'=0}^{t-1}\sum_{w\in\mathcal{W}} \mathbb{P}\left(S_{t'}^{\ell} = w\right) \cdot \nu(w) \\
&= \langle p|v\rangle \lim_{t\to\infty} \frac{1}{t}\sum_{t'=0}^{t-1} \mathbb{P}\left(S_{t'}^{\ell} \in \mathcal{W}\right) \\
&= \langle p|v\rangle \lim_{t\to\infty} \frac{1}{t}\mathbb{E}\left(\sum_{i=0}^{N_t} Z_i\right) < \infty\,.
\end{aligned}$$

It follows that

$$\psi^v \cdot \lim_{t\to\infty} \frac{\mathbb{E}(N_t)}{t} = \langle p|v\rangle\psi \cdot \lim_{t\to\infty} \frac{\mathbb{E}(N_t)}{t}$$

and hence $\psi^v = \langle p|v\rangle\psi$. The claim now follows from Theorem 4.12. □

To sum up, we have found a general formula for the expected clump size with respect to arbitrary finite-memory text models that holds for sets of patterns in the unweighted as well as in the weighted case. The formula is short and involves no laborious operations like matrix inversions, which is in contrast to constructions resulting from generating functions like that of Bassino et al. (2008). Besides its theoretical value, this will prove useful when we derive bounds on the p-values of motifs in Chapter 5.

4.3 Distribution of Clump Sizes

To construct a compound Poisson approximation to the exact distribution of occurrence counts, we need the whole clump size distribution, not only its expectation. In this section, we see how clump size distributions (see Figure 4.2) can be computed using PAAs. They are defined formally in the following.

Definition 4.18 (Clump size distribution)**.** Let a non-empty, non-degenerate word set $\mathcal{W} \subset \Sigma^{\ell}$ be given. By Definition 2.18, the distribution of text model states $\mathcal{L}(C_t)$ converges to an equilibrium when t goes to infinity. Therefore, the distributions of clump sizes $\mathcal{L}\big(Z_i^{\mathcal{W}}\big)$ converge as well. The limit

$$\Psi_{\mathcal{W}} := \lim_{i\to\infty} \mathcal{L}\left(Z_i^{\mathcal{W}}\right) \tag{4.8}$$

is called *the asymptotic clump size distribution of* $\mathcal{W}$. ◊

4.3.1 Size Distribution of the First Clump

Before we compute the distribution of the size of an asymptotic clump, that is, $\Psi = \lim_{i\to\infty} \mathcal{L}(Z_i)$, we consider the easier case of computing the size distribution of the first clump $\mathcal{L}(Z_0)$.

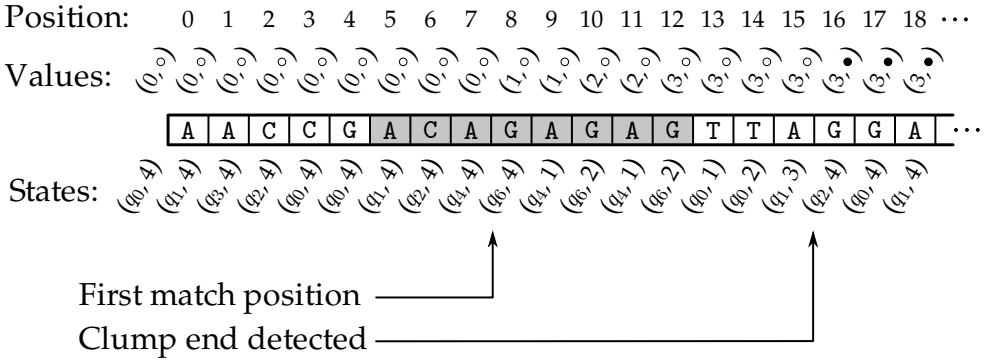


Figure 4.3: Values and states attained by a DAA processing the shown text which contains a clump (shaded gray) of three occurrences of the pattern ANAG. The DAA is based on the DFA shown in Figure 2.5(b). In particular, q_6 is the DFA's only accepting state. Note that, as soon as a value $(\cdot, \bullet)$ has been reached, the clump has ended.

Definition 4.19 (Clump size DAA)**.** Let $\mathcal{W} \subset \Sigma^\ell$ be a non-empty set of words. Consider the minimal DFA $(\mathcal{Q}^{DFA}, \Sigma, \delta^{DFA}, q_0^{DFA}, F)$ accepting all words $\Sigma^*\mathcal{W}$ and define

$$\mathit{ClumpSizeDAA}(\mathcal{W}) := \left(\mathcal{Q}^{DAA}, q_0^{DAA}, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}}\right)$$

by

- $\mathcal{Q}^{DAA} = \mathcal{Q}^{DFA} \times \{1, \ldots, \ell\}$,
- $q_0^{DAA} = (q_0^{DFA}, \ell)$,
- $\delta : \big((q,j),\sigma\big) \mapsto \begin{cases} \big(\delta^{DFA}(q,\sigma), 1\big) & \text{for } q \in F\,, \\ \big(\delta^{DFA}(q,\sigma), \min\{\ell, j+1\}\big) & \text{otherwise}\,, \end{cases}$
- $\mathcal{V} = \mathbb{N}_0 \times \{\circ, \bullet\}$,
- $v_0 = (0, \circ)$,
- $\mathcal{E} = \{0, 1\}$,
- $\eta_{(q,j)} = [\![q \in F]\!]$,
- $\theta_{(q,j)} : \big((k,f),e\big) \mapsto \begin{cases} (k+e, \circ) & \text{if } j < \ell \text{ and } f = \circ, \\ (e, \circ) & \text{if } j = \ell \text{ and } k = 0, \\ (k, \bullet) & \text{otherwise.} \end{cases}$

◊

This DAA is defined such that it is in state (q, j) when the underlying DFA is in state q and the last match position has been seen j steps ago. Here, the current position is excluded, that means j is never zero. Furthermore, j cannot be larger than ℓ and $j = \ell$ means that the last match has been seen *at least* ℓ steps ago. The operations are defined such that the first component of the current value reflects the size of the first

Table 4.4: Case distinction showing that operations $\theta_{(q,j)}$ given in Definition 4.19 behave correctly.

Conditions			Explanation
$j < \ell$	$k = 0$		Impossible as $j < \ell$ implies that a state $q \in F$ has been visited, but then k would also have been incremented.
$j < \ell$	$k > 0$	$f = \circ$	As $k > 0$, a clump has started and, as $f = \circ$, it has not ended yet, thus we add the current emission to the counter k.
$j < \ell$	$k > 0$	$f = \bullet$	First clump has ended as $f = \bullet$, do not change k.
$j = \ell$	$k = 0$	$f = \circ$	First clump has not yet begun, set k to current emission e. If $e > 0$, this is the start of the first clump.
$j = \ell$	$k = 0$	$f = \bullet$	Impossible, $k = 0$ and $f = \bullet$ are mutually exclusive.
$j = \ell$	$k > 0$	$f = \circ$	End of first clump is detected, set $f = \bullet$ and do not change k.
$j = \ell$	$k > 0$	$f = \bullet$	First clump has already ended as $f = \bullet$, do not change k.

clump and the second component indicates whether the first clump has already ended. Once it ends, the second component is changed from $\circ$ to $\bullet$ and the first component remains constant. This behavior is illustrated in Figure 4.3. The case distinction in Table 4.4 shows that the operations $\theta_{(q,j)}$ are correct, that is, that they indeed lead to the described semantics. In summary, we arrive at the following lemma.

Lemma 4.20. *Let $\mathcal{W} \subset \Sigma^\ell$ be a non-empty set of words and $D := \mathit{ClumpSizeDAA}(\mathcal{W})$ according to Definition 4.19. Then, the first component of $\mathit{value}_D(s)$ is the size of the first clump of $\mathcal{W}$ in s for any $s \in \Sigma^*$.*

Now we can use Lemma 2.35 to combine this DAA and a text model into a PAA which in turn allows us to compute the size distribution of the first clump by means of Algorithm 2.1.

Note that the value set $\mathcal{V}$ is infinite. As discussed in Section 2.4.2, this does not pose a problem as the range of each V_t is finite. Furthermore, for many applications it is sufficient to truncate the clump size distribution and use the value set $\mathcal{V}' := \{0, \dots, M\} \times \{\circ, \bullet\}$ along with adapted operations θ'_q. Algorithm 2.1 iteratively computes the joint distribution $\mathcal{L}(Q_t, V_t)$ for growing t. The operations θ_q are defined such that once a value $(\cdot, \bullet)$ is attained, it cannot change any more. That means the intermediate clump size distribution given by

$$k \mapsto \sum_{q \in \mathcal{Q}} \mathbb{P}\big(Q_t = q, V_t = (k, \bullet)\big)$$

might be returned as the result once the total error given by

$$\sum_{q \in \mathcal{Q}} \sum_{k=0}^{M} \mathbb{P}\big(Q_t = q, V_t = (k, \circ)\big)$$

drops below an accuracy threshold. The number of necessary steps until it reaches zero, however, is bounded by $\mathcal{O}(M \cdot \ell)$, because a clump containing M matches can have a length of at most $\mathcal{O}(M \cdot \ell)$. In total, we need $\mathcal{O}(M^2 \cdot \ell^2 \cdot |\mathcal{Q}^{DFA}| \cdot |\Sigma| \cdot |\mathcal{C}|^2)$ operations to compute the exact clump size distribution as follows from Lemma 2.37. Again, a factor of $|\mathcal{C}|$ can be saved if for all $c \in \mathcal{C}$ and $\sigma \in \Sigma$, there exists at most one $c' \in \mathcal{C}$ such that $\varphi(c, \sigma, c') > 0$.

Remark 4.21 (Weighted case)**.** In Section 4.2 we generalized our results on the expected clump size to *weighted* pattern sets, that is, to sets $\mathcal{W}$ equipped with a weight function $\nu : \mathcal{W} \to \mathbb{N}$. We now do the same for the clump size distribution. As in Section 2.6, we can now construct a *counting DFA* (instead of a plain DFA) that respects these multiplicities. That means each DFA state q is associated with an emission $\eta_q \in \mathbb{N}_0$ to be counted when entering q. The DAA we construct can then directly be endowed with the same emissions η_q. Its other components remain unchanged (compared to Definition 4.19). △

4.3.2 Size Distribution of Asymptotic Clumps

In order to obtain the size distribution Ψ of asymptotic clumps (see Definition 4.18), it is helpful to take a closer look at the start of clumps. More precisely, we call the position of a match's last character *match position*. In the unweighted case, where all occurrences of words from $\mathcal{W}$ that constitute a clump contribute a value of one to the clump size, the number of match positions equals the clump size. Now, we consider the first match position in a clump. It exists in every clump as, by definition, a clump consists of at least one occurrence of a word from $\mathcal{W}$.

Example 4.22 (Match positions in a clump)**.** The clump shown in Figure 4.3 has a size of three and therefore contains three match positions. They are located at indices 8, 10, and 12, where the DAA is in states $(q_6, 4)$, $(q_6, 2)$, and $(q_6, 2)$, respectively. Thus, the underlying DFA is in its only accepting state q_6. Moreover, the second component of these states indicate the distance to the last match position, where a value of $4 = \ell$ is to be interpreted as "4 or more". Therefore, the DAA is in state $(q_6, 4)$ at the first match position of every clump. △

As we shall see, the characteristics of a clump are determined by the DAA state at its first match position. Therefore, we formally characterize the DAA states that are active at first match positions of clumps.

Lemma 4.23 (DAA states at first match positions)**.** *Let a non-empty set of words $\mathcal{W} \subset \Sigma^\ell$ be given and let $D = \big(\mathcal{Q}^{DAA}, q_0^{DAA}, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q \in \mathcal{Q}}, (\theta_q)_{q \in \mathcal{Q}}\big)$ be a DAA constructed*

according to Definition 4.19 and $(\mathcal{Q}^{DFA}, \Sigma, \delta^{DFA}, q_0^{DFA}, F)$ *be the underlying DFA. Recall that* $\mathcal{Q}^{DAA} = \mathcal{Q}^{DFA} \times \{1, \dots, \ell\}$ *and define*

$$\mathcal{Q}^{CS} := \left\{ (q^{DFA}, j) \in \mathcal{Q}^{DAA} \,\middle|\, q^{DFA} \in F,\, j = \ell \right\}.$$

Then, D is in a state $(q^{DFA}, \ell) \in \mathcal{Q}^{CS}$ *if and only if it is at a first match position of a clump (CS stands for* **c***lump* **s***tart).*

Note that this lemma applies not only to the first match position of the first clump, but to the first match positions of *all* clumps.

Proof. The claim follows directly as, by definition of the DFA, $q^{DFA} \in F$ if and only if the automaton is at a match position and, by definition of the DAA's transition function, $j = \ell$ if and only if the previous match position has been seen at least ℓ position ago or no match position at all has been seen so far. □

Now consider the PAA constructed from the DAA and a text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ using Lemma 2.35. Its state space is given by

$$\mathcal{Q} = \mathcal{Q}^{DAA} \times \mathcal{C}.$$

Hence, all PAA states $\left(q^{DAA}, c\right) \in \mathcal{Q}$ with $q^{DAA} \in \mathcal{Q}^{CS}$ correspond to first match positions of clumps. The key to computing Ψ is the observation that the size distribution $\mathcal{L}(Z_i)$ of clump i solely depends on the PAA state at its first match position. More precisely, given a $q \in \mathcal{Q}$, the distribution $\mathcal{L}\big(Z_i \big| Q_{T_i} = q\big)$ is the same for all i, where T_i is the first match position of clump i and Q_t the PAA state after step t as usual. This follows directly from Definition 2.22.

The idea now is to compute the state distribution at clump start positions, denoted $\gamma : \mathcal{Q} \to [0, 1]$. It is formally given by

$$\gamma : \left(q^{DAA}, c\right) \mapsto \lim_{t \to \infty} \mathbb{P}\left(Q_t = \left(q^{DAA}, c\right) \,\middle|\, Q_t \in \left\{ (q', c') \in \mathcal{Q} \,\middle|\, q' \in \mathcal{Q}^{CS} \right\} \right). \tag{4.9}$$

Assuming that the state distribution $\mathcal{L}(Q_t)$ converges to an equilibrium (see Remark 4.25 below), the limit exists and γ is well defined.

Remark 4.24 (Connection between γ and $\left|p^{start}\right\rangle$)**.** In spirit, γ is similar to $\left|p^{start}\right\rangle$ as given in Definition 4.11 on Page 72. On the one hand, the component of $\left|p^{start}\right\rangle$ that is indexed by (w, c) gives the probability that w is the first word in a clump and before generating w, the text model is in state c. On the other hand, $\gamma(q^{DAA}, c)$ is the probability that the DAA is in state q^{DAA} and the text model is in state c after generating the first match position in a clump. △

To compute γ, we need the PAA's equilibrium distribution $\lim_{t \to \infty} \mathcal{L}(Q_t)$. This, in turn, can be computed by solving an eigenvalue problem or by iteratively computing $\mathcal{L}(Q_t)$ for growing values of t until convergence to desired precision. In practice, the distribution converges rapidly and the computation is fast.

Once γ is known, Ψ can be obtained by slightly modifying the PAA: We replace the start state by a new state q_0' and redefine the transition function

$$T : (q, q') \mapsto \begin{cases} \gamma(q') & \text{if } q = q_0' \,, \\ T(q, q') & \text{otherwise} \,. \end{cases}$$

Now the PAA simulates an asymptotic clump and Ψ can be computed using Algorithm 2.1 as explained in the previous section.

Remark 4.25 (Convergence of PAA)**.** It is a classical result of Markov chain theory that irreducibility and aperiodicity are sufficient for convergence to a unique equilibrium distribution. Refer to Brémaud (1999) for a detailed introduction. In our case, the PAA's state set has, by construction, a subset absorbing all probability mass for which both properties are fulfilled. That means the state process converges to an equilibrium in which only the states in this subset have non-zero probabilities. This can be verified by noting that the DAA constructed from a minimal DFA accepting the set $\Sigma^*\mathcal{W}$ already has a subset with similar properties. These cannot get lost when it is combined with a well-behaved text model (see Definition 2.18) to obtain a PAA. △

4.4 Quality of Compound Poisson Approximation

In this section, we assess the accuracy of the compound Poisson approximation in a realistic setting. We use the same space of IUPAC motifs that will be used in applications in Chapter 6. That is, the motif space $\mathcal{M}$ consists of all length-10 motifs with at most six wildcards from {R, Y, W, S, K, M}, zero wildcards from {B, D, H, V}, and at most two Ns. We randomly sample 10 000 motifs from $\mathcal{M}$ and calculate the exact distribution of occurrence counts (see Section 2.6) and its compound Poisson approximation (using clump size distributions truncated at size 30). All motifs are considered jointly with their reverse complements. A second order Markov model estimated from the human genome is used as background model.

One measure of similarity between the two probability distributions $\alpha, \beta : \mathcal{V} \to [0, 1]$ is the *total variation distance* defined by

$$d_V : (\alpha, \beta) \mapsto \frac{1}{2} \sum_{v \in \mathcal{V}} |\alpha(v) - \beta(v)| \,.$$

It is the maximal possible difference between the probabilities of an event $\mathcal{V}' \subset \mathcal{V}$ with respect to α and β, i.e. $d_V(\alpha, \beta) = \max\left\{|\alpha(\mathcal{V}') - \beta(\mathcal{V}')| \,\middle|\, \mathcal{V}' \subset \mathcal{V}\right\}$. A histogram of these distances for the 10 000 sampled motifs is shown in Figure 4.5. The mean amounts to 0.0014.

Remark 4.26 (Stein's method)**.** As surveyed by Barbour and Chryssaphinou (2001), Stein's method can be used to obtain error bounds for the total variation distance between original distribution and compound Poisson approximation. This method is applied to the distribution of occurrence counts of words in random texts by Reinert and Schbath (1998). Also refer to Lothaire (2005, Chapter 6) for a brief discussion of

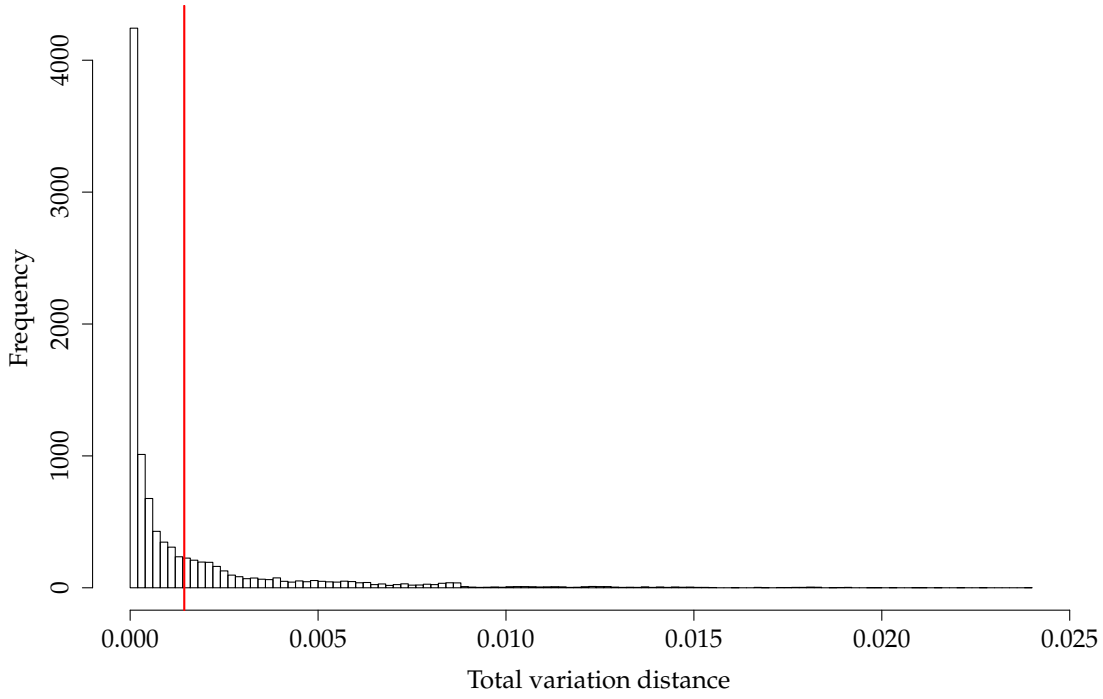


Figure 4.5: Histogram over the total variation distance between exact distribution and compound Poisson approximation for 10 000 motifs sampled from $\mathcal{M}$. Compared are the distributions of the number of occurrences with respect to a text of length 1 000 and a second order Markovian text model estimated from the human genome. The vertical red line gives the mean value.

such bounds and for further references (see Pages 348 and 349). Here, we do not further go into the topic as our main concern is that p-values are approximated accurately. The p-values encountered in motif discovery, however, are often very small. Therefore, even a tight theoretical bound on the total variation distance of, say, 0.0001 would not guarantee that p-values are accurately approximated. For instance, if the true p-value is 10^{-35} and the approximation yields 10^{-70}, we cannot be satisfied although the absolute difference is small. △

To get a more complete view of the errors made, Figure 4.6 shows boxplots of the relative errors of log-probabilities in the occurrence count distributions for 0 to 20 occurrences and random texts of length 1 000 and 10 000. For clarity, the boxplots contain only 1 000 sampled motifs. We consider log-probabilities, because the probabilities themselves range over many orders of magnitude. The probability of observing 20 matches lies in an average order of magnitude of 10^{-33} for text length 1 000 and 10^{-16} for text length 10 000. A relative error of 0.05 here means that we miss the correct order of magnitude (e.g., -33) by 5 %. We see that the relative errors increase towards the right tail of the distributions. This can be explained by observing that the length of a clump

(in terms of number of characters) is not taken into account by our approximation. When the text "gets filled up" with occurrences, the approximation becomes inaccurate. Note that 20 occurrences of length 10 would occupy up to 200 characters (depending on overlap). This is one fifth of a 1 000 character sequence. This explains why the accuracy decreases much slower towards the right tail for text length 10 000 (Figure 4.6, bottom).

It is worth noting that the occurrence count distributions are governed by an exponential decay towards the right tail. Thus, when calculating p-values (that means when summing over a distribution from a fixed k to infinity), errors do not accumulate significantly. The summands, and hence the introduced errors, rapidly become insignificantly small.

We conclude that the introduced approximation is sufficiently accurate for practical purposes and, in particular, for motif discovery.

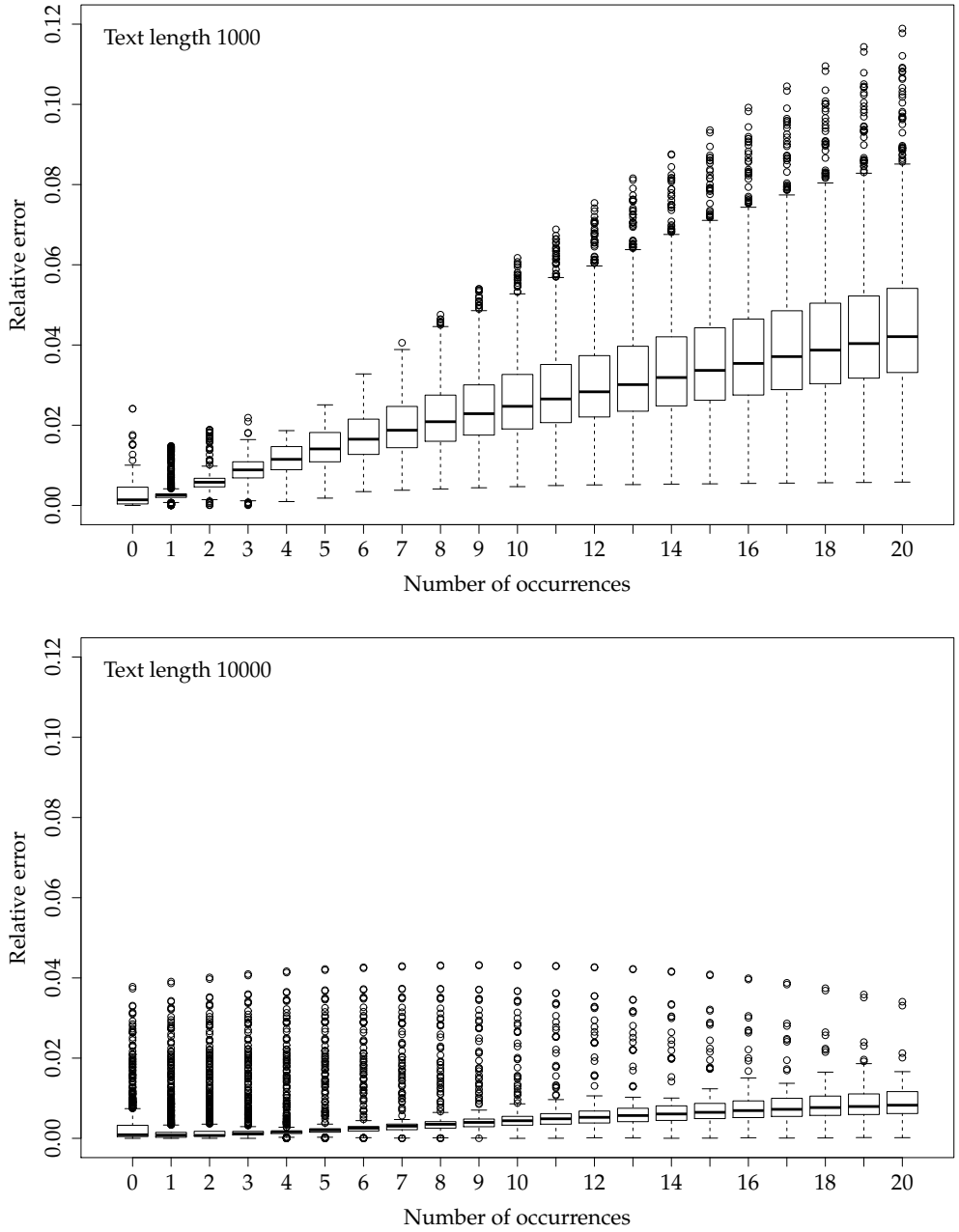


Figure 4.6: Boxplots showing the relative error of log-probabilities made by a compound Poisson approximation. **Top:** On random texts of length 1 000. The expected number of occurrences is 0.348 (averaged over all motifs). **Bottom:** On random texts of length 10 000. The expected number of occurrences is 3.504 (averaged over all motifs).

5 Motif Discovery

In this chapter, we design a motif discovery algorithm to attack Problems 1 and 2 posed in Chapter 1. We assume that the following input is given: a finite set of input sequences $\mathcal{S} \subset \Sigma^*$, a pattern length ℓ, and a finite-memory text model that we assume to be well-behaved (see Definition 2.18) and to start in equilibrium (see Remark 2.19). Our algorithm must then find a generalized string of length ℓ with an optimal p-value, either with respect to the total number of occurrences (Problem 1) or with respect to the number of sequences it occurs in (Problem 2).

The basic idea for a branch-and-bound strategy is explained in Section 5.1. Further details for the specific case of using p-values as an objective function follow in Section 5.2. As discussed earlier, applications to DNA sequences make it necessary to count motif occurrences on both strands. How the algorithm can be modified to allow this is the subject of Section 5.3.

5.1 Branch-and-Bound Algorithm

The idea of performing pattern-driven motif discovery by walking a suffix tree has long been known (Sagot, 1998). Basically, we enumerate all candidate motifs from a given motif space in lexicographic order and skip those parts of the search space that cannot contain motifs of interest. This is done by examining the suffix tree nodes that correspond to the prefixes of the current motif. If a prefix does not occur frequently enough to be interesting, all motifs sharing this prefix can be skipped.

Here, we develop algorithms that find optimal motifs of a given length ℓ, where motifs are modeled as generalized strings. That is, the considered motif space $\mathcal{M}$ for $\mathcal{G} := 2^\Sigma \setminus \{\emptyset\}$ is a subset of $\mathcal{G}^\ell$. Usually, we think of the DNA alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, but in principle, any finite alphabet can be used. By restricting $\mathcal{M}$ to a subset of $\mathcal{G}^\ell$, for example by imposing constraints on the use of wildcards, application-specific knowledge can be incorporated and faster runtimes can be achieved. In this chapter, we use the letter p to refer to a motif $p \in \mathcal{M}$ and write its length as $\ell := |p|$. Furthermore, $\mathcal{W}$ is the set of all strings that match p; that means $\mathcal{W} := \{w \in \Sigma^\ell \mid w \lhd p\}$.

In the following, we decompose the algorithm into components. This decomposition should be understood as a suggestion for a software design leading to a flexible and generic implementation.

Pattern Iterator

The first component of the algorithm is an iterator enumerating all motifs from $\mathcal{M}$ in lexicographic order. That means the pattern iterator encapsulates the specification of a motif space $\mathcal{M} \subseteq \mathcal{G}^\ell$. It must support the following operations:

- *iterator*.*next* returns the lexicographically next motif from $\mathcal{M}$,
- *iterator*.*has-next* indicates whether another motif exists,
- *iterator*.*leftmost-changed-pos* returns the index of the leftmost character different between the two motifs last returned by *iterator*.*next* (if *iterator*.*next* has so far only been called once, zero is returned),
- *iterator*.*skip-by-prefix-length*(i) skips all motifs having a prefix of length i in common with the motif returned last. That means, after successfully calling the functions *iterator*.*skip-by-prefix-length*(i) and *iterator*.*next*, the value of *iterator*.*leftmost-changed-pos* will be smaller than i.

The last two operations are needed to omit parts of the motif space in a branch-and-bound algorithm.

Index Walker

Next, we need a data structure to quickly determine the number of occurrences of a given motif in our (preprocessed) input sequences. There exists a wealth of different such *index structures* like suffix trees (see Gusfield, 1997), enhanced suffix arrays (see Abouelhoda et al., 2004), and various compressed (self-)indexes (see Ferragina et al., 2009). They all implement different tradeoffs between space usage, construction time, and query time. We do not go into the technical details of index structures but use an abstraction we call *index walker*. The idea is that an index walker represents a generalized string at any time and provides information on the number of its occurrences in the input sequences. The represented motif can be changed by appending generalized characters and by pruning characters from the right side. In case of a suffix-tree-based implementation, the internal state of an index walker corresponds to a set of active nodes. We require that an index walker supports the following operations:

- *walker*.*initialize*($\mathcal{S}$) builds an index over the set of input strings $\mathcal{S}$. The represented motif corresponds to the empty string,
- *walker*.*append*(g) appends $g \in \mathcal{G}$ to the represented generalized string,
- *walker*.*trim-to-length*(i) shortens the represented generalized string to length i by retaining its length-i prefix,
- *walker*.*occurrences* returns the number of occurrences of the represented motif in the underlying string set,
- *walker*.*string-count* returns the number of input strings the represented motif occurs in.

Objective Function

As for all optimization tasks, a score function must be given. Furthermore, in order to come up with a branch-and-bound scheme, we must be able to compute bounds on the score when only a motif prefix is given. Therefore, a software component representing the objective function must support the following operations:

- *objective*.*initialize*($\mathcal{S}$) extracts necessary information from the set of input strings $\mathcal{S}$ for later use (e.g. store the total length of sequences, estimate a text model, etc.),
- *objective*.*score*(p, k_{total}, k_{seq}) returns a score for the event that the motif p occurs k_{total} times in k_{seq} different sequences,
- *objective*.*score-bound*($p, \ell, k_{total}, k_{seq}$) returns an upper bound for the score that can be attained by a motif of length ℓ with prefix p when p occurs k_{total} times in k_{seq} different sequences.

In the following, we assume that *objective* returns scores that are to be maximized. It might, for instance, return the negative logarithm of p-values.

Algorithm

Combining pattern iterator, index walker, and an objective function, we can formulate a branch-and-bound motif discovery algorithm. The idea is that the iterator enumerates the desired motif space and the function *objective*.*score-bound* is used to determine whether a part of the search space can be skipped without missing an optimal motif. Pseudocode implementing this idea is given as Algorithm 5.1. Its correctness can be easily verified: Motifs can only be skipped by executing Lines 13 and 14, which are only reached if *objective*.*score-bound* has returned a bound that is lower than the score of the best motif discovered so far. Since all motifs not skipped are evaluated in Line 15, the optimal motif cannot be missed.

5.2 Bounding P-Values

Our aim is to use the statistical significance as an objective function. Instead of computing it exactly, we use a compound Poisson approximation, which we found to be accurate in Section 4.4. Using this approximation has two advantages. On the one hand, computing the p-value is faster in practice and, on the other hand, we are able to give bounds on the p-value knowing only a motif prefix. That is, we are able to implement the function *objective*.*score-bound* introduced in the previous section.

5.2.1 Monotonicity of P-Values

Before we compute bounds, let us recapitulate the ingredients necessary to obtain a compound Poisson approximation.

Algorithm 5.1 Find a motif with maximal score.

Input: Set of input sequences $\mathcal{S}$, an objective function, an iterator defining the motif space, a motif length ℓ, and a minimal score
Output: Best pattern and its score

MOTIF-DISCOVERY($\mathcal{S}$, *objective*, *iterator*, ℓ, *min-score*)

```
 1  objective.initialize(S)
 2  walker.initialize(S)
 3  best-score = min-score
 4  best-motif = none
 5  while iterator.has-next
 6      p = iterator.next
 7      i = iterator.leftmost-changed-pos
 8      walker.trim-to-length(i)
 9      for j = i to ℓ − 1
10          walker.append(p[j])
11          b = objective.score-bound(p[..j], ℓ, walker.occurrences, walker.string-count)
12          if b < best-score
13              iterator.skip-by-prefix-length(j + 1)
14              goto Line 5
15      score = objective.score(p, walker.occurrences, walker.string-count)
16      if score > best-score
17          best-score = score
18          best-motif = p
19  return best-motif, best-score
```

Definition 5.1 (Compound Poisson p-value for the total number of occurrences)**.** Let a motif $p \in \mathcal{M}$, a set of input sequences $\mathcal{S}$, and a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ be given. Then, the *compound Poisson p-value* of the total number of occurrences of p in $\mathcal{S}$ with respect to the text model is given by

$$\mathcal{CP}\text{-}pvalue\,(p, \mathcal{S}) \;:=\; \sum_{i=occ_p(\mathcal{S})}^{\infty} \mathcal{CP}_{\lambda_{p,\mathcal{S}},\,\Psi_p}(i) \;=\; 1 - \sum_{i=0}^{occ_p(\mathcal{S})-1} \mathcal{CP}_{\lambda_{p,\mathcal{S}},\,\Psi_p}(i)\,,$$

where $\lambda_{p,\mathcal{S}}$ is the expected number of clumps of p in a set of random texts with the same lengths as the strings in $\mathcal{S}$ and Ψ_p is the clump size distribution of p. Both quantities implicitly depend on the text model $(\mathcal{C}, c_0, \Sigma, \varphi)$. ◊

The clump size distribution Ψ_p can be computed using a PAA as detailed in Section 4.3. The expected number of clumps equals the expected number of occurrences divided by the expected clump size ψ_p. Formally, it is given by

$$\lambda_{p,\mathcal{S}} = \frac{\zeta_p}{\psi_p} \cdot \sum_{s \in \mathcal{S}} (|s| - \ell + 1)\,, \tag{5.1}$$

where

$$\zeta_p := \mathbb{E}\left(occ_p\left(S_t^\ell\right)\right) = \mathbb{P}\left(S_t^\ell \lhd p\right) = \mathbb{P}\left(S_t \cdots S_{t+\ell-1} \lhd p\right) \tag{5.2}$$

is the expected number of occurrences of p at any position t. Note that ζ_p does not depend on t as we have assumed that the text model starts in equilibrium according to Remark 2.19. It can directly be obtained by using a PAA to compute the distribution of the number of occurrences in a text of length ℓ as explained in Section 2.6. Furthermore, recall that the expected clump size ψ_p can immediately be calculated from the clump size distribution Ψ_p.

Remark 5.2 (Fast computation of ζ_p)**.** Besides using a PAA to compute ζ_p for a given text-model $(\mathcal{C}, c_0, \Sigma, \varphi)$, we can formulate recurrences directly. This has the advantage that no DFA needs to be constructed, saving time in practice. We define $z(i, c) := \mathbb{P}(S_0^i \lhd p[..i-1], C_i = c)$ and get ζ_p by marginalization over $\mathcal{C}$, i.e. $\zeta_p = \sum_{c\in\mathcal{C}} z(\ell, c)$. To compute $z(i, c)$ by dynamic programming, we set up the recurrences $z(0, c) = [\![c = c_0]\!]$ and $z(i + 1, c) = \sum_{c'\in\mathcal{C}} \sum_{\sigma\in p[i]} z(i, c')\varphi(c', \sigma, c)$. We can start in state c_0 as the text model starts in equilibrium according to Remark 2.19. This approach is not only faster and more direct than using a PAA, but also allows reusing results: when we have calculated ζ_p we just need to add another row to the z-table to get ζ_{pg} for a generalized character $g \in \mathcal{G}$. △

Definition 5.1 is analogous to Definition 1.9 but uses a compound Poisson approximation instead of the exact distribution. Both refer to the total number of occurrences. In analogy to Definition 1.11, we now formally introduce a compound Poisson version of the p-value with respect to the number of sequences that contain at least one motif occurrence. Before we define it, we make an auxiliary definition to ease notation.

Definition 5.3 (Binary distribution $\mathcal{D}$)**.** For $\lambda > 0$, define

$$\mathcal{D}_\lambda(k) := \begin{cases} e^{-\lambda} & k = 0\,, \\ 1 - e^{-\lambda} & k = 1\,. \end{cases}$$

◊

That means $\mathcal{D}_\lambda$ is a Bernoulli distribution with success probability $1 - e^{-\lambda}$. For every clump size distribution Ψ, we have $\Psi(0) = 0$ and, hence, $\mathcal{D}_\lambda(0) = \mathcal{CP}_{\lambda,\Psi}(0)$ and $\mathcal{D}_\lambda(1) = \sum_{i=1}^{\infty} \mathcal{CP}_{\lambda,\Psi}(i)$. In other words, when λ is the expected number of clumps in a random text, then $\mathcal{D}_\lambda(0)$ is the probability that it contains no motif occurrences, while $\mathcal{D}_\lambda(1)$ is the probability that it contains one or more motif occurrences.

Definition 5.4 (Compound Poisson p-value for the number of matching strings)**.** Let a motif $p \in \mathcal{M}$, a set of input sequences $\mathcal{S} = \{s_0, \ldots, s_k\}$, and a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ be given. Then, the compound Poisson p-value of the number of strings in $\mathcal{S}$ containing p with respect to the text model is given by

$$\mathit{\mathcal{CP}\text{-}pvalue}_{\mathit{seqs}}(p, \mathcal{S}) := \sum_{i=\mathit{occ\text{-}seqs}_p(\mathcal{S})}^{|\mathcal{S}|} \left(\mathcal{D}_{\lambda_{p,0}} * \ldots * \mathcal{D}_{\lambda_{p,k}}\right)(i)\,,$$

where $*$ denotes the convolution (see Section 1.3) operation and

$$\lambda_{p,j} := \frac{\zeta_p}{\psi_p} \cdot (|s_j| - \ell + 1) \tag{5.3}$$

is the expected number of clumps of p in a random text of length $|s_j|$ for $0 \leq j \leq k$. $\Diamond$

As stated formally in the next lemma, both types of p-values have in common that they depend monotonously on the quantity ζ_p/ψ_p which is contained as a factor in Equations (5.1) and (5.3). The quotient ζ_p/ψ_p can be interpreted as the expected number of clumps per character, as multiplying it with the (effective) text length $|s| - \ell + 1$ yields the expected number of clumps.

Lemma 5.5 (Monotonicity of compound Poisson p-values)**.** *Let a finite set of input sequences $\mathcal{S} = \{s_0, \ldots, s_k\} \subset \Sigma^*$, a pattern $p \in \mathcal{M}$, and a constant λ' be given such that $0 < \lambda' \leq \zeta_p/\psi_p$. Then,*

$$\mathcal{CP}\text{-}pvalue\,(p, \mathcal{S}) \geq \sum_{i=occ_p(\mathcal{S})}^{\infty} \mathcal{P}_{\lambda'_{\mathcal{S}}}(i) \tag{5.4}$$

with $\lambda'_{\mathcal{S}} := \lambda' \cdot \sum_{s \in \mathcal{S}} (|s| - \ell + 1)$ and

$$\mathcal{CP}\text{-}pvalue_{seqs}\,(p, \mathcal{S}) \geq \sum_{i=occ\text{-}seqs_p(\mathcal{S})}^{|\mathcal{S}|} \left(\mathcal{D}_{\lambda'_0} * \ldots * \mathcal{D}_{\lambda'_k}\right)(i) \tag{5.5}$$

with $\lambda'_j := \lambda' \cdot (|s_j| - l + 1)$ for $0 \leq j \leq k$.

This lemma says that we can obtain a lower bound for the p-value by providing a lower bound for ζ_p/ψ_p. Note that a lower bound for the p-value corresponds to an upper bound for a score defined as the p-value's negative logarithm and, thus, Lemma 5.5 gives us a handle to implement the function *objective.score-bound* needed for Algorithm 5.1.

Informally, Inequality (5.4) is correct as the right hand side is the probability of observing $occ_p(\mathcal{S})$ or more clumps when the expected number of clumps is $\lambda'_{\mathcal{S}}$, which is smaller than the true expected clump number. As each clump contains at least one match, the probability of observing $occ_p(\mathcal{S})$ or more clumps cannot be larger than the probability of observing $occ_p(\mathcal{S})$ or more matches. This argument is made formal in the following proof.

Proof. Recall that Ψ_p^{*j} denotes the j-fold convolution of Ψ_p with itself. Starting from

Definition 5.1, we get

$$
\begin{aligned}
\mathcal{CP}\text{-}pvalue\,(p,\mathcal{S}) &= \sum_{i=occ_p(\mathcal{S})}^{\infty} \mathcal{CP}_{\lambda_{p,\mathcal{S}},\,\Psi_p}(i) \\
&= \sum_{i=occ_p(\mathcal{S})}^{\infty} \sum_{j=0}^{\infty} \mathcal{P}_{\lambda_{p,\mathcal{S}}}(j) \cdot \Psi_p^{*j}(i) \\
&= \sum_{j=0}^{occ_p(\mathcal{S})-1} \mathcal{P}_{\lambda_{p,\mathcal{S}}}(j) \sum_{i=occ_p(\mathcal{S})}^{\infty} \Psi_p^{*j}(i) + \sum_{j=occ_p(\mathcal{S})}^{\infty} \mathcal{P}_{\lambda_{p,\mathcal{S}}}(j) \sum_{i=occ_p(\mathcal{S})}^{\infty} \Psi_p^{*j}(i) \\
&\geq \sum_{j=occ_p(\mathcal{S})}^{\infty} \mathcal{P}_{\lambda_{p,\mathcal{S}}}(j) \sum_{i=occ_p(\mathcal{S})}^{\infty} \Psi_p^{*j}(i) \\
&\overset{\text{(i)}}{=} \sum_{j=occ_p(\mathcal{S})}^{\infty} \mathcal{P}_{\lambda_{p,\mathcal{S}}}(j) \\
&\overset{\text{(ii)}}{\geq} \sum_{j=occ_p(\mathcal{S})}^{\infty} \mathcal{P}_{\lambda'_{\mathcal{S}}}(j)
\end{aligned}
$$

Equality (i) is true because clumps have, by definition, at least size one and, hence,

$$
\sum_{i=occ_p(\mathcal{S})}^{\infty} \Psi_p^{*j}(i) = 1 \qquad \text{for } j \geq occ_p\,(\mathcal{S})\ .
$$

Inequality (ii) holds due to $\lambda_{p,\mathcal{S}} \geq \lambda'_{\mathcal{S}}$ and the fact that the cumulative distribution function of a Poisson distribution is monotone in its parameter λ, completing the proof of Equation (5.4). Equation (5.5) follows directly from the fact that the cumulative distribution function of $\mathcal{D}_{\lambda_0} * \ldots * \mathcal{D}_{\lambda_k}$ is monotone in each λ_j. □

In summary, given a motif prefix p', we need to find a lower bound λ' such that $\lambda' \leq \zeta_p/\psi_p$ for all $p \in \mathcal{M}$ sharing this prefix. We approach this problem by computing a lower bound for ζ_p in Section 5.2.2 and an upper bound for the expected clump size ψ_p in Section 5.2.3.

Remark 5.6 (Caching p-value bounds)**.** The function *objective.score-bound* is called in the inner loop of Algorithm 5.1. Therefore, its implementation should be as fast as possible. Once the set of input sequences $\mathcal{S}$ is known, the right hand sides of (5.4) and (5.5) can be precomputed for different values of λ' and all reasonable values of $occ_p\,(\mathcal{S})$. We could, for example, precompute results for all values $\lambda' = i/n \cdot \lambda_{max}$ for $i \in \{1, \ldots, n\}$ and choose n according to available memory and λ_{max} such that all motifs with reasonable expectation are covered. The most conservative approach is to set $\lambda_{max} := 1/\ell$. △

5.2.2 Lower Bound for the Expected Number of Occurrences

Given a motif prefix p', the idea to compute a lower bound for ζ_p valid for all $p \in \mathcal{M}$ sharing this prefix is simple. As formalized in the following lemma, we compute $\zeta_{p'}$ and use the lowest possible continuation probability for each unknown character.

Lemma 5.7 (Lower bound for ζ_p)**.** *Let a pattern prefix $p' \in \mathcal{G}^{\ell'}$ with $\ell' \leq \ell$ and a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$ be given. Then,*

$$\zeta_{p'} \cdot \Bigg(\underbrace{\min_{c \in \mathcal{C}, \sigma \in \Sigma} \sum_{c' \in \mathcal{C}} \varphi(c, \sigma, c')}_{=:\varphi_{min}} \Bigg)^{\ell - \ell'} \leq \zeta_p$$

for all $p \in \mathcal{G}^{\ell}$ with prefix p'.

Proof. The lemma follows directly from Equation (5.2) and Lemma 2.12. □

The quantity φ_{min} does not depend on p or p' and is completely determined by the text model. Therefore, it can be precomputed and used to quickly obtain lower bounds given a motif prefix p'.

5.2.3 Upper Bound for the Expected Clump Size

The following theorem translates the results from Section 4.2 into bounds for the expected clump size of motifs. It is valid for arbitrary non-empty sets of words $\mathcal{W} \subset \Sigma^{\ell}$ that are not degenerate according to Definition 4.3. In the context of our motif discovery algorithm, the set $\mathcal{W}$ is given through a motif $p = g_0 \cdots g_{\ell-1} \in \mathcal{G}^{\ell}$ by $\mathcal{W} := \{w \in \Sigma^{\ell} \,|\, w \lhd p\}$. It is used in the next theorem, but we shall see that, in the resulting algorithm, it does not need to be constructed explicitly.

Theorem 5.8. *Let a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$, a non-empty set of words $\mathcal{W} \subset \mathcal{G}^{\ell}$, and a bound $P < 1$ be given such that*

$$B_{\mathcal{W}} := \lim_{t \to \infty} \sum_{i=1}^{\ell-1} \mathbb{P}\left(S_{t+i}^{\ell} \in \mathcal{W} \,\middle|\, S_t^{\ell} \in \mathcal{W} \right) \leq P\,. \tag{5.6}$$

Then, the expected clump size $\psi_{\mathcal{W}}$ satisfies $\psi_{\mathcal{W}} \leq 1/(1 - B_{\mathcal{W}}) \leq 1/(1 - P)$.

Here $B_{\mathcal{W}}$, and thus also P, are upper bounds for the conditional probability that a given occurrence of $\mathcal{W}$ is right-overlapped by another occurrence.

Proof. Applying the definitions of K and $|p\rangle$ (Definitions 4.9 and 4.11) and using $\mathcal{X}$ as defined on Page 71, we obtain

$$\langle p|K|1\rangle = \sum_{(w,c) \in \mathcal{X}} \sum_{(w',c') \in \mathcal{X}} p_{(w,c)}\, k_{(w,c),(w',c')}$$

$$
\begin{aligned}
&= \lim_{t\to\infty} \sum_{(w,c)} \sum_{(w',c')} \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_{t+i} = (w',c'), S_{t+i-1}^{\ell}, \dots, S_{t+1}^{\ell} \notin \mathcal{W} \,\big|\, X_t = (w,c)\big) \\
&\qquad\qquad \cdot \mathbb{P}\big(X_t = (w,c) \,\big|\, S_t^{\ell} \in \mathcal{W}\big) \\
&\le \lim_{t\to\infty} \sum_{(w,c)} \sum_{(w',c')} \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_{t+i} = (w',c') \,\big|\, X_t = (w,c)\big)\, \mathbb{P}\big(X_t = (w,c) \,\big|\, S_t^{\ell} \in \mathcal{W}\big) \\
&= \lim_{t\to\infty} \sum_{(w,c)} \sum_{(w',c')} \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_{t+i} = (w',c'),\, X_t = (w,c) \,\big|\, S_t^{\ell} \in \mathcal{W}\big) \\
&= \lim_{t\to\infty} \sum_{i=1}^{\ell-1} \mathbb{P}\big(S_{t+i}^{\ell} \in \mathcal{W} \,\big|\, S_t^{\ell} \in \mathcal{W}\big) \\
&= B_{\mathcal{W}} \le P < 1\,.
\end{aligned}
$$

As $P < 1$ implies that $\mathcal{W}$ is not degenerate according to Definition 4.3 and $x \mapsto 1/(1-x)$ is increasing, applying Theorem 4.12 yields the claimed result. □

Given only a length-ℓ' motif prefix $p' = g_0 \cdots g_{\ell'-1} \in \mathcal{G}^{\ell'}$, we derive a bound P to be used in Theorem 5.8 that is valid for all possible continuations of this prefix within $\mathcal{G}^{\ell}$. To be useful for motif discovery, fast calculation of the bound must be possible. We approach the problem by computing bounds $P_1, \dots, P_{\ell-1}$ for each possible shift i separately such that

$$
\max_{g_{\ell'},\dots,g_{\ell-1} \in \mathcal{G}} \lim_{t\to\infty} \mathbb{P}\big(S_{t+i}^{\ell} \lhd g_0 \cdots g_{\ell-1} \,\big|\, S_t^{\ell} \lhd g_0 \cdots g_{\ell-1}\big) \le P_i\,. \tag{5.7}
$$

Using Theorem 5.8, we immediately arrive at the following corollary.

Corollary 5.9. *Let a motif prefix $p' = g_0 \cdots g_{\ell'-1} \in \mathcal{G}^{\ell'}$ and bounds $P_1, \dots, P_{\ell-1}$ satisfying Inequality* (5.7) *be given. If $P := P_1 + \ldots + P_{\ell-1} < 1$, then the expected clump size ψ_p is bounded by $\psi_p \le 1/(1-P)$ for all $p \in \mathcal{G}^{\ell}$ with prefix p'.*

The difficulty now lies in finding good bounds $P_1, \dots, P_{\ell-1}$. The bounds are not helpful in Corollary 5.9 if their sum is larger than (or equal to) one. In particular, all P_i must be smaller than one.

Recall that $S_t^{\ell} := S_t \cdots S_{t+\ell-1}$. To further unclutter notation, we write $S_t^{\cdot\cdot}$ instead of S_t^k if k is clear from the context. For instance, for $h \in \mathcal{G}^*$, we write $S_t^{\cdot\cdot} \lhd h$ instead of $S_t^{|h|} \lhd h$. Now we repeatedly apply Bayes' theorem to decompose the conditional probability in Inequality (5.7).

$$
\begin{aligned}
&\mathbb{P}\big(S_{t+i}^{\ell} \lhd g_0 \cdots g_{\ell-1} \,\big|\, S_t^{\ell} \lhd g_0 \cdots g_{\ell-1}\big) \\
&= \mathbb{P}\big(S_{t+i} \lhd g_0, \dots, S_{t+i+\ell-1} \lhd g_{\ell-1} \,\big|\, S_t \lhd g_0, \dots, S_{t+\ell-1} \lhd g_{\ell-1}\big) \\
&= \prod_{j=0}^{\ell-1} \mathbb{P}\big(S_{t+i+j} \lhd g_j \,\big|\, S_t^{\cdot\cdot} \lhd g_0' \cdots g_{i+j-1}', S_{t+i+j}^{\cdot\cdot} \lhd g_{i+j} \cdots g_{\ell-1}\big)\,,
\end{aligned} \tag{5.8}
$$

where $g_k' := g_k \cap g_{k-i}$ for $0 \le k < \ell - 1$ and $g_k := \Sigma$ for $k < 0$ and $k \ge \ell$. This decomposition shows that the problem reduces to computing (bounds for) probabilities

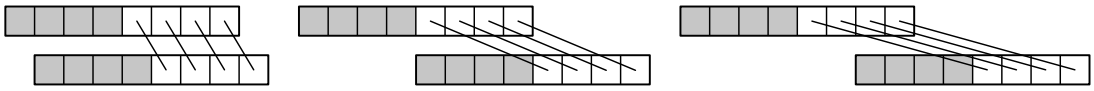


Figure 5.1: Different overlap situations of a partially known motif are illustrated. The first part of the motif (gray) is known, while the second part (white) is unknown. The diagonal lines emphasize that, although unknown, the generalized characters in the top motif are the same as the ones in the bottom motif. Left: shift of one, the aligned known IUPAC characters must be compatible. Center: shift of four, no known characters align. Right: shift of six, the known characters to the right of the previous occurrence must be present.

for generalized characters conditional on past, present, and future. That means that conditions are imposed on positions before, right at, and after the position for which we compute the probability of observing a given generalized character. To obtain bounds for these conditional probabilities, we assume that the text model allows computing meaningful bounds $P_{max}(g|g')$ for all $g, g' \in \mathcal{G}$ such that

$$\mathbb{P}(S_t \lhd g \mid S_0^{\cdot\cdot} \lhd h, S_t \lhd g', S_{t+1}^{\cdot\cdot} \lhd f) \leq P_{max}(g|g') \tag{5.9}$$

for all $h, f \in \mathcal{G}^*$ with $t := |h|$. For i.i.d. text models, where the probability of observing a character does not depend on history or future, valid bounds are directly given by $P_{max}(g|g') := \mathbb{P}(S_t \lhd g \mid S_t \lhd g')$. For arbitrary finite-memory text models, computing such bounds requires more effort, as we discuss in Section 5.2.4.

Here, we use P_{max} to construct bounds P_i for $1 \leq i < \ell$ satisfying Inequality (5.7). We start from the left hand side of this inequality, use Equation (5.8), and apply Inequality (5.9) to each factor. Again, we set $g_k := \Sigma$ for $k \geq \ell$ and obtain

$$\begin{aligned} &\max_{g_{\ell'},\ldots,g_{\ell-1} \in \mathcal{G}} \lim_{t\to\infty} \mathbb{P}\left(S_{t+i}^{\ell} \lhd g_0 \cdots g_{\ell-1} \,\middle|\, S_t^{\ell} \lhd g_0 \cdots g_{\ell-1}\right) \\ \leq &\max_{g_{\ell'},\ldots,g_{\ell-1} \in \mathcal{G}} \prod_{j=0}^{\ell-1} P_{max}(g_j|g_{i+j})\,. \end{aligned} \tag{5.10}$$

One way to further bound this quantity is to compute the maximum over each factor individually:

$$\max_{g_{\ell'},\ldots,g_{\ell-1} \in \mathcal{G}} \prod_{j=0}^{\ell-1} P_{max}(g_j|g_{i+j}) \leq \prod_{j=0}^{\ell-1} \max_{g_{\ell'},\ldots,g_{\ell-1} \in \mathcal{G}} P_{max}(g_j|g_{i+j})\,. \tag{5.11}$$

For short shifts, where many known motif characters overlap (Figure 5.1, left), this strategy works well. Expressed formally, the known characters g_i and g_{i+j} overlap if $i + j < \ell'$, which implies that $\max_{g_{\ell'},\ldots,g_{\ell-1} \in \mathcal{G}} P_{max}(g_j|g_{i+j}) = P_{max}(g_j|g_{i+j})$. For incompatible generalized characters, i.e. $g_j \cap g_{i+j} = \emptyset$, an overlap is impossible and

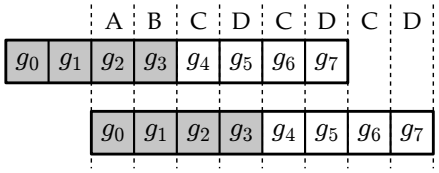


Figure 5.2: For each set of columns labeled with the same letter (A, B, C, or D), a bound can be obtained. For A and B, the bound is given by $P_{max}(g_0 \mid g_2)$ and $P_{max}(g_1 \mid g_3)$, respectively. For group C, a bound is given by $P_{max}(g_2 \mid g_4)P_{max}(g_4 \mid g_6)P_{max}(g_6 \mid \Sigma)$. For group D, it is given by $P_{max}(g_3 \mid g_5)P_{max}(g_5 \mid g_7)P_{max}(g_7 \mid \Sigma)$. Assuming that the characters $g_4, \ldots, g_7$ are unknown (white background), the maximum over all possible values of $g_4, \ldots, g_7$ must be used.

$P_{max}(g_j|g_{i+j}) = 0$. However, the strategy of using (5.11) does not work effectively for longer shifts, as shown in Figure 5.1 (center). For each position, the unknown characters can be chosen unfavorably, such that the bound for each column equals one. To obtain a meaningful bound, we have to take into account that the top and bottom motifs are the same, and therefore unknown positions cannot be chosen independently. One way to exploit this is to cleverly partition the set of columns into groups and obtain bounds for each group. As stated formally in the next lemma, these bounds can then be combined to yield a constant P_i satisfying Inequality (5.7).

Lemma 5.10 (Decomposition of Inequality (5.7))**.** *Let a length $\ell \in \mathbb{N}$, a motif prefix $p' = g_0 \cdots g_{\ell'-1} \in \mathcal{G}^{\ell'}$ with $\ell' \leq \ell$, and a shift $i \in \{1, \ldots, \ell-1\}$ be given. Furthermore, let $\mathcal{J}$ be a partition of $\{0, \ldots, \ell-1\}$, i.e. $\mathcal{J} \subset 2^{\{0,\ldots,\ell-1\}}$ such that $\bigcup_{J\in\mathcal{J}} = \{0, \ldots, \ell-1\}$ and the elements of $\mathcal{J}$ are pairwise disjoint. Let further constants P_i^J be given for all $J \in \mathcal{J}$ such that*

$$\max_{g_{\ell'},\ldots,g_{\ell-1}\in\mathcal{G}} \prod_{j\in J} P_{max}(g_j \mid g_{i+j}) \leq P_i^J \,, \tag{5.12}$$

where $g_j := \Sigma$ for $j \geq \ell$. Then, the constant $P_i := \prod_{J\in\mathcal{J}} P_i^J$ satisfies Inequality (5.7)*.*

Proof. The claim follows directly by further bounding the right hand side of Inequality (5.10):

$$\max_{g_{\ell'},\ldots,g_{\ell-1}\in\mathcal{G}} \prod_{j=0}^{\ell-1} P_{max}(g_j|g_{i+j}) \leq \prod_{J\in\mathcal{J}} \max_{g_{\ell'},\ldots,g_{\ell-1}\in\mathcal{G}} \prod_{j\in J} P_{max}(g_j|g_{i+j}) = \prod_{J\in\mathcal{J}} P_i^J = P_i \,.$$

□

One example of how to use Lemma 5.10 to obtain a bound P_2 for a shift of two is given in Figure 5.2. For each column containing two known generalized characters (columns A and B), P_{max} can be used directly. That means we set $P_2^{\{0\}} = P_{max}(g_0|g_2)$ and $P_2^{\{1\}} = P_{max}(g_1|g_3)$. The remaining columns are partitioned into groups such that

each top character is the bottom character in the next column in that group. Now we must find bounds for the groups defined in that way (groups C and D in Figure 5.2), that satisfy

$$P_2^{\{2,4,6\}} \geq \max_{g_4,g_6 \in \mathcal{G}} P_{max}(g_2|g_4) P_{max}(g_4|g_6) P_{max}(g_6|\Sigma) \tag{5.13}$$

and

$$P_2^{\{3,5,7\}} \geq \max_{g_5,g_7 \in \mathcal{G}} P_{max}(g_3|g_5) P_{max}(g_5|g_7) P_{max}(g_7|\Sigma)\,. \tag{5.14}$$

Then, we can apply Lemma 5.10 for $\mathcal{J} = \big\{\{0\},\{1\},\{2,4,6\},\{3,5,7\}\big\}$ and get

$$P_2 = P_2^{\{0\}} P_2^{\{1\}} P_2^{\{2,4,6\}} P_2^{\{3,5,7\}}\,.$$

To simplify notation, we define $P^{tel} : \mathcal{G} \times \mathcal{G}^+ \to [0,1]$ by

$$P^{tel}(g,a) := P_{max}\big(g \,|\, a[0]\big) \left(\prod_{i=1}^{|a|-1} P_{max}\big(a[i-1] \,\big|\, a[i]\big) \right) P_{max}\big(a[|a|-1] \,\big|\, \Sigma\big)$$

and call $P^{tel}(g,a)$ a ***tel**escope product*. Inequalities (5.13) and (5.14) can now be written

$$P_2^{\{2,4,6\}} \geq \max_{g_4,g_6 \in \mathcal{G}} P^{tel}(g_2, g_4 g_6) \quad \text{and} \quad P_2^{\{3,5,7\}} \geq \max_{g_5,g_7 \in \mathcal{G}} P^{tel}(g_3, g_5 g_7)\,,$$

respectively. We further define

$$P_{max}^{tel}(g) := \max_{a \in \mathcal{G}^+} P^{tel}(g,a)\,. \tag{5.15}$$

Obviously, Inequalities (5.13) and (5.14) are satisfied if we set $P_2^{\{2,4,6\}} := P_{max}^{tel}(g_2)$ and $P_2^{\{3,5,7\}} := P_{max}^{tel}(g_3)$. When the values $P_{max}^{tel}(g)$ are precomputed for all $g \in \mathcal{G}$, they can be obtained by a single table look-up. The next lemma is the key to actually computing P_{max}^{tel} and suggests that P_{max}^{tel} indeed yields bounds smaller than one.

Lemma 5.11 (Telescope bounds)**.** *$P_{max}^{tel}(g)$ is well-defined for all $g \in \mathcal{G}$. That means the maximum $\max_{a \in \mathcal{G}^+} P^{tel}(g,a)$ exists for all $g \in \mathcal{G}$. Moreover, there exists an $a_g \in \mathcal{G}^*$ with $a_g[i] \subsetneq a_g[i+1]$ for $0 \leq i < |a_g| - 1$ such that $P^{tel}(g,a_g) = \max_{a \in \mathcal{G}^+} P^{tel}(g,a) = P_{max}^{tel}(g)$.*

Proof. We prove the claim by showing that for all $g \in \mathcal{G}$ and $a \in \mathcal{G}^+$, there exists an

$$a' \in \big\{ a'' \in \mathcal{G}^* \,\big|\, a''[i] \subsetneq a''[i+1] \text{ for } 0 \leq i < |a''| - 1 \big\} \tag{5.16}$$

such that $P^{tel}(g,a) \leq P^{tel}(g,a')$. As a' is chosen from a finite set, the maximum exists and is attained for an a' from this set. First, note that $P_{max}(g|g) = 1$ for all $g \in \mathcal{G}$, which implies that we can replace all runs of the same character in a generalized string $a \in \mathcal{G}^+$ by a single occurrence of this character without changing the value of $P^{tel}(g,a)$. The function performing this replacement is denoted $r : \mathcal{G}^* \to \mathcal{G}^*$, e.g.

$r(\texttt{ACCCNNT}) = \texttt{ACNT}$. Using this notation, $P^{tel}(g, a) = P^{tel}(g, r(a))$ for all $a \in \mathcal{G}^+$. We define another operation, namely $u : \mathcal{G}^* \to \mathcal{G}^*$ defined by

$$u : a \mapsto b[0] \cdots b[|a| - 1] \quad \text{with } b[0] := a[0] \\ \text{and } b[i] := b[i-1] \cup a[i] \quad \text{for } 0 < i < |a| \,.$$

The mapping u ensures that $P^{tel}(g, a) \leq P^{tel}(g, u(a))$ because $P_{max}(g|g') \leq P_{max}(g \cup g''|g')$ and $P_{max}(g|g') \leq P_{max}(g|g' \cup g)$ for all $g, g', g'' \in \mathcal{G}$. While the former inequality is obvious, we verify the latter.

$$\begin{aligned} P_{max}(g \,|\, g') &= \max_{s,s' \in \Sigma^r} \mathbb{P}(S_r \lhd g \,|\, S_0^{\cdot\cdot} = s, S_r \lhd g', S_{r+1}^{\cdot\cdot} = s') \\ &= \max_{s,s' \in \Sigma^r} \big(1 - \mathbb{P}(S_r \lhd (\Sigma \setminus g) \,|\, S_0^{\cdot\cdot} = s, S_r \lhd g', S_{r+1}^{\cdot\cdot} = s')\big) \\ &= \max_{s,s' \in \Sigma^r} \left(1 - \frac{\mathbb{P}(S_r \lhd (g' \setminus g) \,|\, S_0^{\cdot\cdot} = s, S_{r+1}^{\cdot\cdot} = s')}{\mathbb{P}(S_r \lhd g' \,|\, S_0^{\cdot\cdot} = s, S_{r+1}^{\cdot\cdot} = s')}\right) \\ &\leq \max_{s,s' \in \Sigma^r} \left(1 - \frac{\mathbb{P}(S_r \lhd ((g' \cup g) \setminus g) \,|\, S_0^{\cdot\cdot} = s, S_{r+1}^{\cdot\cdot} = s')}{\mathbb{P}(S_r \lhd (g' \cup g) \,|\, S_0^{\cdot\cdot} = s, S_{r+1}^{\cdot\cdot} = s')}\right) \\ &= \max_{s,s' \in \Sigma^r} \mathbb{P}(S_r \lhd g \,|\, S_r \lhd (g' \cup g), S_0^{\cdot\cdot} = s, S_{r+1}^{\cdot\cdot} = s') \\ &= P_{max}(g \,|\, g' \cup g) \,. \end{aligned}$$

Given $g \in \mathcal{G}$ and $a \in \mathcal{G}^+$, we set $a' := r\big(u(a)\big)$. By definition of r and u, this a' satisfies (5.16). Furthermore, $P^{tel}(g, a) \leq P^{tel}(g, a')$, completing the proof. □

By virtue of this lemma, we can find $P_{max}^{tel}(g)$ by examining $P^{tel}(g, a)$ for all a with $a[i] \subsetneq a[i+1]$ for $0 \leq i < |a| - 1$, which we do in a preprocessing step once the text model is known.

The next lemma summarizes different types of bounds. The three cases considered in the lemma correspond to situations where (1) a known character is below another known character, which is the case for three generalized characters in the left part of Figure 5.1, (2) a known character is below an unknown character, which is the case for all four characters in the middle part of Figure 5.1, and (3) a known character is below a position not covered by the previous motif occurrence, which is the case for two known generalized characters in the right part of Figure 5.1. Furthermore, the lemma formally gives the conditions under which each bound is applicable.

Lemma 5.12. *Let a length $\ell \in \mathbb{N}$, a motif prefix $p' = g_0 \cdots g_{\ell'-1} \in \mathcal{G}^{\ell'}$ with $\ell' \leq \ell$, a shift $i \in \{1, \ldots, \ell - 1\}$, and an index $j \in \{0, \ldots, \ell' - 1\}$ be given.*

1. *(Known character below known character) If $0 \leq j < \ell' - i$, set $J := \{j\}$. Then, $P_i^J := P_{max}(g_j|g_{j+i})$ satisfies Inequality* (5.12).

2. *(Known character below unknown character) If $\max\{0, \ell' - i\} \leq j < \min\{\ell', \ell - i\}$, set $J := \big\{j' \in \{0, \ldots, \ell - 1\} \,\big|\, \exists k \in \mathbb{N}_0 : j' = j + k \cdot i\big\}$. Then, $P_i^J := P_{max}^{tel}(g_j)$ satisfies Inequality* (5.12).

3. *(Known character below position not covered by previous occurrence) If $\ell - i \leq j < \ell'$, set $J := \{j\}$. Then, $P_i^J := P_{max}(g_j|\Sigma)$ satisfies Inequality* (5.12).

Proof. In each case, we start from the left hand side of Inequality (5.12)

1.
$$\max_{g_{\ell'},\dots,g_{\ell-1}\in\mathcal{G}} \prod_{j'\in J} P_{max}(g_{j'} \mid g_{i+j'}) = \max_{g_{\ell'},\dots,g_{\ell-1}\in\mathcal{G}} P_{max}(g_j \mid g_{i+j}) \overset{\text{(i)}}{=} P_{max}(g_j \mid g_{i+j}) = P_i^J \,,$$
where (i) is true as $j < \ell'$ and $j + i < \ell'$.

2.
$$\begin{aligned} &\max_{g_{\ell'},\dots,g_{\ell-1}\in\mathcal{G}} \prod_{j'\in J} P_{max}(g_{j'} \mid g_{i+j'}) \\ &= \max_{g_{\ell'},\dots,g_{\ell-1}\in\mathcal{G}} \prod_{k=0}^{\lfloor(\ell-j-1)/i\rfloor} P_{max}(g_{j+ki} \mid g_{j+(k+1)i}) \end{aligned} \tag{5.17}$$
Note that $j + (k+1)i \geq \ell$ for $k = \lfloor(\ell - j - 1)/i\rfloor$ and recall the convention that $g_m := \Sigma$ for $m \geq \ell$. We set $g := g_j$ and $a := g_{j+1i} \cdots g_{j+\lfloor(\ell-j-1)/i\rfloor i}$. Hence, we get
$$\begin{aligned} (5.17) &= \max_{a\in\mathcal{G}^{\lfloor(\ell-j-1)/i\rfloor}} P_{max}(g \mid a[0]) \left(\prod_{j'=1}^{|a|-1} P_{max}(a[j'-1] \mid a[j']) \right) \\ &\qquad \cdot P_{max}(a[|a|-1] \mid \Sigma) \\ &= \max_{a\in\mathcal{G}^{\lfloor(\ell-j-1)/i\rfloor}} P^{tel}(g,a) \leq P_{max}^{tel}(g) = P_i^J \,. \end{aligned}$$

3.
$$\max_{g_{\ell'},\dots,g_{\ell-1}\in\mathcal{G}} \prod_{j'\in J} P_{max}(g_{j'} \mid g_{i+j'}) = \max_{g_{\ell'},\dots,g_{\ell-1}\in\mathcal{G}} P_{max}(g_j \mid g_{i+j}) \overset{\text{(ii)}}{=} P_{max}(g_j \mid \Sigma) \,,$$
where (ii) is true as $j < \ell'$ and $j + i \geq \ell$ and, thus, by convention, $g_{i+j} = \Sigma$.

□

Algorithm 5.2 shows how the different cases of Lemma 5.12 can be applied to obtain a bound P_i for each shift i and, in the end, an upper bound for the expected clump size.

Theorem 5.13. *Algorithm 5.2 is correct. That means, given a motif prefix $p' = g_0 \cdots g_{\ell'-1} \in \mathcal{G}^{\ell'}$ and tables P_{max} and P_{max}^{tel} satisfying Inequality* (5.9) *and Equation* (5.15)*, respectively, it indeed computes a bound ψ', such that $\psi_p \leq \psi'$ for all $p \in \mathcal{G}^{\ell}$ with prefix p'.*

Algorithm 5.2 Compute an upper bound for the expected clump size of a partially known motif.

Input: A motif prefix $p' = g_0 \cdots g_{\ell'-1} \in \mathcal{G}^{\ell'}$, precomputed tables P_{max} and P^{tel}_{max} satisfying Inequality (5.9) and Equation (5.15), respectively

Output: A bound ψ', such that $\psi_p \leq \psi'$ for all $p \in \mathcal{G}^{\ell}$ with the prefix $g_0 \cdots g_{\ell'-1}$

BOUND-EXPECTED-CLUMP-SIZE$\left(p', \ell, P_{max}, P^{tel}_{max}\right)$

```
1  P = 0
2  for i = 1 to ℓ − 1
3      P_i = 1.0
4      for j = 0 to |p'| − 1
5          if j + i < ℓ
6              if j + i < |p'|
7                  P_i = P_i · P_max(g_j | g_{j+i})
8              else
9                  P_i = P_i · P^tel_max(g_j)
10         else
11             P_i = P_i · P_max(g_j | Σ)
12     P = P + P_i
13 if P ≥ 1
14     return ∞
15 else
16     return 1/(1 − P)
```

Proof. In the outer loop (Lines 2 to 12), a value P_i is computed for all $i \in \{1, \ldots, \ell - 1\}$. These values are added up in Line 12 and, hence, $P = \sum_{i=1}^{\ell-1} P_i$ after the outer loop has been executed. We show that each P_i computed in Lines 3 to 11 satisfies Inequality (5.7). Then, the claim follows by Corollary 5.9. In Line 3, P_i is initialized to 1.0 and, in each iteration of the inner loop (Lines 4 to 11), it is multiplied with exactly one factor for each $j \in \{0, \ldots, |p'| - 1\}$. Analyzing the if clauses in Lines 5 and 6, we find that Line 7 is executed if $j < |p'| - i$, Line 9 is executed if $|p'| - i \leq j < \ell - i$, and Line 11 is executed if $\ell - i \leq j$. Considering that $j \in \{0, \ldots, |p'| - 1\}$, these are exactly the conditions for the three cases of Lemma 5.12 (in that order). In each case, Lemma 5.12 yields a set J and a bound P_i^J satisfying Inequality (5.12). As a set J is generated for each i and j, we refer to it as $J_{i,j}$. In order to apply Lemma 5.10 to conclude that each P_i computed in this way satisfies Inequality (5.7), we need to give a partition $\mathcal{J}_i$ of $\{0, \ldots, \ell - 1\}$ such that all produced sets $J_{i,j}$ are elements of $\mathcal{J}_i$. By construction, the sets $J_{i,0}, \ldots, J_{i,\ell'-1}$ are pairwise disjoint for all i. Thus, we set $\mathcal{J}_i' := \{J_{i,0}, \ldots, J_{i,\ell'-1}\}$ and $\mathcal{J}_i := \mathcal{J}_i' \cup \{J_{rest}\}$, where $J_{rest} := \{0, \ldots, \ell - 1\} \setminus \bigcup_{J \in \mathcal{J}_i'} J$. Observing that the trivial bound $P_i^{J_{rest}} = 1$ satisfies Inequality (5.12) completes the proof. □

Remark 5.14. The set J_{rest} may contain positions $j \in J_{rest}$ that correspond to *unknown* characters not overlapped by the previous occurrence. More precisely, it may contain

indices j with $\ell - i \leq j < \ell$. If, by definition of the motif space $\mathcal{M}$, constraints on the multiplicity of wildcard characters are known, they might be used to find an improved bound $P_i^{J_{rest}}$. When, for example, $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ is the DNA alphabet, $\mathcal{G}$ corresponds to the IUPAC alphabet, the prefix `ANNT` is known, and at most two wildcards are allowed per motif, the unknown characters must be either `A`, `C`, `G`, or `T` and the bound $\max_{\sigma \in \Sigma} P_{max}(\{\sigma\} \mid \Sigma)$ can be used for each $j \in J_{rest}$ with $\ell - i \leq j < \ell$. △

Example 5.15 (Bound for expected clump size)**.** We consider a uniform text model over the DNA alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Thus, each letter in the IUPAC alphabet corresponds to a generalized character $g \in \mathcal{G}$. Due to the uniform text model, the bounds P_{max} and P_{max}^{tel} take the particular simple form of

$$P_{max}(g \mid g') = \frac{|g \cap g'|}{|g'|} \qquad \text{and} \qquad P_{max}^{tel}(g) = |g| \cdot \frac{1}{4}$$

for all $g, g' \in \mathcal{G}$. Now assume that the motif space of interest is the set of all motifs of length eight; that means $\mathcal{M} = \mathcal{G}^\ell$ for $\ell = 8$. We use Algorithm 5.2 to compute an upper bound for the expected clump size of motifs with the prefix `ARRA` (recall that $\mathtt{R} = \{\mathtt{A}, \mathtt{G}\}$). The calculation results in the following bounds.

$$P_1 = \underbrace{P_{max}(\mathtt{A} \mid \mathtt{R})}_{=:P_1^{\{0\}}} \cdot \underbrace{P_{max}(\mathtt{R} \mid \mathtt{R})}_{=:P_1^{\{1\}}} \cdot \underbrace{P_{max}(\mathtt{R} \mid \mathtt{A})}_{=:P_1^{\{2\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{A})}_{=:P_1^{\{3,4,5,6,7\}}} = \frac{1}{2} \cdot 1 \cdot 1 \cdot \frac{1}{4} = \frac{1}{8}$$

$$P_2 = \underbrace{P_{max}(\mathtt{A} \mid \mathtt{R})}_{=:P_2^{\{0\}}} \cdot \underbrace{P_{max}(\mathtt{R} \mid \mathtt{A})}_{=:P_2^{\{1\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{R})}_{=:P_2^{\{2,4,6\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{A})}_{=:P_2^{\{3,5,7\}}} = \frac{1}{2} \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{16}$$

$$P_3 = \underbrace{P_{max}(\mathtt{A} \mid \mathtt{A})}_{=:P_3^{\{0\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{R})}_{=:P_3^{\{1,4,7\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{R})}_{=:P_3^{\{2,5\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{A})}_{=:P_3^{\{3,6\}}} = 1 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{16}$$

$$P_4 = \underbrace{P_{max}^{tel}(\mathtt{A})}_{=:P_4^{\{0,4\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{R})}_{=:P_4^{\{1,5\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{R})}_{=:P_4^{\{2,6\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{A})}_{=:P_4^{\{3,7\}}} = \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{64}$$

$$P_5 = \underbrace{P_{max}^{tel}(\mathtt{A})}_{=:P_5^{\{0,5\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{R})}_{=:P_5^{\{1,6\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{R})}_{=:P_5^{\{2,7\}}} \cdot \underbrace{P_{max}(\mathtt{A} \mid \Sigma)}_{=:P_5^{\{4\}}} = \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{64}$$

$$P_6 = \underbrace{P_{max}^{tel}(\mathtt{A})}_{=:P_6^{\{0,6\}}} \cdot \underbrace{P_{max}^{tel}(\mathtt{R})}_{=:P_6^{\{1,7\}}} \cdot \underbrace{P_{max}(\mathtt{R} \mid \Sigma)}_{=:P_6^{\{2\}}} \cdot \underbrace{P_{max}(\mathtt{A} \mid \Sigma)}_{=:P_6^{\{3\}}} = \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{64}$$

$$P_7 = \underbrace{P_{max}^{tel}(\mathtt{A})}_{=:P_6^{\{0,7\}}} \cdot \underbrace{P_{max}(\mathtt{R} \mid \Sigma)}_{=:P_6^{\{1\}}} \cdot \underbrace{P_{max}(\mathtt{R} \mid \Sigma)}_{=:P_6^{\{2\}}} \cdot \underbrace{P_{max}(\mathtt{A} \mid \Sigma)}_{=:P_6^{\{3\}}} = \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{64}$$

$$\implies P = \sum_{i=1}^{\ell-1} P_i = \frac{1}{8} + \frac{1}{16} + \frac{1}{16} + \frac{1}{64} + \frac{1}{64} + \frac{1}{64} + \frac{1}{64} = \frac{5}{16}$$

$$\implies \psi' = \frac{1}{1-P} = \frac{16}{11} \approx 1.45$$

Therefore, the computed upper bound for the expected clump size amounts to 1.45. Enumeration shows that, for all possible continuations, the largest exact expected clump size is 1.3144 (for the motif `ARRANNNN`). △

5.2.4 Bounds for Conditional Character Probabilities

In this section, we show how bounds $P_{max}(g|g')$ satisfying Inequality (5.9) can be computed for a given finite-memory text model. We focus on Markovian models and sketch how the results can be extended to arbitrary finite-memory text models in Remark 5.19 at the end of this section. Therefore, we now assume a text model to be given that is Markovian according to Definition 2.15 on Page 25. Bounds are then provided by the following lemma.

Lemma 5.16 (Bounds for Markovian text models)**.** *Let $(\mathcal{C}, c_0, \Sigma, \varphi)$ be a finite-memory text model and assume that it is Markovian of order r. For all $g, g' \in \mathcal{G}$, we define the constants*

$$P_{max}(g|g') := \max_{s,s' \in \Sigma^r} \mathbb{P}(S_r \lhd g \mid S_0^{\cdot\cdot} = s, S_r \lhd g', S_{r+1}^{\cdot\cdot} = s') \,. \tag{5.18}$$

Then,

$$\mathbb{P}(S_t \lhd g \mid S_0^{\cdot\cdot} \lhd h, S_t \lhd g', S_{t+1}^{\cdot\cdot} \lhd f) \leq P_{max}(g|g') \tag{5.19}$$

for all $g, g' \in \mathcal{G}$ and $h, f \in \mathcal{G}^*$ with $t := |h| \geq r$ and $|f| \geq r$.

The bounds $P_{max}(g|g')$ can be precomputed for all $g, g' \in \mathcal{G}$. Even for high-order models, e.g. for $r = 5$, this is possible within seconds on current hardware for the DNA alphabet ($|\Sigma| = 4$). Once obtained, they can be used to quickly bound conditional probabilities for a generalized character, no matter what constraints are imposed on history h or future f. The requirements that $|h| \geq r$ and $|f| \geq r$ simplify the proof by avoiding some case distinctions but do hardly restrict the situations the lemma can be applied in. When, for a given position $t \geq r$, known constraints on history or future are shorter than r, they can be padded with Σs such that $|h| = t$ and $|f| \geq r$. To prove the claim, we need two auxiliary lemmas that are (generalized) variants of Equation (2.8) from Definition 2.15.

Lemma 5.17 (Markovian text models: dependence on the past)**.** *For a Markovian text model of order r,*

$$\mathbb{P}\big(S_{t+r}^{\cdot\cdot} \lhd f \,\big|\, S_0^{\cdot\cdot} \lhd h, S_t^{\cdot\cdot} = s''\big) = \mathbb{P}\big(S_{t+r}^{\cdot\cdot} \lhd f \,\big|\, S_t^{\cdot\cdot} = s''\big) = \mathbb{P}\big(S_r^{\cdot\cdot} \lhd f \,\big|\, S_0^{\cdot\cdot} = s''\big)$$

for all $h, f \in \mathcal{G}^*$ and all $s'' \in \Sigma^r$, where $t := |h|$.

Proof.

$$\begin{aligned}
&\mathbb{P}\big(S_{t+r}^{\cdot\cdot} \lhd f \,\big|\, S_0^{\cdot\cdot} \lhd h, S_t^{\cdot\cdot} = s''\big) \\
&= \sum_{s \in \Sigma^{|f|} : s \lhd f} \sum_{s' \in \Sigma^t : s' \lhd h} \mathbb{P}\big(S_0^{\cdot\cdot} = s', S_{t+r}^{\cdot\cdot} = s \,\big|\, S_0^{\cdot\cdot} \lhd h, S_t^{\cdot\cdot} = s''\big) \\
&= \sum_{s \in \Sigma^{|f|} : s \lhd f} \sum_{s' \in \Sigma^t : s' \lhd h} \mathbb{P}\big(S_{t+r}^{\cdot\cdot} = s \,\big|\, S_0^{\cdot\cdot} = s', S_t^{\cdot\cdot} = s''\big) \cdot \mathbb{P}\big(S_0^{\cdot\cdot} = s' \,\big|\, S_0^{\cdot\cdot} \lhd h, S_t^{\cdot\cdot} = s''\big)
\end{aligned}$$

$$\overset{\text{(i)}}{=} \sum_{s\in\Sigma^{|f|}:s\lhd f} \mathbb{P}\big(S^{\cdot\cdot}_{t'+r} = s \,\big|\, S^{\cdot\cdot}_{t'} = s''\big) \cdot \underbrace{\sum_{s'\in\Sigma^t:s'\lhd h} \mathbb{P}\big(S^{\cdot\cdot}_0 = s' \,\big|\, S^{\cdot\cdot}_0 \lhd h, S^{\cdot\cdot}_t = s''\big)}_{=1}$$

$$= \sum_{s\in\Sigma^{|f|}:s\lhd f} \mathbb{P}\big(S^{\cdot\cdot}_{t'+r} = s \,\big|\, S^{\cdot\cdot}_{t'} = s''\big)$$

$$= \mathbb{P}\big(S^{\cdot\cdot}_{t'+r} \lhd f \,\big|\, S^{\cdot\cdot}_{t'} = s''\big)$$

for all $t' \in \mathbb{N}_0$. To get (i), we have applied the definition of Markovian text models (Definition 2.15). □

Lemma 5.18 (Markovian text models: dependence on the future)**.** *For a Markovian text model of order r,*

$$\mathbb{P}\big(S^{\cdot\cdot}_t \lhd h \,\big|\, S^{\cdot\cdot}_{t+|h|} = s, S^{\cdot\cdot}_{t+|h|+r} \lhd f\big) = \mathbb{P}\big(S^{\cdot\cdot}_t \lhd h \,\big|\, S^{\cdot\cdot}_{t+|h|} = s\big)$$

for all $t \in \mathbb{N}_0$, all $h, f \in \mathcal{G}^*$, and all $s \in \Sigma^r$.

Proof. For a shorter notation, we define the events

- $A := (S^{\cdot\cdot}_t \lhd h)$,
- $B := (S^{\cdot\cdot}_{t+|h|} = s)$, and
- $C := (S^{\cdot\cdot}_{t+|h|+r} \lhd f)$.

Then, the claim can be written $\mathbb{P}(A \,|\, B, C) = \mathbb{P}(A \,|\, B)$. By Lemma 5.17, we get $\mathbb{P}(C \,|\, A, B) = \mathbb{P}(C \,|\, B)$. The proof is completed by using this relation and applying Bayes' theorem.

$$\begin{aligned}\mathbb{P}(A \,|\, B, C) &= \frac{\mathbb{P}(A, B, C)}{\mathbb{P}(B, C)} = \frac{\mathbb{P}(C \,|\, A, B) \cdot \mathbb{P}(A, B)}{\mathbb{P}(B, C)} = \frac{\mathbb{P}(C \,|\, B) \cdot \mathbb{P}(A, B)}{\mathbb{P}(B, C)} \\ &= \frac{\mathbb{P}(C, B) \cdot \mathbb{P}(A, B)}{\mathbb{P}(B) \cdot \mathbb{P}(B, C)} = \mathbb{P}(A \,|\, B)\,.\end{aligned}$$

□

Now we are ready to prove Lemma 5.16.

Proof of Lemma 5.16. We have to show that

$$\mathbb{P}(S_t \lhd g \,|\, S^{\cdot\cdot}_0 \lhd h, S_t \lhd g', S^{\cdot\cdot}_{t+1} \lhd f) \leq \max_{s,s'\in\Sigma^r} \mathbb{P}(S_r \lhd g \,|\, S^{\cdot\cdot}_0 = s, S_r \lhd g', S^{\cdot\cdot}_{r+1} = s')$$

for all $g, g' \in \mathcal{G}$ and $h, f \in \mathcal{G}^*$ with $t := |h| \geq r$ and $|f| \geq r$. Let $h_1 := h[..t-r-1]$, $h_2 := h[t-r..]$, $f_1 := f[..r-1]$, and $f_2 := f[r..]$. We start from the left hand side.

$$\begin{aligned}&\mathbb{P}\big(S_t \lhd g \,\big|\, S^{\cdot\cdot}_0 \lhd h, S_t \lhd g', S^{\cdot\cdot}_{t+1} \lhd f\big) \\ &= \sum_{\substack{s\in\Sigma^r:s\lhd h_2 \\ s'\in\Sigma^r:s'\lhd f_1}} \mathbb{P}\big(S^{\cdot\cdot}_{t-r} = s, S_t \lhd g, S^{\cdot\cdot}_{t+1} = s' \,\big|\, S^{\cdot\cdot}_0 \lhd h, S_t \lhd g', S^{\cdot\cdot}_{t+1} \lhd f\big)\end{aligned}$$

$$
\begin{aligned}
&\leq \max_{\substack{s\in\Sigma^r: s\lhd h_2 \\ s'\in\Sigma^r: s'\lhd f_1}} \mathbb{P}\left(S_t \lhd g \,\middle|\, S_0^{\cdot\cdot} \lhd h_1, S_{t-r}^{\cdot\cdot} = s, S_t \lhd g', S_{t+1}^{\cdot\cdot} = s', S_{t+r+1}^{\cdot\cdot} \lhd f_2\right) \\
&\leq \max_{s,s'\in\Sigma^r} \mathbb{P}\left(S_t \lhd g \,\middle|\, S_0^{\cdot\cdot} \lhd h_1, S_{t-r}^{\cdot\cdot} = s, S_t \lhd g', S_{t+1}^{\cdot\cdot} = s', S_{t+r+1}^{\cdot\cdot} \lhd f_2\right) \\
&= \max_{s,s'\in\Sigma^r} \frac{\mathbb{P}\left(S_t \lhd g, S_t \lhd g', S_{t+1}^{\cdot\cdot} = s', S_{t+r+1}^{\cdot\cdot} \lhd f_2 \,\middle|\, S_0^{\cdot\cdot} \lhd h_1, S_{t-r}^{\cdot\cdot} = s\right)}{\mathbb{P}\left(S_t \lhd g', S_{t+1}^{\cdot\cdot} = s', S_{t+r+1}^{\cdot\cdot} \lhd f_2 \,\middle|\, S_0^{\cdot\cdot} \lhd h_1, S_{t-r}^{\cdot\cdot} = s\right)} \\
&\overset{\text{(i)}}{=} \max_{s,s'\in\Sigma^r} \frac{\mathbb{P}\left(S_r \lhd g, S_r \lhd g', S_{r+1}^{\cdot\cdot} = s', S_{2r+1}^{\cdot\cdot} \lhd f_2 \,\middle|\, S_0^{\cdot\cdot} = s\right)}{\mathbb{P}\left(S_r \lhd g', S_{r+1}^{\cdot\cdot} = s', S_{2r+1}^{\cdot\cdot} \lhd f_2 \,\middle|\, S_0^{\cdot\cdot} = s\right)} \\
&= \max_{s,s'\in\Sigma^r} \frac{\mathbb{P}\left(S_0^{\cdot\cdot} = s, S_r \lhd g, S_r \lhd g' \,\middle|\, S_{r+1}^{\cdot\cdot} = s', S_{2r+1}^{\cdot\cdot} \lhd f_2\right)}{\mathbb{P}\left(S_0^{\cdot\cdot} = s, S_r \lhd g' \,\middle|\, S_{r+1}^{\cdot\cdot} = s', S_{2r+1}^{\cdot\cdot} \lhd f_2\right)} \\
&\overset{\text{(ii)}}{=} \max_{s,s'\in\Sigma^r} \frac{\mathbb{P}\left(S_0^{\cdot\cdot} = s, S_r \lhd g, S_r \lhd g' \,\middle|\, S_{r+1}^{\cdot\cdot} = s'\right)}{\mathbb{P}\left(S_0^{\cdot\cdot} = s, S_r \lhd g' \,\middle|\, S_{r+1}^{\cdot\cdot} = s'\right)} \\
&= \max_{s,s'\in\Sigma^r} \mathbb{P}\left(S_r \lhd g \,\middle|\, S_0^{\cdot\cdot} = s, S_r \lhd g', S_{r+1}^{\cdot\cdot} = s'\right).
\end{aligned}
$$

For (i) and (ii), we have used Lemma 5.17 and Lemma 5.18, respectively. □

Remark 5.19 (Generalization to arbitrary finite-memory text models)**.** Lemma 5.16 introduces bounds $P_{max}(g|g')$ for Markovian text models of arbitrary order. For arbitrary finite-memory text models, a similar derivation can be used to obtain

$$
P_{max}(g|g') := \max_{\substack{c,c'\in\mathcal{C} \\ s,s'\in\Sigma^k}} \mathbb{P}(S_{t+k} \lhd g \,|\, C_t = c, S_t^{\cdot\cdot} = s, S_{t+k}\lhd g', S_{t+k+1}^{\cdot\cdot} = s', C_{t+2k+1} = c')
$$

for an arbitrary choice of $k \in \mathbb{N}_0$. Whether or not meaningful bounds, that is, bounds smaller than one, are obtained using this approach depends on the choice of k and on the text model. △

5.2.5 Quality of Bounds for the Expected Clump Size

To assess the quality of the resulting bounds for the expected clump size, we consider the (comparably) small motif space of all motifs of length eight over the restricted IUPAC alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}, \mathtt{N}\}$ that is amenable to exhaustive enumeration. For all prefixes with lengths between three and six, Figure 5.3 on Page 108 shows the ratio of bound and worst case expected clump size. Remarkably, when considering prefixes of length three and i.i.d. text models, this ratio is below 2.0 in 106 of all 125 cases. For Markovian text models, the bounds are worse, but when more than half of the motif is known, in this case five motif positions, they already assume a good quality.

To sum up, Algorithm 5.2 quickly provides good bounds for the expected clump size for partially known motifs. All components of Algorithm 5.1 are now specified, that is, our motif discovery algorithm is complete. In Chapter 6, we apply it to several data sets and discuss the algorithm's runtime in practice. Before that, we describe the modifications necessary to jointly consider occurrences on forward and reverse strand of DNA.

5.3 Reverse Complements

DNA is a double-stranded molecule (see Example 1.1). When no prior knowledge about the strand to be searched is available, we should count motif occurrences on both strands. As discussed in Section 4.1, this is equivalent to considering a motif jointly with its reverse complement, which influences its statistical properties. Refer to Figure 4.1 on Page 68 for an example of a clump of such a joint motif. As we see in this section, small modifications are sufficient to extend our motif discovery algorithm to search for motifs on both strands.

Remark 5.20. It is tempting to think that running motif discovery jointly on both strands can be done by augmenting the set of input sequences with their reverse complements. Then, however, the scoring would be wrong. Recall that a motif's p-value is the probability of finding k or more occurrences in *independent* random texts of the same lengths as the input sequences, where k is the number of occurrences in the input sequences. When the reverse complements are added, they obviously depend on the forward sequences and, thus, the independence assumption is no longer justified. △

Counting occurrences. To count the number of occurrences, we create an index structure not over the sequences in $\mathcal{S}$ alone, but over the sequences in $\mathcal{S}$ and their reverse complements. Therefore, we must pass $\mathcal{S}$ and all reverse complements to *walker.initialize* in Line 2 of Algorithm 5.1. Then, *walker.occurrences* returns the number of motif occurrences for both strands.

Computing p-values. Computing the compound Poisson p-value (see Definitions 5.1 and 5.4) of a motif p when considered jointly with its reverse complement requires us to compute its clump size distribution Ψ_p^{joint} and its expectation ζ_p^{joint}. Computing Ψ_p^{joint} has already been covered in Section 4.3: we now need to construct a *counting* DFA for the joint motif (see Remark 4.21 for further explanations), build a PAA from it, and compute Ψ using dynamic programming as usual. The expectation ζ_p^{joint} can directly be calculated as it is the sum of the expectations ζ_p and $\zeta_{p_{rc}}$, where p_{rc} is the reverse complement of p.

Avoiding duplicate computations. Algorithm 5.1 enumerates all motifs (that are not skipped) and may thus examine motifs whose reverse complements have already been tested. This is unnecessary and can be avoided by computing a motif's score (i.e., executing Lines 15 to 18) only if the motif is lexicographically smaller than or equal to its reverse complement.

Bounding p-values. As explained in Section 5.2.1, we need a lower bound for the expectation ζ_p^{joint} and an upper bound for the expected clump size ψ_p^{joint} in order to obtain a p-value bound for the joint motif. A lower bound for ζ_p^{joint} is given by the sum of lower bounds for ζ_p and $\zeta_{p_{rc}}$, which can be computed using Lemma 5.7.

For a given motif, let $\mathcal{W}_f$ and $\mathcal{W}_{rc}$ be the sets of instances of the forward and the reverse complementary motif, respectively. To produce an upper bound for the expected clump size of the joint motif, we consider the set $\mathcal{W} := \mathcal{W}_f \cup \mathcal{W}_{rc}$ and endow it with a weight function $\nu : \mathcal{W} \to \mathbb{N}$ with $\nu : w \mapsto 1 + [\![w \in \mathcal{W}_f \cap \mathcal{W}_{rc}]\!]$. Therefore, the weight function ν tells which instances are palindromic and have to be counted twice. For the IUPAC motif `AGNT`, for instance, $\mathcal{W} = \{\texttt{AGAT}, \texttt{AGCT}, \texttt{AGGT}, \texttt{AGTT}, \texttt{AACT}, \texttt{ACCT}, \texttt{ATCT}\}$ and $\nu : w \mapsto 1 + [\![w = \texttt{AGCT}]\!]$ as `AGCT` is the only string in $\mathcal{W}$ that matches `AGNT` as well as its reverse complement `ANCT`. Let $|v\rangle$ be the weight vector derived from ν according to Definition 4.16. By Theorem 4.17, the expected clump size increases by a factor of $\langle p|v\rangle$ when using the weight function ν instead of unit weights. Applying the definitions of $|p\rangle$ (see Definition 4.11) and $|v\rangle$ yields

$$\langle p|v\rangle = 1 + \lim_{t\to\infty} \mathbb{P}\big(S_t^\ell \in \mathcal{W}_f \cap \mathcal{W}_{rc} \,\big|\, S_t^\ell \in \mathcal{W}\big) \leq 1 + \lim_{t\to\infty} \mathbb{P}\big(S_t^\ell \in \mathcal{W}_{rc} \,\big|\, S_t^\ell \in \mathcal{W}_f\big)\,. \quad (5.20)$$

This quantity can be bounded by using the bounds P_{max} for each position. The resulting bound for $\langle p|v\rangle$ lies between 1 and 2. It is a measure for the degree of *palindromicity* of the motif in question.

Left to be determined is a bound for the expected clump size of $\mathcal{W}$ (without weights). To this end, we use a variant of Theorem 5.8 that takes into account that the word set $\mathcal{W}$ is given as a union of two other sets (in our case, $\mathcal{W} = \mathcal{W}_f \cup \mathcal{W}_{rc}$).

Theorem 5.21. *Let a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$, two non-empty sets of words $\mathcal{W}_1, \mathcal{W}_2 \subset \mathcal{G}^\ell$ with $\mathcal{W} := \mathcal{W}_1 \cup \mathcal{W}_2$, and a bound $P < 1$ be given such that*

$$\lim_{t\to\infty} \Bigg(\max\left\{ \sum_{i=1}^{\ell-1} \mathbb{P}\big(S_{t+i}^\ell \in \mathcal{W}_1 \,\big|\, S_t^\ell \in \mathcal{W}_1\big)\,,\ \sum_{i=1}^{\ell-1} \mathbb{P}\big(S_{t+i}^\ell \in \mathcal{W}_1 \,\big|\, S_t^\ell \in \mathcal{W}_2\big) \right\} + \max\left\{ \sum_{i=1}^{\ell-1} \mathbb{P}\big(S_{t+i}^\ell \in \mathcal{W}_2 \,\big|\, S_t^\ell \in \mathcal{W}_1\big)\,,\ \sum_{i=1}^{\ell-1} \mathbb{P}\big(S_{t+i}^\ell \in \mathcal{W}_2 \,\big|\, S_t^\ell \in \mathcal{W}_2\big) \right\} \Bigg) \leq P\,.$$

Then, the expected clump size $\psi_{\mathcal{W}}$ satisfies $\psi_{\mathcal{W}} \leq 1/(1-P)$.

Proof. We start from an intermediate result in the proof of Theorem 5.8.

$$\begin{aligned}
\langle p|K|1\rangle &\leq \lim_{t\to\infty} \sum_{(w,c)\in\mathcal{X}} \sum_{(w',c')\in\mathcal{X}} \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_{t+i} = (w',c'),\, X_t = (w,c) \,\big|\, S_t^\ell \in \mathcal{W}\big) \\
&\leq \lim_{t\to\infty} \sum_{a,b\in\{1,2\}} \sum_{\substack{(w,c)\in\mathcal{X}:\\ w\in\mathcal{W}_a}} \sum_{\substack{(w',c')\in\mathcal{X}:\\ w\in\mathcal{W}_b}} \sum_{i=1}^{\ell-1} \mathbb{P}\big(X_{t+i} = (w',c'),\, X_t = (w,c) \,\big|\, S_t^\ell \in \mathcal{W}\big) \\
&= \lim_{t\to\infty} \sum_{b\in\{1,2\}} \max_{a\in\{1,2\}} \left\{ \sum_{i=1}^{\ell-1} \mathbb{P}\big(S_{t+i}^\ell \in \mathcal{W}_b \,\big|\, S_t^\ell \in \mathcal{W}_a\big) \right\} \leq P < 1
\end{aligned}$$

Again, the existence of $P < 1$ implies that $\mathcal{W}$ is not degenerate according to Definition 4.3 and, thus, Theorem 4.12 yields the claimed result. □

When the reverse complement was not considered, we only needed to compute a bound for $\sum_{i=1}^{\ell-1} \mathbb{P}\left(S_{t+i}^{\ell} \in \mathcal{W}_f \,\middle|\, S_t^{\ell} \in \mathcal{W}_f\right)$, which is done in Algorithm 5.2. Now, we need to modify the algorithm to also take the other three cases into account. This does not involve further technical difficulties, but can be done using the same techniques as for the forward case. We name the four cases as follows.

$$\lim_{t\to\infty} \Bigg(\max \Bigg\{ \underbrace{\sum_{i=1}^{\ell-1} \mathbb{P}\left(S_{t+i}^{\ell} \in \mathcal{W}_f \,\middle|\, S_t^{\ell} \in \mathcal{W}_f\right)}_{\text{Case FF}} , \underbrace{\sum_{i=1}^{\ell-1} \mathbb{P}\left(S_{t+i}^{\ell} \in \mathcal{W}_f \,\middle|\, S_t^{\ell} \in \mathcal{W}_{rc}\right)}_{\text{Case FR}} \Bigg\} + \max \Bigg\{ \underbrace{\sum_{i=1}^{\ell-1} \mathbb{P}\left(S_{t+i}^{\ell} \in \mathcal{W}_{rc} \,\middle|\, S_t^{\ell} \in \mathcal{W}_f\right)}_{\text{Case RF}} , \underbrace{\sum_{i=1}^{\ell-1} \mathbb{P}\left(S_{t+i}^{\ell} \in \mathcal{W}_{rc} \,\middle|\, S_t^{\ell} \in \mathcal{W}_{rc}\right)}_{\text{Case RR}} \Bigg\} \Bigg) \leq P\,.$$

We now give four functions computing upper bounds for each case. The input to each function is a motif prefix $p' \in \mathcal{G}^{\ell'}$, the motif length ℓ, and precomputed tables P_{max} and P_{max}^{tel} satisfying Inequality (5.9) and Equation (5.15), respectively. The first case corresponds to Lines 1 to 12 in Algorithm 5.2.

BOUND-CASEFF$\left(p', \ell, P_{max}, P_{max}^{tel}\right)$

```
1  P_FF = 0
2  for i = 1 to ℓ − 1
3       P_i = 1.0
4       for j = 0 to |p'| − 1
5            if j + i < ℓ
6                 if j + i < |p'|
7                      P_i = P_i · P_max(p'[j] | p'[j + i])
8                 else
9                      P_i = P_i · P_max^tel(p'[j])
10           else
11                P_i = P_i · P_max(p'[j] | Σ)
12      P_FF = P_FF + P_i
13 return P_FF
```

BOUND-CASEFR$\left(p', \ell, P_{max}, P_{max}^{tel}\right)$

```
1  P_FR = 0
2  for i = 1 to ℓ − 1
3       P_i = 1.0
4       for j = max{0, ℓ − |p'| − i} to min{|p'| − 1, ℓ − i − 1}
5            P_i = P_i · P_max(p'[j] | COMPLEMENT(p'[ℓ − j − i − 1]))
6       for j = ℓ − i to |p'| − 1
7            P_i = P_i · P_max(p'[j] | Σ)
8       P_FR = P_FR + P_i
9  return P_FR
```

BOUND-CASERF$(p', \ell, P_{max}, P_{max}^{tel})$

```
1  P_RF = 0
2  for i = 1 to ℓ − 1
3       P_i = 1.0
4       for j = 0 to i − 1
5            P_i = P_i · P_max(COMPLEMENT(p'[j]) | Σ)
6       for j = ℓ + i − |p'| to |p'| − 1
7            P_i = P_i · P_max(COMPLEMENT(p'[ℓ + i − j − 1]) | p'[j])
8       P_RF = P_RF + P_i
9  return P_RF
```

BOUND-CASERR$(p', \ell, P_{max}, P_{max}^{tel})$

```
1  P_RR = 0
2  for i = 1 to ℓ − 1
3       P_i = 1.0
4       for j = 0 to |p'| − 1
5            if j < i
6                 P_i = P_i · P_max(COMPLEMENT(p'[j]) | Σ)
7            else
8                 P_i = P_i · P_max(COMPLEMENT(p'[j]) | COMPLEMENT(p'[j − i]))
9       P_RR = P_RR + P_i
10 return P_RR
```

A bound for the expected clump size is now obtained by using these four functions and combining the results with a bound for the palindrome probability, that is, a bound for the quantity $\lim_{t\to\infty} \mathbb{P}\left(S_t^\ell \in \mathcal{W}_{rc} \,\middle|\, S_t^\ell \in \mathcal{W}_f\right)$ that appears in Equation (5.20).

BOUND-EXPECTED-CLUMP-SIZE-REVCOMP$(p', \ell, P_{max}, P_{max}^{tel})$

```
1  P_palindrome = 1.0
2  for j = ℓ − |p'| to |p'| − 1
3       P_palindrome = P_palindrome · P_max(COMPLEMENT(p'[ℓ − j − 1]) | p'[j])
4  P = 0
5  P = P + max{BOUND-CASEFF(p', ℓ, P_max, P_max^tel),
                BOUND-CASEFR(p', ℓ, P_max, P_max^tel)}
6  P = P + max{BOUND-CASERF(p', ℓ, P_max, P_max^tel),
                BOUND-CASERR(p', ℓ, P_max, P_max^tel)}
7  if P ≥ 1
8       return ∞
9  else
10      return (1 + P_palindrome)/(1 − P)
```

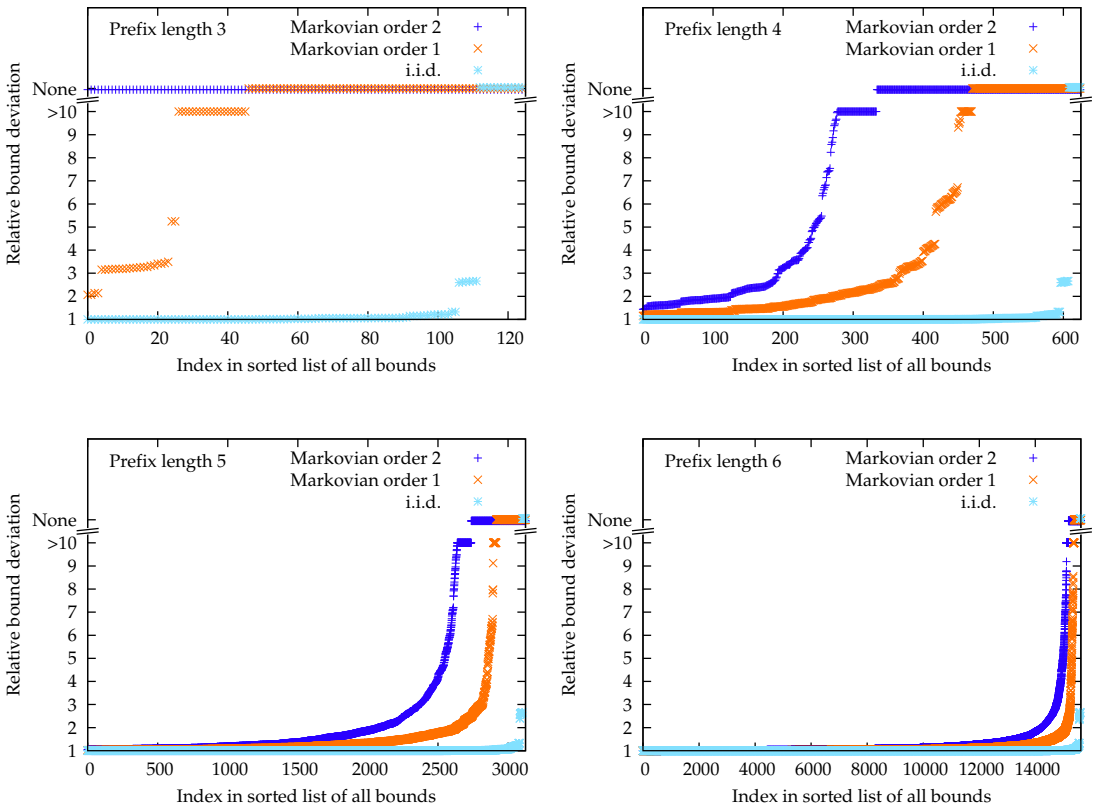


Figure 5.3: The quality of bounds for the expected clump size for motifs of length eight over the restricted IUPAC alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}, \mathtt{N}\}$ are shown. Each plot depicts the qualities for bounds derived from prefixes with a length of three, four, five, and six, respectively. For each plot, all prefixes of the respective length were enumerated and the bound deviation, defined as bound divided by worst actual expected clump size, sorted into ascending order. The worst cases were determined by exhaustive enumeration of all length-8 motifs. Each plot shows the situation for three different text models, each estimated from the human genome. For prefixes for which the computation did not yield a bound $P < 1$ to be used in Corollary 5.9, the quality is plotted at value *None* in the figures. All values larger than 10 are plotted at value ">10".

6 Applications

This chapter is devoted to the application of the motif discovery algorithm developed in the previous chapters.

In Section 6.1, we give a brief introduction to the architecture and usage of the MoSDi software which implements it. Then, MoSDi is compared to existing tools using an established publicly available benchmark suite in Section 6.2. In Sections 6.3 and 6.4, we report on new motifs discovered in the non-coding regions of *Mycobacterium tuberculosis* and human *CpG islands*, respectively.

6.1 MoSDi Software

MoSDi is a software package written in Java that contains many algorithms from the fields of **Mo**tif **S**tatistics and **Di**scovery. Its oldest parts were written in the context of the author's diploma thesis (Marschall, 2007). Since then, the package has constantly been extended and improved. Inspired by *extreme programming* introduced by Beck (1999), the idea was to implement the *simplest solutions* without unnecessary features while maintaining automated *unit tests* to improve software quality and aid code refactoring (see Fowler et al., 1999). Indeed, the source code has often been refactored when new features were added. As a result, the software design is generic and flexible. In its present version 1.2, MoSDi consists of 216 classes with 28 451 lines of source code.

The introduced mathematical constructs like DAAs, PAAs, and finite-memory text models are implemented as *abstract types* in MoSDi. The corresponding algorithms are generic as well. For instance, the tasks of computing a motif's significance (see Section 2.6) and of analyzing pattern matching algorithms (see Section 2.7) rely on the same generic implementation of Algorithm 2.1 to compute the value distribution of a PAA. This avoids duplicated source code and allows adding further applications easily. Implementing a subclass of the abstract type `DAA`, for example, immediately allows calculating the distribution of values the DAA computes on random texts distributed according to a finite-memory text model.

Moreover, dependencies between software components were reduced to a minimum. The components of a *pattern iterator*, an *index walker*, and an *objective function* described in Section 5.1, for instance, are encapsulated into classes, allowing them to be exchanged easily.

MoSDi features four *command-line tools* that make its functionality available to the user.

- `mosdi-stat` computes various pattern statistics like the occurrence count distribution, either exactly (Section 2.6) or in compound Poisson approximation

(Section 4.3), the expected clump size (Section 4.2), and empirical clump size distributions obtained by sampling.

- `mosdi-discovery` contains the main motif discovery algorithm described in Chapter 5. Furthermore, it allows refining motifs using a local search strategy.
- `mosdi-pm-analysis` computes the distribution of the number of text accesses needed by pattern matching algorithms as discussed in Section 2.7. It can also be used to compute statistics on automata sizes as shown in Table 2.9 and Figure 2.10.
- `mosdi-utils` can perform many different small tasks, for example, generate random strings, count or excise motif instances, compute a set of IUPAC strings equivalent to a PWM with a given score threshold, count q-gram occurrences in a text, compute the equilibrium expectation of all q-grams under a given text model, etc.

On invocation of each of these programs, the name of a *subcommand* must be given. Section A.1.1 contains a list of all 26 available subcommands. When called without additional parameters, each subcommand outputs a list of mandatory arguments and optional parameters. An overview of common use cases and examples of valid command lines is given in Section A.1.2. In particular, the use of MoSDi for motif discovery is explained. The examples given there can directly be adapted to reproduce the experiments described in the following sections.

6.2 Algorithm Benchmark

Designing good benchmark sets for motif discovery is not trivial. While synthetic sequences involve a somewhat arbitrary choice of background- and motif-model, real data sets are never annotated perfectly.

Tompa et al. (2005) have performed a large study comparing 13 motif discovery algorithms. They used confirmed binding site instances from the TRANSFAC database (Wingender et al., 1996) and implanted them into three different types of background; first, into their real genomic context, second, into randomly chosen promoter sequences from the same species, and, third, into random sequences. With respect to a large fraction of all considered performance statistics, the Weeder algorithm of Pavesi et al. (2004) performed best.

Later, Sandve et al. (2007) analyzed the data sets of Tompa et al. (2005) to determine whether or not common motif models are capable of separating the true binding sites from the remainder of the sequences. They point out that for most data sets this is not the case. PWMs, consensus strings with a Hamming neighborhood, and IUPAC strings all perform comparably but fail to discriminate signal from noise in many of the data sets of Tompa et al. (2005). This can have (at least) two causes. On the one hand, it might indicate that more sophisticated models are needed to capture the motifs. As discussed in Section 1.1, however, a higher model complexity entails the risk of *over-fitting*, that is, an increased chance of obtaining false positive results. On the other

hand, it might indicate that the sequence information alone is not sufficient to reliably detect transcription factor binding sites. In fact, cells use a sophisticated machinery to initiate transcription that involves more than binding of a single factor to a single binding site. In particular, transcription factors often act *cooperatively* and transcription can only be initiated when multiple factors have formed a complex. Furthermore, whether or not a gene is transcribed can depend on the chromatin structure, on distant enhancers, and DNA methylation. Refer to Chapters 6 and 7 of the book by Alberts et al. (2007) for a detailed explanation of transcription initiation and the mechanisms that influence it.

In summary, studying promoter sequences alone cannot fully explain transcriptional regulation. *If* a motif discovery algorithm has nevertheless found a common motif in a set of promoters of putatively co-regulated genes, it can help to understand the processes involved in transcription initiation. An advantage of exact motif discovery algorithms as developed in this thesis is that they allow drawing a conclusion when *no* motif has been detected. In this case, we know that no overrepresented motif in the hypothesized motif space exists and we must therefore search for another mechanism of regulation that explains the observed co-regulation.

Evaluation

To evaluate the developed algorithm, we use the benchmark suites proposed by Sandve et al. (2007). The authors generated different data sets by either implanting transcription factor binding site (TFBS) occurrences into a background generated from a third order Markov model or by extracting their original context from the respective genome. For each data set, they analyzed whether or not the motif can, in principle, be discriminated from the background using the motif models discussed above. They propose to use the data sets with good theoretical discrimination to benchmark algorithms and the rest to benchmark more powerful models. This makes the performance analysis more informative, as effects originating from motif model and algorithm are not mixed up. Consequently, we do not consider the suite of data sets intended to benchmark motif models (called *Model real*) and use the benchmark suites *Algorithm Markov* and *Algorithm real* to assess our method. For each of 50 different motifs, *Algorithm real* contains a data set with the motif's original context and *Algorithm Markov* contains a similar data set with the same number of sequences, the same sequence lengths, and the same motif instances, but the background parts of the sequences are replaced by random strings generated by a third order Markov model. The total number of characters in each data set ranges from 2 843 to 24 667 (median: 9 331) and the number of sequences per data set ranges from 5 to 18 (median: 7).

We follow Sandve et al. (2007) in choosing Weeder developed by Pavesi et al. (2004) and MEME developed by Bailey and Elkan (1994) as reference algorithms. As mentioned above, Weeder outperformed its 12 competitors with respect to most evaluated measures in the study by Tompa et al. (2005). MEME, on the other hand, is one of the most established heuristic motif discovery algorithms. Its implementation is mature and well maintained by its developers. Furthermore, Weeder and MEME represent different approaches to motif discovery. While MEME models motifs as position

weight matrices (PWMs) and optimizes them using an expectation maximization (EM) approach, Weeder is based on mismatch models and employs an exact pattern-driven search on a suffix tree. Another reason supporting this choice is that both implementations allow a background text model to be provided through a table of q-gram frequencies. This furthers a fair comparison as all tools can use the same text models and permits studying the influence of the text model order on each tool's performance.

By using publicly available data sets, future comparison to other tools is possible. In fact it is made easy as Sandve et al. (2007) provide a website[1] to which motif predictions can be uploaded and resulting performance statistics be retrieved from. For each data set, the positions of the instances of the predicted motif must be uploaded. That means that only one motif can be predicted per data set. If a tool generates more than one candidate, it must decide on one of them. In practice, a human user of motif discovery tools would probably look at more than only the top candidate, but for a benchmark study it is reasonable to require each tool to choose one candidate. Otherwise, a fair comparison of tools that make a different number of predictions becomes difficult. The *nucleotide-level correlation coefficients* (nCC) given in Table 6.1 are the values computed by the aforementioned website. The nCC is defined as follows:

$$\mathrm{nCC} := \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP}) \cdot (\mathrm{FP} + \mathrm{TN}) \cdot (\mathrm{TN} + \mathrm{FN}) \cdot (\mathrm{FN} + \mathrm{TP})}},$$

where TP, TN, FP, and FN are the numbers of true/false positive/negative predicted characters (nucleotides). The use of this measure allows an integrated assessment of sensitivity and specificity.

The benchmark suite of Sandve et al. (2007) is also used by Fauteux et al. (2008) to assess their algorithm called Seeder, which they report to perform good on the *Model real* suite. They compare it to five other algorithms: BioProspector (Liu et al., 2001), GibbsSampler (Lawrence et al., 1993), MEME (Bailey and Elkan, 1994), MotifSampler (Thijs et al., 2001), and Weeder (Pavesi et al., 2004). For *Algorithm Markov*, BioProspector performs best with an average nCC of approximately 0.12 (this value has been read off the bar chart in Figure 2 of Fauteux et al., 2008). For *Algorithm real*, they report Weeder and MEME to perform best, both with an average nCC of (approximately) 0.12.

Before we discuss the results of the present study, we describe how MEME, Weeder, and MoSDi were invoked.

MEME

MEME uses an EM approach to find PWMs with high information content. Refer to Page 11 for a brief introduction to this technique. We used MEME version 4.5.0 for all experiments. As a first step, we reproduced the results obtained by Sandve et al. (2007), who used MEME version 3.5.3. That is, we ran MEME with default settings:

```
meme.bin -dna -text <file>
```

[1] `http://tare.medisin.ntnu.no/single/single.php`

where `-dna` tells MEME to use the DNA alphabet and `-text` requests the output to be made in text format (rather than in HTML). A list of occurrences as reported by MEME was then converted to the required format and uploaded. The plots returned by the website (see Section A.4) show that the obtained scores agree almost perfectly.

Reviewing the possible parameters that can be passed to MEME, we found that the default settings are indeed appropriate for the examined data sets. By default, MEME searches for motifs with a length between 8 and 50, uses an i.i.d. background model estimated from the input data, and assumes that each sequence contains zero or one motif instances (the so called *zoops* mode). Although the data sets are created such that every sequence contains exactly one motif instance, this is not the case in most practical situations. Therefore, we decided not to provide this information to any of the three tools.

By using the option `-bsize`, MEME can be instructed to load a Markovian text model from a file. Besides running it without using this option, we ran it for text models of order one, two, and three on each data set. To do this, we computed the distribution of $(q+1)$-grams for each individual data set (where q is the text model order) and converted it into the format understood by MEME.

As detailed below, Weeder and MoSDi were run on a compute cluster under the Solaris operating system. Unfortunately, MEME could not be compiled on this platform due to a missing header file (`err.h`). Therefore, the runtimes for MEME refer to other hardware as those for Weeder and MoSDi. It was executed on an Intel Core 2 Quad CPU at 2.66 GHz with 8 GB RAM running Linux.

Weeder

Weeder falls into the class of algorithms that walk a suffix tree as discussed on Pages 16ff. It models motifs as consensus strings with a Hamming neighborhood. To achieve feasible runtimes, it imposes constraints on the placement of mismatches: each prefix of a motif instance may only contain a number of mismatches (to the consensus) that is proportional to its length. This disallows mismatches to accumulate at the beginning.

As done for MEME, we want to reproduce the results obtained by Sandve et al. (2007) using Weeder. Unfortunately, not all information needed to do this is publicly available. They write: "As Weeder requires the specification of organism, we supplied for each data set the most frequent organism." For a number of commonly-analyzed species, Weeder supplies files containing the frequencies of 6- and 8-grams in the regions upstream of genes. These data are used as a background model once the user has chosen a species. Whether or not Weeder's performance benefits from this additional knowledge (which is not available to the other tools) is not clear *a priori*. It might benefit from it, but it might also be possible that Weeder performs better when *not* using these text models but ones estimated from the input data. To shed light on this matter, Weeder was supplied with tables of the expected number of 6- and 8-grams computed from the same text models as used for MEME and MoSDi. That is, an individual text model was estimated from the input for each data set. In this sense, Weeder can also be run with Markovian text models of varying orders. Weeder was

invoked as:

```
./weederlauncher.out input.fasta MODEL (small|medium|large|extra)
```

where `MODEL` is the common prefix of the files containing the text model. One of the arguments `small`, `medium`, `large`, and `extra` must be selected to choose the size of the examined motif space. Refer to Pavesi et al. (2004) and the manual accompanying Weeder for further details on the these motif spaces. In the present evaluation, we use all combinations of `medium`, `large`, and `extra` and text model orders from zero to three (where order zero corresponds to an i.i.d. model). The optional argument `T1` that is supposed to instruct Weeder to report only one (the best) motif was tried but its use led to Weeder reporting no motifs at all but the message "Sorry! No advice on this one" for all data sets.

The output was parsed and the topmost reported motif used as prediction. Weeder reports two sets of motif instances called "all" and "best". As using "all" consistently produced worse results, only the "best" instances were used.

Weeder was executed on a compute cluster where it could be run in parallel for each of the 100 data sets. All machines run the Solaris operating system. They are partly equipped with AMD Opteron CPUs and partly equipped with Intel Xeon CPUs. Their clock rates range from 1.8 to 2.6 GHz. The runtimes reported in Table 6.1 are averaged over all data sets and, thus, also averaged over the different CPUs the jobs were performed on. In other words, the right column of Table 6.1 gives the average time needed to search one of the hundred data sets on one average CPU core.

MoSDi

MoSDi was used to solve Problem 2 introduced on Page 10 for each data set. That means the p-value for the number of sequences a motif occurs in was optimized. The runtime strongly depends on the searched space of motifs. Therefore, experiments using different motif spaces were performed in order to investigate whether the longer runtimes for more complex motif spaces are justified by better prediction quality. The wildcard characters contained in the IUPAC alphabet can be classified according to the number of matched characters: an `N`, for instance, matches four different characters, while a `Y` matches two, and a `T` matches only one (see the Table in Section A.2). The four resulting character classes are

- `A`, `C`, `G`, `T`, which each match one character,
- `R`, `Y`, `S`, `W`, `K`, `M`, which each match two characters,
- `B`, `D`, `H`, `V`, which each match three characters, and
- `N`, which matches four characters.

MoSDi allows the user to input constraints on the number of allowed characters from each of these classes. To search the space of all IUPAC motifs of length twelve with at most twelve regular characters, two wildcards from the second group, zero wildcards from the third groups, and at most six `N`s, we call

Table 6.1: A comparison of Algorithm 5.1 (optimizing $\mathcal{CP}\text{-}pvalue_{seqs}$), Weeder, and MEME on the benchmark suites proposed by Sandve et al. (2007) is shown. Each of the two suites *Algorithm Markov* and *Algorithm real* contains 50 data sets. Nucleotide-level correlation coefficient (nCC) averaged over all data sets for each of these suites is given. Columns with headings 0.25, 0.5, and 0.75 give the number of data sets with an nCC equal to or above these values. The rightmost column gives the runtimes per data set (averaged over all 100 data sets). Algorithm configuration is given in parentheses. For details, refer to the main text. (*) MEME was run on different hardware (see main text).

Algorithm	Model order	Alg. Markov (nCC) avg.	0.25	0.5	0.75	Alg. real (nCC) avg.	0.25	0.5	0.75	Runtime h:m:s
MoSDi (10,6,0,2)	0	0.092	9	5	1	0.079	7	4	0	0:12:09
MoSDi (10,6,0,2)	1	0.107	10	5	0	0.123	11	5	0	1:02:11
MoSDi (10,6,0,2)	2	0.129	11	6	0	0.164	15	8	1	17:38:46
MoSDi (12,2,0,6)	0	0.090	9	4	0	0.132	11	5	2	0:11:03
MoSDi (12,2,0,6)	1	0.108	9	5	1	0.184	15	8	2	0:33:09
MoSDi (12,2,0,6)	2	0.162	14	9	2	0.198	17	7	2	3:46:12
MoSDi (14,0,0,7)	0	0.066	4	3	1	0.086	9	2	0	0:00:50
MoSDi (14,0,0,7)	1	0.072	6	4	1	0.137	12	6	2	0:01:40
MoSDi (14,0,0,7)	2	0.086	7	5	1	0.134	11	5	1	0:02:22
MoSDi (14,0,0,7)	3	0.089	7	5	1	0.091	7	3	0	1:12:07
MoSDi (14,1,0,7)	0	0.081	6	5	1	0.088	9	2	0	0:10:57
MoSDi (14,1,0,7)	1	0.089	7	5	1	0.157	13	7	2	0:27:20
MoSDi (14,1,0,7)	2	0.123	10	8	2	0.187	15	7	3	1:12:47
MoSDi (14,1,0,7)	3	0.115	9	6	2	0.130	11	4	2	79:15:24
MEME	0	0.095	7	6	1	0.083	8	2	1	*0:00:12
MEME	1	0.116	8	7	0	0.117	9	5	3	*0:00:12
MEME	2	0.119	8	8	0	0.137	11	5	3	*0:00:12
MEME	3	0.108	7	7	1	0.109	9	4	2	*0:00:12
Weeder (medium)	0	0.018	2	1	0	0.033	4	0	0	0:02:19
Weeder (medium)	1	0.038	4	0	0	0.053	7	1	0	0:01:52
Weeder (medium)	2	0.040	5	1	0	0.074	8	1	0	0:01:56
Weeder (medium)	3	0.067	8	2	0	0.050	6	1	0	0:01:57
Weeder (large)	0	0.018	2	1	0	0.033	4	0	0	0:30:15
Weeder (large)	1	0.038	4	0	0	0.053	7	1	0	0:38:10
Weeder (large)	2	0.040	5	1	0	0.074	8	1	0	0:31:29
Weeder (large)	3	0.067	8	2	0	0.050	6	1	0	0:29:32
Weeder (extra)	0	0.017	2	1	0	0.035	4	0	0	0:32:05
Weeder (extra)	1	0.052	4	2	0	0.074	10	2	0	0:35:29
Weeder (extra)	2	0.051	6	3	0	0.097	10	4	0	0:33:57
Weeder (extra)	3	0.072	9	4	0	0.092	11	4	0	0:31:53

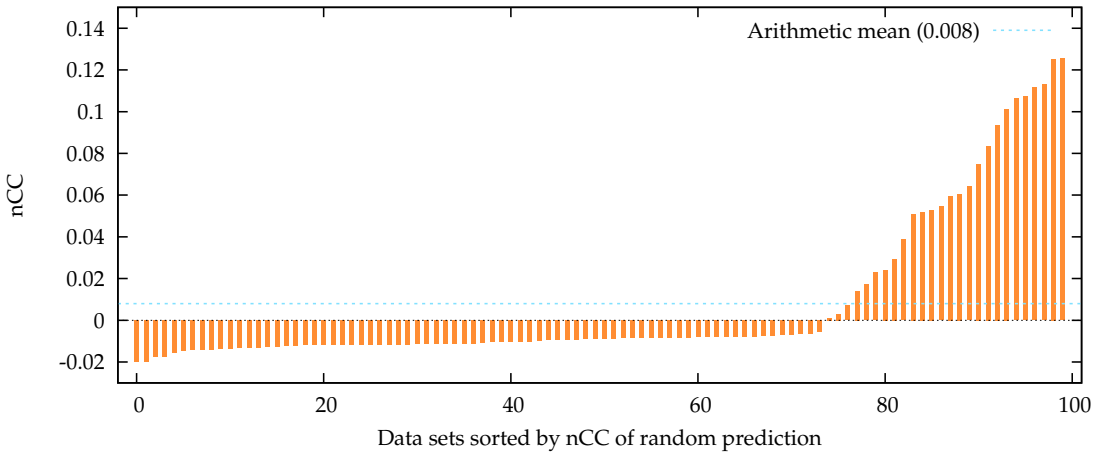


Figure 6.2: Scores of randomly predicted motifs are shown. For each data set, a motif length was sampled uniformly between 10 and 20. Then, a random position (also drawn uniformly) in each sequence was predicted to be a motif instance.

```
mosdi-discovery -v -t discovery -M 12,2,0,6 -O2 12 seq-count <file>
```

The option `-O2` says that a second order text model estimated from the input data is to be used. The global options `-v` and `-t` enable verbose output and measuring of runtimes, respectively. Refer to Section A.1.2 for more examples of using MoSDi. There, we also discuss the steps necessary to distribute the computation over several hosts.

The following motif spaces were used in this evaluation; they are given in terms of the motif length and of the maximum number of IUPAC characters from each group.

- Motif length: 10; wildcard constraints: 10,6,0,2; number of motifs: 17 880 633 344
- Motif length: 12; wildcard constraints: 12,2,0,6; number of motifs: 26 934 972 416
- Motif length: 14; wildcard constraints: 14,0,0,7; number of motifs: 6 088 884 224
- Motif length: 14; wildcard constraints: 14,1,0,7; number of motifs: 108 500 221 952

For the first two motif spaces, the optimal motifs with respect to i.i.d. (zeroth order), first order, and second order text models were extracted for all data sets. Experiments for third order text models were started but aborted as they would have taken significantly more CPU time than was available. For the last two motif spaces, experiments for all text models from zeroth to third order were completed successfully. Using the command-line given above, MoSDi estimates the text model from the input data by determining the frequencies of all $(q+1)$-grams, where q is the text model order. For third order models, we used the option `-s1` to add one pseudocount to each $(q+1)$-gram frequency to avoid zero values.

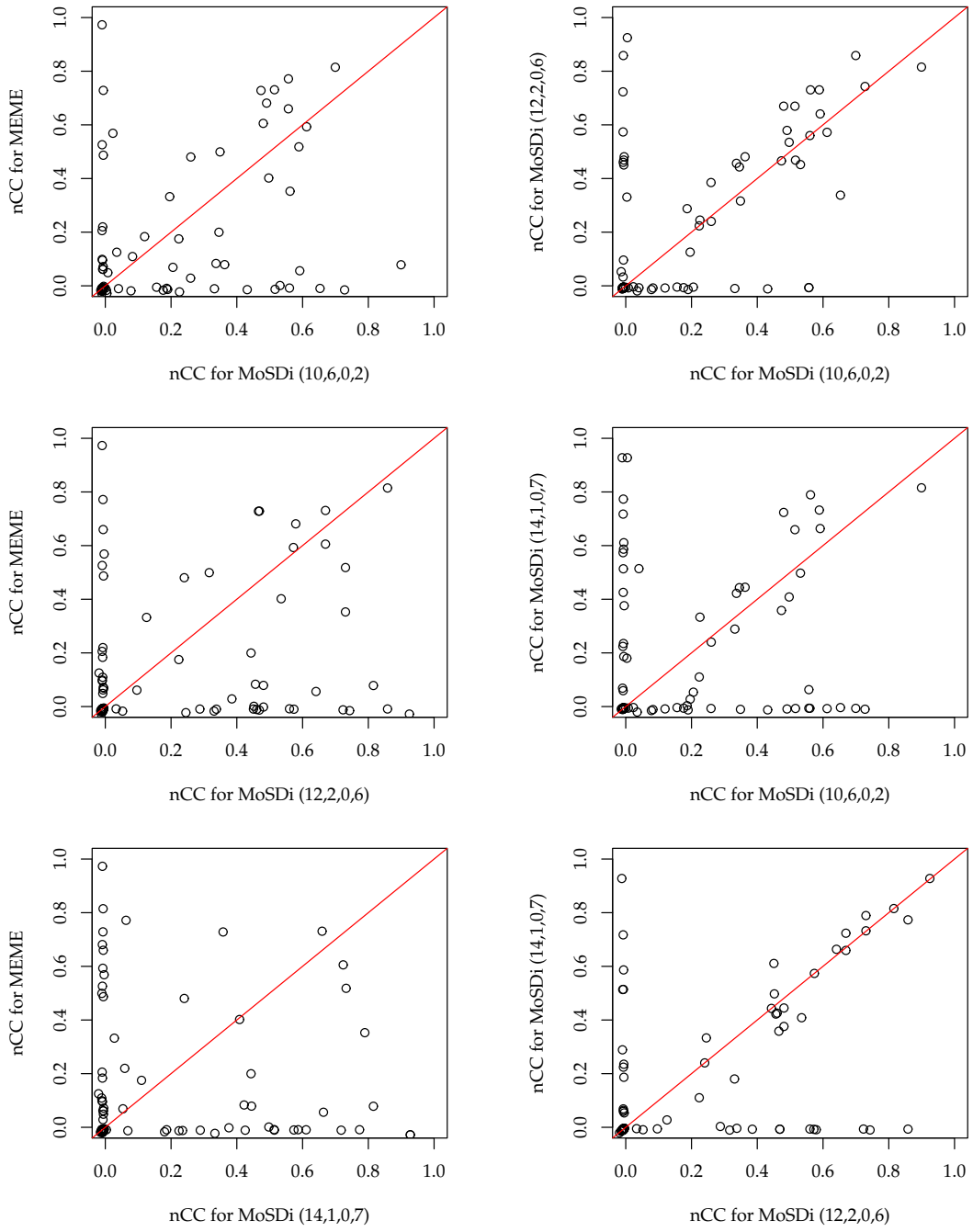


Figure 6.3: Comparison of nucleotide-level correlation coefficients (nCC) for different algorithms using a second order Markovian text model. Each circle represents one of the 100 data sets in the benchmark suites *Algorithm Markov* and *Algorithm real*. Continued in Figure 6.4.

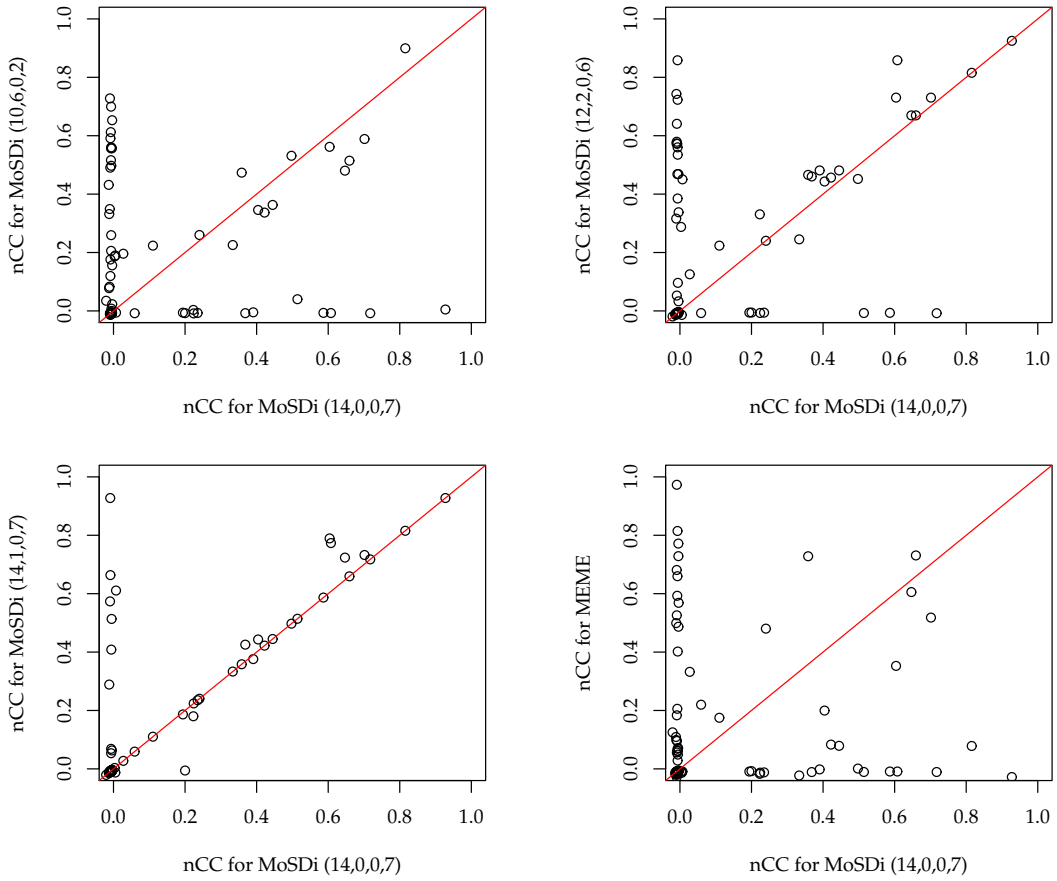


Figure 6.4: Continuation of Figure 6.3: Comparison of nucleotide-level correlation coefficients (nCC) for different algorithms using a second order Markovian text model. Each circle represents one of the 100 data sets in the benchmark suites *Algorithm Markov* and *Algorithm real*.

Results

To evaluate each tool's performance, we used the web service accompanying the paper by Sandve et al. (2007) to calculate the nucleotide-level correlation coefficient (nCC). The resulting scores for the three algorithms with different parameters and text models as described above are shown in Table 6.1. Figure 6.2 shows scores obtained when making random predictions for each of the 100 data sets. In the shown case, the average nCC amounts to 0.008. The best nCC score observed for a single data set is 0.125. In Table 6.1, the number of data sets for which each algorithm achieved an nCC above 0.25 are given. The comparison to the random predictions suggests that it is unlikely that such an nCC is obtained by chance.

The results in Table 6.1 show that increasing the text model order improves the prediction quality. For text model orders zero, one, and two, this is the case for all tools in all configurations with the only exception of Weeder run with the `extra` parameter

that obtained a score of 0.052 for a first-order model and of 0.051 for a second-order model. Further increasing the model order to three did not always have a positive effect. This could be due to the limited length of the input sequences which does not allow a reliable estimation of a model of this high order. Recall that we had to add a pseudocount to each observed $(q+1)$-gram frequency as some 4-grams do not occur in some of the input sequences. When the $(q+1)$-gram frequencies are this low, then the implanted motif itself can have a substantial impact on the model. Therefore, its significance decreases as well as the chance to detect it.

One unexpected result is that the average nCC was better for the data sets in *Algorithm real* compared to the data sets in *Algorithm Markov*. This is the case for 26 of all 30 algorithm variants listed in Table 6.1. On average, the performance was by 37.7 % better for *Algorithm real*. This is contrary to expectation because the real sequences might harbor other common patterns that putatively hamper the detection of the sought binding motif.

Independently of the order of the used text model, MEME needed 12 seconds to process a data set (on average) and was by far the fastest tool in this evaluation. This result is not surprising as MEME is the only tested tool that does not exhaustively scan the considered motif space. Therefore, it is remarkable that, for all text model orders, MEME obtained a higher score than Weeder for all configurations of Weeder.

The behavior of Weeder was somewhat surprising. When changing the parameter controlling the size of the motif space from `medium` to `large`, the top motif reported did not change for any of the data sets, but the runtime increased from around two minutes per data sets to around thirty minutes per data set. When it was changed from `large` to `extra`, Weeder made better predictions but the runtime did not increase significantly. In Table 2 of their article, Sandve et al. (2007) report that Weeder obtains an average nCC of 0.052 and 0.11 for *Algorithm Markov* and *Algorithm real*, respectively. This is in contrast to the numbers shown in the plots returned by their software (see Section A.4), where the values are 0.052 and 0.096. In either case, MEME performs better in less runtime when using a first or second order background model. As shown in Table 6.1, Weeder is also outperformed by MEME in the present evaluation.

The runtime of MoSDi strongly depends on the searched motif space and on the order of the text model. The best results were obtained for motifs of length 12 with wildcards constrained by (12,2,0,6). In this configuration, MoSDi took an average of 3:46 hours per data set and achieved an nCC of 0.162 and 0.198 for the benchmark suites *Algorithm Markov* and *Algorithm real*, respectively. MEME achieved nCC scores of 0.119 and 0.137, respectively, but took only 12 seconds per data set. Hence, MoSDi's results were better by 36.1 % and 44.5 %, but at the cost of using hours of CPU time instead of using seconds.

The smallest motif space MoSDi was tested for is the one given by the constraints (14,0,0,7), which means that it consists of those IUPAC motifs of length 14 that contain at most 7 `N`s and no other wildcard characters. For this motif space and a second order background model, MoSDi takes around 2.5 minutes per data set (on average) and performs better than Weeder using any of the tested configurations. A comparison of the runs for the motif spaces (14,0,0,7) and (14,1,0,7) shows that allowing one wildcard from the set $\{\texttt{R}, \texttt{Y}, \texttt{S}, \texttt{W}, \texttt{K}, \texttt{M}\}$ further increases prediction quality at the cost of longer

Table 6.5: Comparison of nCC obtained by MoSDi for different motif spaces and by MEME, both using a second order text model. Each row represents two data sets for the same TRANSFAC motif: once in random background, once in its original context. Yellow: nCC $\geq$ 0.25, orange: nCC $\geq$ 0.5, red: nCC $\geq$ 0.75.

	Algorithm Markov				Algorithm real			
Motif	10,6,0,2	12,2,0,6	14,1,0,7	MEME	10,6,0,2	12,2,0,6	14,1,0,7	MEME
M00441	0.08	-0.01	-0.01	0.11	0.73	0.74	-0.01	-0.02
M00444	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
M00621	-0.01	0.57	0.57	-0.01	-0.01	-0.01	-0.01	-0.01
M00622	0.03	-0.02	-0.02	0.13	0.35	0.44	0.44	0.20
M00639	0.56	0.56	-0.01	-0.01	-0.01	0.72	-0.01	-0.01
M00647	-0.01	-0.01	-0.01	0.21	-0.01	-0.01	-0.01	0.97
M00650	-0.01	-0.01	-0.01	-0.01	0.33	-0.01	0.29	-0.01
M00652	-0.01	0.47	-0.01	0.73	0.90	0.82	0.82	0.08
M00658	0.21	-0.00	0.05	0.07	-0.01	-0.00	-0.00	-0.01
M00693	-0.01	-0.01	-0.01	0.07	-0.01	-0.01	-0.01	-0.01
M00697	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	0.72	-0.01
M00699	0.43	-0.01	-0.01	-0.01	0.22	0.22	0.11	0.18
M00701	-0.01	-0.01	-0.01	0.10	-0.01	-0.01	-0.01	0.53
M00703	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.02
M00712	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	0.24	-0.01
M00733	0.08	-0.01	-0.01	-0.02	-0.01	0.05	-0.01	-0.02
M00743	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.02
M00764	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
M00765	-0.01	0.86	0.77	-0.01	0.70	0.86	-0.01	0.81
M00771	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	0.07	-0.01
M00774	0.56	-0.01	-0.01	0.66	0.56	-0.01	0.06	0.77
M00797	0.59	0.64	0.66	0.06	0.53	0.45	0.50	0.00
M00799	0.16	-0.00	-0.00	-0.01	0.65	0.34	-0.00	-0.01
M00800	-0.01	-0.01	0.06	0.22	-0.01	-0.01	-0.01	-0.01
M00809	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
M00810	0.49	0.58	-0.01	0.68	0.35	0.32	-0.01	0.50
M00918	-0.01	-0.01	-0.01	-0.01	0.00	-0.01	-0.01	-0.01
M00919	0.02	-0.00	-0.00	0.57	0.34	0.46	0.42	0.08
M00920	-0.00	0.48	0.38	-0.00	0.36	0.48	0.44	0.08
M00929	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	0.06
M00936	0.52	0.47	-0.01	-0.01	-0.01	-0.01	0.59	-0.01
M00938	-0.01	-0.01	-0.01	-0.02	-0.01	-0.01	-0.01	-0.01
M00939	-0.01	-0.01	0.19	-0.01	-0.01	0.46	0.43	-0.01
M00960	-0.01	-0.01	-0.00	-0.01	-0.01	-0.01	-0.01	0.49
M00965	-0.01	-0.01	0.51	-0.01	0.00	0.33	0.18	-0.02
M00966	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	0.22	-0.01
M00978	0.19	-0.01	-0.01	-0.01	0.20	0.13	0.03	0.33
M00979	0.59	0.73	0.73	0.52	0.56	0.73	0.79	0.35
M00982	-0.01	-0.01	-0.00	-0.01	-0.01	0.45	0.61	-0.01
M00983	0.61	0.57	-0.01	0.59	0.50	0.53	0.41	0.40
M00998	0.18	-0.01	-0.01	-0.02	-0.01	-0.01	-0.01	-0.01
M01007	0.47	0.47	0.36	0.73	0.26	0.24	0.24	0.48
M01011	0.26	0.38	-0.01	0.03	0.19	0.29	0.00	-0.01
M01013	0.04	-0.01	0.51	-0.01	0.23	0.25	0.33	-0.02
M01028	0.48	0.67	0.72	0.61	0.51	0.67	0.66	0.73
M01035	-0.01	-0.01	-0.01	-0.01	-0.01	0.03	-0.01	-0.01
M01036	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
M01037	0.12	-0.01	-0.01	0.18	-0.01	-0.01	-0.01	0.10
M01067	0.01	-0.01	-0.01	0.05	-0.01	0.10	-0.01	0.06
M01068	0.00	0.92	0.93	-0.03	-0.01	-0.01	0.93	-0.03

runtimes. Figures 6.3 and 6.4 show scatter plots comparing the performance of different pairs of algorithm variants. The bottom left scatter plot in Figure 6.4 shows that the improvement of the average nCC achieved by using (14,1,0,7) instead of (14,0,0,7) mainly stems from newly recognized motifs rather than from improvements of the scores for already recognized motifs. Except for one outlier, all motifs found using the smaller motif space are also found for the larger one. In addition, seven motifs are recognized with an nCC greater than 0.25 that were not spotted before.

For the remaining scatter plots in Figures 6.3 and 6.4, the picture is less clear. For all pairs of algorithm variants, there are data sets successfully solved by both variants but also data sets that could only be solved by one of them. The plots suggest that each algorithm variant might have its own strengths and weaknesses. To further illuminate this matter, Table 6.5 shows the nCC for all data sets for MEME and for MoSDi on the motif spaces constrained by (10,6,0,2), (12,2,0,6), and (14,1,0,7), where all algorithm variants use a second order text model. It indeed shows that each algorithm variant achieves an nCC equal to or larger than 0.25 for at least one data set for which the other three algorithm variants fail to do so. That means there is no single best algorithm variant and, in practice, one should consider the output of all of them.

To sum up, we have used the carefully crafted benchmark suite proposed by Sandve et al. (2007) to assess the algorithms developed in this thesis in comparison to Weeder and MEME. The use of Weeder does not seem to be advisable as it is neither as fast nor as accurate as MEME, which is by far the fastest tool tested. Consistently over all tools, a second order text model turned out to yield best results. For such text models, the predictions made by MoSDi were more accurate than those made by MEME. We are not aware of other motif discovery tools for which better results on this benchmark set have been reported. In practice, generating data sets of (for example) co-expressed genes which are commonly subjected to motif discovery can be costly and laborious. Therefore, it seems reasonable to employ an analysis that uses hours of CPU time and, in return, yields accurate predictions. Nonetheless, MEME remains an excellent tool for the rapid search for motifs. To conclude, both methods have their merits and should be included in the toolbox of a computational biologist.

6.3 Non-Coding Regions of M. tuberculosis

Mycobacterium tuberculosis is a species of pathogenic bacteria causing tuberculosis. Its genome has been completely deciphered by Cole et al. (1998). To demonstrate the utility of the proposed motif discovery algorithm in a whole genome setting, we search for motifs in the non-coding (i.e. putatively regulatory) regions of *M. tuberculosis*. The non-coding parts of the genome comprise 2 402 regions consisting of 398 419 base pairs, which is about one tenth of the whole genome.

We use MoSDi to search these regions for motifs of length 14 with wildcards constrained by (14,1,0,7). As demonstrated in Section 6.2, using this motif space yields good results on the benchmark suite of Sandve et al. (2007). We prefer it over the motif spaces with motifs of length 10 or 12 as our input is by more than a factor of 16 larger than the largest data from the benchmark suite. Therefore, motifs need to be more

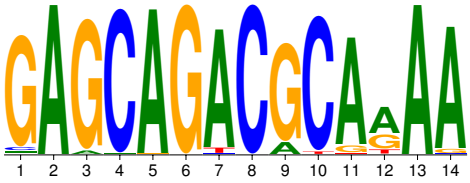
specific to be discernible from background noise. As we do not have prior knowledge on the placement of motifs to be discovered, we optimize the p-value for the total number of occurrences on both strands of DNA. That means that multiple motifs are allowed (and counted) in each sequence and its complement.

We estimated a second order text model from the input sequences and used it as the background model. As an additional constraint, we skipped all motifs with an expected number of occurrences above 20 as it seems unlikely that such unspecific motifs are of biological interest. Once the optimal motif had been discovered, all its instances were masked from the input sequences and the algorithm was rerun. The p-value was then optimized with respect to the masked sequences. This procedure was repeated until no more motifs with a p-value better than 10^{-25} existed. The resulting 24 motifs are shown below. They are given in the order they were discovered in. The p-value is given with respect to the original sequences (without masked motifs). Some motifs have more instances in the original sequences than in the masked sequences they were discovered in and, hence, a better p-value with respect to the original sequences than with respect to the masked sequences. Therefore, the p-values of the shown motifs do not increase monotonically. If a motif breaks monotonicity, that is, it has a better p-value than the motif before it, then its p-value on the original sequences must be better than on the masked sequences. Thus, the motif is most probably related to another motif discovered earlier.

In the lines prefixed with "Exact", we give the total number of occurrences, the expected number of occurrences (E) and the p-value (p). In the lines prefixed with "Dist. 1" and "Dist. 2", the same statistics are given for motif occurrences within a Hamming distance of one or two of the IUPAC motif, respectively. The sequence logos are generated from the set of occurrences within a Hamming distance of one.

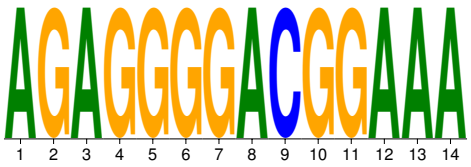
Motif 1: `GAGCAGACRCANAA`

Exact:	81,	E: 0.014,	p: $2.0 \cdot 10^{-271}$
Dist. 1:	121,	E: 0.61,	p: $7.6 \cdot 10^{-228}$
Dist. 2:	155,	E: 11.9,	p: $6.8 \cdot 10^{-113}$



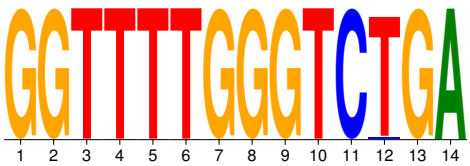
Motif 2: `AGAGGGGACGGAAA`

Exact:	42,	E: 0.00093,	p: $3.0 \cdot 10^{-179}$
Dist. 1:	42,	E: 0.053,	p: $1.5 \cdot 10^{-105}$
Dist. 2:	44,	E: 1.3,	p: $3.6 \cdot 10^{-50}$



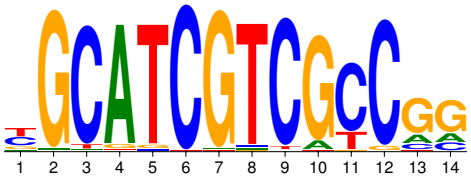
Motif 3: `GGTTTTGGGTCTGA`

Exact:	41,	E: 0.0031,	p: $2.6 \cdot 10^{-153}$
Dist. 1:	42,	E: 0.13,	p: $1.1 \cdot 10^{-89}$
Dist. 2:	44,	E: 2.4,	p: $4.3 \cdot 10^{-39}$



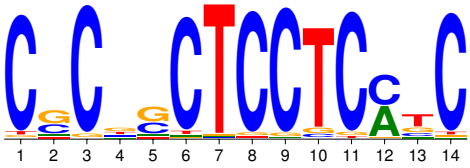
Motif 4: `NGCATCGTCGYCNN`

Exact:	88,	E: 1.2,	p: $2.6 \cdot 10^{-128}$
Dist. 1:	171,	E: 32.4,	p: $7.5 \cdot 10^{-65}$
Dist. 2:	635,	E: 398.8,	p: $4.6 \cdot 10^{-26}$



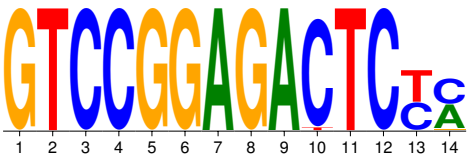
Motif 5: CNCNNCTCCTCMNC

Exact:	87,	E: 1.5,	p: $2.5 \cdot 10^{-113}$
Dist. 1:	203,	E: 52.3,	p: $2.9 \cdot 10^{-52}$
Dist. 2:	1092,	E: 758.4,	p: $1.8 \cdot 10^{-24}$



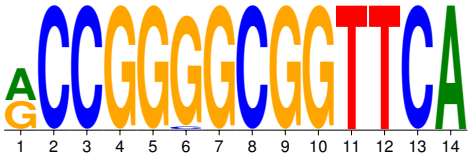
Motif 6: GTCCGGAGACTCYN

Exact:	29,	E: 0.016,	p: $8.2 \cdot 10^{-84}$
Dist. 1:	30,	E: 0.76,	p: $1.9 \cdot 10^{-36}$
Dist. 2:	43,	E: 15.8,	p: $1.7 \cdot 10^{-08}$



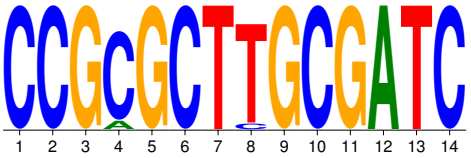
Motif 7: RCCGGGGCGGTTCA

Exact:	31,	E: 0.034,	p: $5.5 \cdot 10^{-80}$
Dist. 1:	32,	E: 1.2,	p: $1.5 \cdot 10^{-34}$
Dist. 2:	64,	E: 18.2,	p: $6.2 \cdot 10^{-17}$



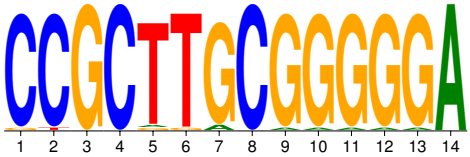
Motif 8: CCGMGCTTGCGATC

Exact:	30,	E: 0.031,	p: $2.4 \cdot 10^{-78}$
Dist. 1:	32,	E: 1.0,	p: $2.5 \cdot 10^{-36}$
Dist. 2:	51,	E: 15.6,	p: $1.3 \cdot 10^{-12}$



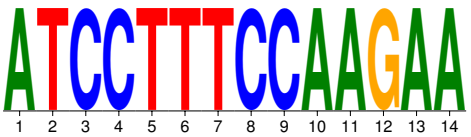
Motif 9: CCGCTTRCGGGGGA

Exact:	25,	E: 0.0096,	p: $2.2 \cdot 10^{-76}$
Dist. 1:	35,	E: 0.44,	p: $2.2 \cdot 10^{-53}$
Dist. 2:	47,	E: 8.9,	p: $3.1 \cdot 10^{-19}$



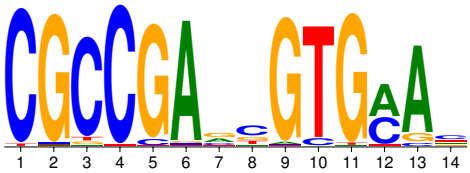
Motif 10: ATCCTTTCCAAGAA

Exact:	16,	E: 0.00051,	p: $10.0 \cdot 10^{-67}$
Dist. 1:	16,	E: 0.026,	p: $2.6 \cdot 10^{-39}$
Dist. 2:	16,	E: 0.64,	p: $1.9 \cdot 10^{-17}$



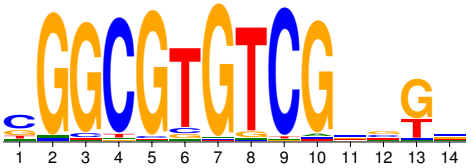
Motif 11: CGCCGANNGTGMAN

Exact:	48,	E: 1.3,	p: $2.4 \cdot 10^{-57}$
Dist. 1:	120,	E: 33.2,	p: $3.4 \cdot 10^{-31}$
Dist. 2:	580,	E: 400.6,	p: $8.5 \cdot 10^{-17}$



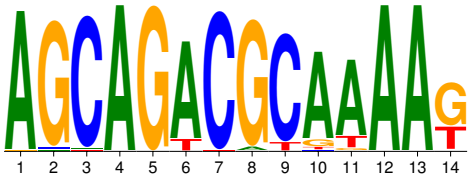
Motif 12: NGGCGTGTCGNNKN

Exact:	70,	E: 5.0,	p: $6.9 \cdot 10^{-54}$
Dist. 1:	215,	E: 120.4,	p: $7.8 \cdot 10^{-15}$
Dist. 2:	1508,	E: 1314.5,	p: $4.6 \cdot 10^{-07}$



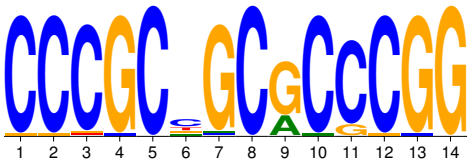
Motif 13: AGCAGWCGNNAAAG

Exact:	32,	E: 0.062,	p: $1.0 \cdot 10^{-74}$
Dist. 1:	89,	E: 2.4,	p: $2.0 \cdot 10^{-104}$
Dist. 2:	162,	E: 40.9,	p: $3.2 \cdot 10^{-46}$



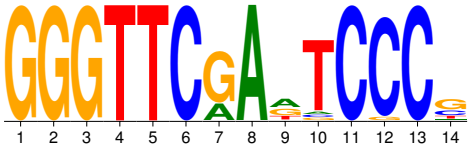
Motif 14: CCCGCNGCRCCCGG

Exact:	25,	E: 0.33,	p: $4.7 \cdot 10^{-38}$
Dist. 1:	40,	E: 9.5,	p: $1.6 \cdot 10^{-13}$
Dist. 2:	156,	E: 125.0,	p: $4.7 \cdot 10^{-03}$



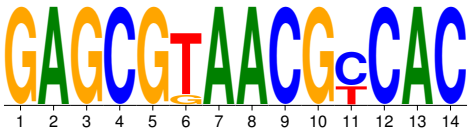
Motif 15: GGGTTCRANTCCCN

Exact:	20,	E: 0.11,	p: $3.9 \cdot 10^{-38}$
Dist. 1:	24,	E: 4.1,	p: $1.8 \cdot 10^{-11}$
Dist. 2:	96,	E: 66.8,	p: $5.4 \cdot 10^{-04}$



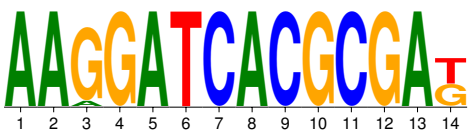
Motif 16: GAGCGTAACGYCAC

Exact:	12,	E: 0.0051,	p: $6.6 \cdot 10^{-37}$
Dist. 1:	14,	E: 0.25,	p: $3.1 \cdot 10^{-20}$
Dist. 2:	27,	E: 5.2,	p: $1.7 \cdot 10^{-11}$



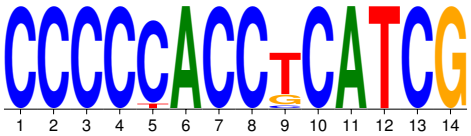
Motif 17: AAGGATCACGCGAK

Exact:	12,	E: 0.0062,	p: $6.1 \cdot 10^{-36}$
Dist. 1:	13,	E: 0.25,	p: $1.8 \cdot 10^{-18}$
Dist. 2:	18,	E: 4.7,	p: $2.4 \cdot 10^{-06}$



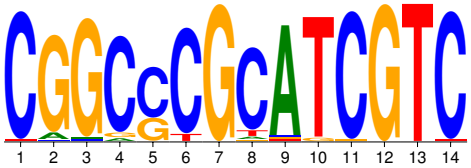
Motif 18: CCCCYACCTCATCG

Exact:	12,	E: 0.0076,	p: $7.2 \cdot 10^{-35}$
Dist. 1:	16,	E: 0.33,	p: $5.9 \cdot 10^{-22}$
Dist. 2:	36,	E: 6.4,	p: $5.7 \cdot 10^{-16}$



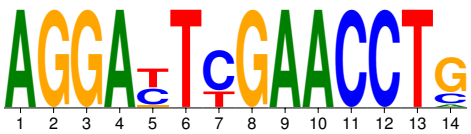
Motif 19: CGGCSCGNATCGTC

Exact:	33,	E: 0.17,	p: $2.8 \cdot 10^{-63}$
Dist. 1:	57,	E: 5.0,	p: $2.2 \cdot 10^{-39}$
Dist. 2:	163,	E: 70.1,	p: $3.5 \cdot 10^{-21}$



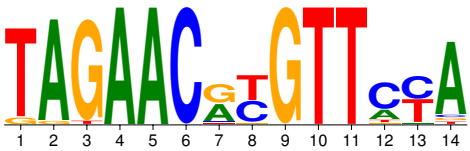
Motif 20: AGGANTYGAACCTN

Exact:	14,	E: 0.047,	p: $2.6 \cdot 10^{-30}$
Dist. 1:	14,	E: 1.8,	p: $1.1 \cdot 10^{-08}$
Dist. 2:	65,	E: 32.9,	p: $5.1 \cdot 10^{-07}$



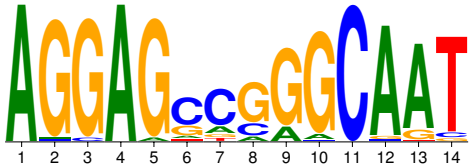
Motif 21: `TAGAACNYGTTNNA`

Exact:	17,	E: 0.078,	p: $1.7 \cdot 10^{-29}$
Dist. 1:	25,	E: 3.3,	p: $1.3 \cdot 10^{-12}$
Dist. 2:	100,	E: 61.1,	p: $1.6 \cdot 10^{-05}$



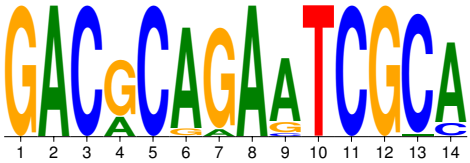
Motif 22: `AGGAGNNNRGCAAT`

Exact:	30,	E: 0.17,	p: $2.5 \cdot 10^{-56}$
Dist. 1:	55,	E: 6.6,	p: $1.5 \cdot 10^{-31}$
Dist. 2:	175,	E: 112.3,	p: $3.0 \cdot 10^{-08}$



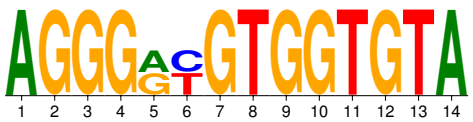
Motif 23: `GACRCAGANTCGCA`

Exact:	29,	E: 0.030,	p: $9.9 \cdot 10^{-76}$
Dist. 1:	42,	E: 1.2,	p: $2.1 \cdot 10^{-49}$
Dist. 2:	88,	E: 20.6,	p: $5.0 \cdot 10^{-28}$



Motif 24: `AGGGRNGTGGTGTA`

Exact:	10,	E: 0.0094,	p: $1.5 \cdot 10^{-27}$
Dist. 1:	10,	E: 0.44,	p: $5.6 \cdot 10^{-11}$
Dist. 2:	16,	E: 9.4,	p: $3.0 \cdot 10^{-02}$



All 24 discovered motifs have low p-values and are hence highly significant in the statistical sense. Even when taking into account that the motif space and thus the number of tested hypotheses is large ($108\,500\,221\,952 \approx 10^{11}$), for example by applying a Bonferroni correction (see Wasserman, 2003), all results remain significant. Judging whether these findings are biologically significant is more difficult. Here, we restrict ourselves to reporting some peculiarities of the above motifs.

One question we ask is whether considering a Hamming neighborhood improves motifs. We can argue that, if a motif is a false positive, then the chance of also finding an exceptional number of occurrences in a Hamming neighborhood is small. However, when the number of occurrences does *not* increase by adding a Hamming neighborhood, we cannot draw the converse conclusion as it could also mean that a true motif is strongly conserved across all its instance. Out of all motifs shown above, Motif 13 is the only one for which the p-value with respect to all matches within a Hamming distance of one is better than the p-value with respect to all exact matches. And in fact, a biological function can be attributed to this motif as we discuss below. Although the p-value of no other motif improves by adding a Hamming neighborhood, the number of additional occurrences is remarkable for some. For the Motifs 1, 3, 9, 13, 16, 17, 18, 19, 22, and 23, the increase in the number of occurrences is more than three times as high as the increase in the expected number of occurrences. Interestingly, four of these motifs are among those five motifs we could find a biological explanation for.

In an earlier analysis of the non-coding regions of *M. tuberculosis* (Marschall and Rahmann, 2009), the best motif found was `AGACSCARAA`. As pointed out by Christoph Kaleta (personal communication), the biological meaning of this motif is twofold. On

the one hand, its instances are part of *clustered regularly-interspaced short palindromic repeats* (CRISPRs) which have been shown to be involved in bacteriophage response by Barrangou et al. (2007). On the other hand, the motif plays a role in transcription termination. Due to the longer motifs (length 14 instead of length 10) used in the present study, MoSDi succeeds in separating these two variants of instances of AGACSCARAA.

The entry on *M. tuberculosis* in the database of putative CRISPRs developed by Grissa et al. (2007) contains the sequence

GTTTCCGTCCCCTCTCGGGGTTTTGGGTCTGACGAC.

The underlined parts correspond precisely to the reverse complement of Motif 2 and to Motif 3, respectively. A number of instances of AGACSCARAA can be explained by this CRISPR as TTYTGSGTCT, the reverse complement of AGACSCARAA, matches Positions 3 to 12 of Motif 3.

The second type of sequences matched by AGACSCARAA are intrinsic transcription terminators. These are palindromic sequences that form stem loops in case of transcription, forcing the DNA polymerase to stop to further transcribe the DNA (see Alberts et al., 2007, Page 338). According to Kaleta (personal communication), the sequences AGACGCAAAA and AGACGCAGAA, which both match AGACSCARAA, are part of such transcription terminators. In the present study, these sequences are found in Motifs 1, 13, and 23: in Motif 1, the consensus of Positions 5 to 14 is AGACGCAAAA; in Motif 13, the consensus of Positions 4 to 13 is AGACGCAAAA; and in Motif 23, the consensus of Positions 1 to 9 is GACGCAGAA.

To sum up, we have discovered 24 highly significant motifs for 5 of which we are aware of a biological meaning. Searching for motifs of length 14 instead of for motifs of length 10 as in an earlier study (Marschall and Rahmann, 2009) leads to a more detailed view of motifs in *M. tuberculosis*. On the one hand, a larger number of different motifs are found, and, on the other hand, the two different functions of instances of AGACSCARAA are now reflected in distinct motifs.

Presently, the function of the remaining motifs is unknown to us. Because of their high statistical significance, they are excellent candidates for further investigations about their biological meaning.

6.4 CpG Islands

The dinucleotide CG is found rarely in mammalian genomes. That means cytosine is seldom followed by guanine on the same strand of DNA. This dinucleotide is often referred to as CpG, where the "p" represents the phosphodiester bond that holds together two adjacent nucleotides in a DNA strand. Only as few as one percent of all dinucleotides in the human genome are CpGs. Furthermore, the CpGs are not distributed equidistantly over the genome, but are concentrated in regions whose length typically does not exceed 2 000 nucleotides. These regions are called *CpG islands*. Commonly, the criteria of Gardiner-Garden and Frommer (1987) and Takai and Jones (2002) are used to formally define CpG islands in terms of their minimal length, their CG content, and the ratio of observed and expected number of CpGs. However, finding

Table 6.6: Statistics on data sets taken from Supplementary Data Set S1 accompanying the paper by Illingworth et al. (2008). The four data sets consist of regions they classify as *Category 1*; these are sets of regions that are exclusively methylated in one of the examined tissues. The column *CpGs* gives the percentage of dinucleotides that are CpGs. The rightmost column lists the number of regions that are labeled as 3p of genes, 5p of genes, intra-, or intergenic by Illingworth et al. (2008). Some regions are multiply annotated such that numbers sum to values larger than the number of regions.

Data set	Sequences	Avg. length	Total length	CpGs	3p / 5p / intra / inter
Blood	42	1 572.0	66 023	6.8 %	6 / 15 / 11 / 15
Brain	30	1 325.6	39 767	6.6 %	3 / 6 / 10 / 12
Muscle	157	1 224.5	192 241	7.5 %	19 / 54 / 45 / 48
Spleen	122	1 519.3	185 359	7.6 %	9 / 62 / 30 / 30

an appropriate definition of CpG islands that does not contain *ad hoc* thresholds is a subject of ongoing research.

As detailed in the textbook by Alberts et al. (2007, Pages 470ff.), the low number of CpGs in the human genome can be explained by events of *spontaneous deamination* of nucleotides that take place in the course of evolution. Such a deamination reaction removes an amine group from a nucleotide. Under normal conditions, deaminated nucleotides are repaired by special molecules that restore them into their previous state. But when a cytosine is methylated, that is, an additional methyl group is attached to it, deamination results in a thymine which cannot be detected and subsequently repaired by the cell's machinery. In differentiated cells (as opposed to stem cells), methylated cytosines are found almost exclusively in CpG dinucleotides (see Lister et al., 2009). Therefore, the C in a CpG is only protected from being changed into a T over evolutionary time spans when it is either unmethylated in germ cells or evolution selects against its mutation. Indeed, CpG islands are most often unmethylated in most tissues.

Most CpG islands are found in or near genes, many of them in promoter regions. Therefore, the rare cases where CpG islands are methylated are important as methylation of promoter regions silences the corresponding gene (see Suzuki and Bird, 2008) and is involved in the development of cancer (see Jones and Baylin, 2007). Examining Chromosomes 6, 20 and 22 by means of bisulfite sequencing, Eckhardt et al. (2006) show that the methylation patterns of CpG islands are tissue specific.

Here, we ask whether we can find *overrepresented motifs* in CpG islands that are methylated in a tissue-specific way. To this end, we use a data set of differentially methylated regions published by Illingworth et al. (2008). They use a method called

CXXC affinity purification (CAP) to extract unmethylated CpG-rich regions from the DNA in human blood. By sequencing the resulting sample of CXXC-affine DNA, they identified a library of 17 387 CpG islands. Although this protocol excludes fully-methylated CpG islands, the authors reason that CpG islands that are only methylated in a fraction of all copies might be included. In a subsequent step, they construct a microarray from the library. Regions of DNA methylated in blood, brain, muscle, and spleen, respectively, are selected using *MDB affinity purification* (MAP) and subsequently hybridized to the microarray. The resulting expression profile then allows concluding which CpG islands were metyhlated in which tissues. Table 6.6 describes the sets of CpG islands identified as being methylated in only one of the four tissues.

We scan each of the four data sets for overrepresented motifs using MoSDi. As in the previous section, we search the space of all motifs of length 14 with wildcards constrained by (14,1,0,7) and iteratively rerun motif discovery once an optimal motif has been discovered and excised from the input sequences. Again, we stop when no further motifs with a p-value better than 10^{-25} are found. Here, the p-value for the number of sequences a motif occurs in is used.

For the data sets blood and spleen, 14 motifs and 15 motifs were discovered, respectively. For the data sets muscle and brain, only 3 motifs and 0 motifs were found, respectively. All these motifs are listed in Tables 6.7, 6.8, and 6.9. The difference in the number of discovered motifs is remarkable. The *muscle* data set is the largest one, while the *brain* data set is the smallest one. Therefore, the number of found motifs does not seem to correlate with the size of the data sets. The rightmost column in each table shows statistics on the placement of motif-containing CpG islands with respect to known genes. These distributions do not deviate substantially from the respective distributions for the whole data sets as shown in Table 6.6. Therefore, it seems unlikely that the found motifs are directly involved in the regulation of transcription.

For each motif, the tables show the number of sequences it occurs in and the corresponding p-value. The motifs are listed in the order they were discovered in. As Table 6.7 shows, all motifs discovered in the blood data set also occur in the muscle and spleen data sets, but none occurs in the brain data set. The three motifs shown in Table 6.8 that were discovered in the muscle data set occur in all other data sets, including brain. These motifs seem to have a low complexity; the first targets sequences mostly composed of `A`s, the second sequences mostly composed of `C`s, and the third sequences consisting of repeated instances of `GCC`. For all these three motifs, similar ones have been discovered in the spleen data set as can be seen in Table 6.9. Across all three tables, similar motifs are given the same color.

To detect similar motifs, we analyzed for each pair of motifs whether their instances overlap. The most remarkable overlaps were detected for the groups of motifs colored red, gray, and light blue. A further analysis of the placement of motifs revealed that the motifs in these groups are part of longer motifs. To detect these longer motifs, a local search strategy was employed. In each iteration, all possible substitutions of IUPAC characters and additions of IUPAC characters at either end of the motif are tested and the modified motif with the best p-value is retained. The motifs resulting from this procedure and their relation to the shorter motifs initially discovered are

Table 6.7: Motifs discovered in CpG islands methylated only in *blood* are listed. For each tissue, the number of sequences with at least one occurrence and the corresponding p-value with respect to a second order background model is given. The rightmost column lists the number of matching regions that are labeled as 3p of genes, 5p of genes, intra-, or intergenic by Illingworth et al. (2008). Some regions are multiply annotated such that numbers may sum to values larger than the sum of regions with at least one occurrence. Groups of motifs printed with the same background color are related: some of their instances overlap. The same color codes are used in Tables 6.8 and 6.9.

Motif	**Blood**	Brain	Muscle	Spleen	3p / 5p / intra / inter
TACTAAAAMTACAA	**11 ($1 \cdot 10^{-56}$)**	0 (1.0)	2 ($4 \cdot 10^{-09}$)	7 ($4 \cdot 10^{-33}$)	5 / 10 / 5 / 5
GTGCTRGGATTACA	**13 ($9 \cdot 10^{-55}$)**	0 (1.0)	2 ($4 \cdot 10^{-07}$)	9 ($2 \cdot 10^{-33}$)	4 / 11 / 6 / 7
AGWTCGAGACNANC	**13 ($7 \cdot 10^{-44}$)**	0 (1.0)	2 ($5 \cdot 10^{-05}$)	9 ($2 \cdot 10^{-25}$)	3 / 10 / 7 / 8
AAATTAGCNGGGCR	**13 ($3 \cdot 10^{-41}$)**	0 (1.0)	1 ($1 \cdot 10^{-02}$)	11 ($7 \cdot 10^{-30}$)	5 / 14 / 8 / 5
ACTGCACTCCASCC	**12 ($1 \cdot 10^{-39}$)**	0 (1.0)	6 ($4 \cdot 10^{-16}$)	7 ($1 \cdot 10^{-18}$)	4 / 13 / 4 / 7
AGTANCTGNGAYTA	**11 ($7 \cdot 10^{-38}$)**	0 (1.0)	3 ($2 \cdot 10^{-08}$)	9 ($2 \cdot 10^{-27}$)	3 / 12 / 7 / 6
TCRGCTCACTGCAA	**10 ($6 \cdot 10^{-38}$)**	0 (1.0)	3 ($2 \cdot 10^{-09}$)	8 ($3 \cdot 10^{-26}$)	3 / 14 / 6 / 3
ATTCTCCTGYCTCA	**10 ($4 \cdot 10^{-35}$)**	0 (1.0)	4 ($3 \cdot 10^{-11}$)	6 ($1 \cdot 10^{-17}$)	4 / 10 / 7 / 4
RCCAACATGGNGAA	**10 ($2 \cdot 10^{-35}$)**	0 (1.0)	1 ($5 \cdot 10^{-03}$)	8 ($3 \cdot 10^{-23}$)	2 / 10 / 7 / 5
CTNRGCCTCCCAAA	**13 ($9 \cdot 10^{-34}$)**	0 (1.0)	4 ($2 \cdot 10^{-07}$)	12 ($9 \cdot 10^{-26}$)	4 / 15 / 8 / 6
AANAAAANAWANAA	**14 ($2 \cdot 10^{-30}$)**	0 (1.0)	10 ($2 \cdot 10^{-17}$)	17 ($2 \cdot 10^{-34}$)	6 / 23 / 11 / 8
TCAAGNGATYCNCC	**12 ($7 \cdot 10^{-32}$)**	0 (1.0)	5 ($1 \cdot 10^{-09}$)	11 ($5 \cdot 10^{-24}$)	6 / 12 / 9 / 6
GGCGTGAGNCACYG	**10 ($1 \cdot 10^{-26}$)**	0 (1.0)	1 ($3 \cdot 10^{-02}$)	4 ($5 \cdot 10^{-08}$)	3 / 7 / 4 / 4
ACAGARCNANACTC	**10 ($4 \cdot 10^{-26}$)**	0 (1.0)	1 ($4 \cdot 10^{-02}$)	3 ($9 \cdot 10^{-06}$)	2 / 7 / 5 / 4

Table 6.8: Motifs discovered in CpG islands methylated only in *muscle* are listed. See caption of Table 6.7 for an explanation of shown values.

Motif	Blood	Brain	**Muscle**	Spleen	3p / 5p / intra / inter
AAANNANWANANNA	17 ($5 \cdot 10^{-16}$)	3 ($3 \cdot 10^{-02}$)	**44 ($2 \cdot 10^{-39}$)**	27 ($5 \cdot 10^{-21}$)	10 / 36 / 24 / 28
CCNCNWCCCNNNCC	19 ($1 \cdot 10^{-06}$)	10 ($1 \cdot 10^{-03}$)	**77 ($4 \cdot 10^{-29}$)**	42 ($4 \cdot 10^{-08}$)	22 / 71 / 40 / 27
GCCGCCRCCGCCNC	2 ($2 \cdot 10^{-03}$)	3 ($6 \cdot 10^{-06}$)	**17 ($2 \cdot 10^{-27}$)**	17 ($2 \cdot 10^{-26}$)	5 / 24 / 8 / 4

Table 6.9: Motifs discovered in CpG islands methylated only in *spleen* are listed. See caption of Table 6.7 for an explanation of shown values.

Motif	Blood	Brain	Muscle	**Spleen**	3p / 5p / intra / inter
AAGTNYTGGNATTA	12 ($6 \cdot 10^{-41}$)	0 (1.0)	3 ($7 \cdot 10^{-08}$)	**17 ($5 \cdot 10^{-53}$)**	6 / 16 / 9 / 7
AAAANANAAAAAAW	12 ($2 \cdot 10^{-26}$)	0 (1.0)	9 ($4 \cdot 10^{-17}$)	**21 ($9 \cdot 10^{-48}$)**	6 / 23 / 9 / 9
AAWTACAAAAATTA	9 ($1 \cdot 10^{-41}$)	0 (1.0)	3 ($2 \cdot 10^{-12}$)	**10 ($2 \cdot 10^{-44}$)**	6 / 10 / 8 / 4
CAKCCTGGNCAACA	8 ($5 \cdot 10^{-23}$)	0 (1.0)	4 ($3 \cdot 10^{-09}$)	**17 ($8 \cdot 10^{-45}$)**	4 / 14 / 9 / 7
TAGNTGGGAYTACA	7 ($1 \cdot 10^{-26}$)	0 (1.0)	4 ($2 \cdot 10^{-13}$)	**11 ($2 \cdot 10^{-39}$)**	3 / 13 / 7 / 4
CAGGAGWTCNAGAC	10 ($7 \cdot 10^{-31}$)	0 (1.0)	3 ($5 \cdot 10^{-07}$)	**13 ($1 \cdot 10^{-35}$)**	3 / 10 / 9 / 8
CCTRCCTCAGCCTC	8 ($3 \cdot 10^{-22}$)	0 (1.0)	4 ($9 \cdot 10^{-09}$)	**14 ($2 \cdot 10^{-35}$)**	4 / 12 / 8 / 6
AGNYTGCAGTGANC	12 ($2 \cdot 10^{-30}$)	0 (1.0)	5 ($3 \cdot 10^{-09}$)	**16 ($8 \cdot 10^{-35}$)**	4 / 18 / 11 / 6
CCACTGYACTCNAG	10 ($2 \cdot 10^{-29}$)	0 (1.0)	6 ($2 \cdot 10^{-14}$)	**13 ($8 \cdot 10^{-34}$)**	5 / 18 / 5 / 5
AACCCCRTNTCTAC	8 ($2 \cdot 10^{-29}$)	0 (1.0)	1 ($3 \cdot 10^{-03}$)	**10 ($1 \cdot 10^{-33}$)**	3 / 12 / 6 / 3
AANANNAWNNANNA	20 ($2 \cdot 10^{-12}$)	4 ($1 \cdot 10^{-01}$)	44 ($1 \cdot 10^{-20}$)	**54 ($3 \cdot 10^{-35}$)**	11 / 50 / 36 / 36
GCCGCCRCCGCCGC	1 ($1 \cdot 10^{-02}$)	3 ($8 \cdot 10^{-08}$)	11 ($2 \cdot 10^{-22}$)	**15 ($5 \cdot 10^{-31}$)**	5 / 18 / 7 / 2
CCCSNNCCCCNNCC	12 ($5 \cdot 10^{-04}$)	10 ($2 \cdot 10^{-05}$)	48 ($5 \cdot 10^{-15}$)	**61 ($7 \cdot 10^{-30}$)**	16 / 66 / 36 / 25
GGTTCAAGNRATTC	8 ($7 \cdot 10^{-29}$)	0 (1.0)	6 ($2 \cdot 10^{-17}$)	**9 ($2 \cdot 10^{-28}$)**	4 / 13 / 6 / 5
CAGGCRTNAGCCAC	9 ($6 \cdot 10^{-25}$)	0 (1.0)	2 ($2 \cdot 10^{-04}$)	**11 ($1 \cdot 10^{-26}$)**	3 / 12 / 6 / 4

shown in Figures 6.10, 6.11, and 6.12. All three combined motifs occur in the data sets blood, muscle, and spleen, but none in brain.

Next, we counted the number of occurrences of these three motifs in the whole human genome, noticing that they are *extremely abundant*. As Table 6.13 shows, most of the occurrences lie in annotated repeats. Some lie in regions upstream of genes.

In summary, we have discovered three strikingly strong motifs in tissue-specifically methylated CpG islands that are also extraordinarily common in the whole human genome. The presence of these motifs cannot be attributed to chance. The fact that most of their instances are annotated as repeats does not necessarily imply that they are meaningless. In any case, all regions covered by these motifs should be annotated as repeats. Elucidating the role of these motifs is an interesting task for future research.

Figure 6.10: A sequence logo for AMMCCYYRTYTCTACYARAAATWCAAAAATTAGCHKKGYR is shown. All occurrences in any of the four data sets with a Hamming distance of at most two to this IUPAC string are counted, totaling 22 occurrences. This long motif subsumes the motifs AACCCCRTNTCTAC (Positions 2 to 15), TACTAAAAMTACAA (Positions 13 to 26), AAWTACAAAAATTA (Positions 19 to 32), and AAATTAGCNGGGCR (Positions 27 to 40).

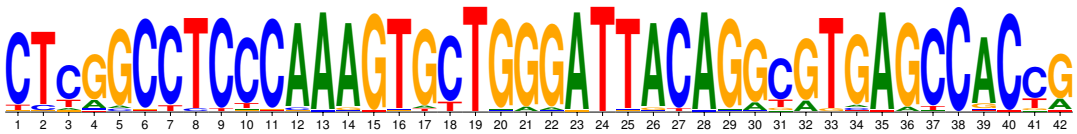


Figure 6.11: A sequence logo for YBNNRCYYYNCAAVGTDYTGGNATTACANRNRTNARHCNYND is shown. All occurrences in any of the four data sets with a Hamming distance of at most two to this IUPAC string are counted, totaling 53 occurrences. This long motif subsumes the motifs CTNRGCCTCCCAAA (Positions 1 to 14), AAGTNYTGGNATTA (Positions 13 to 26), GTGCTRGGATTACA (Positions 15 to 28), CAGGCRTNAGCCAC (Positions 27 to 34), and GGCGTGAGNCACYG (Positions 29 to 42).

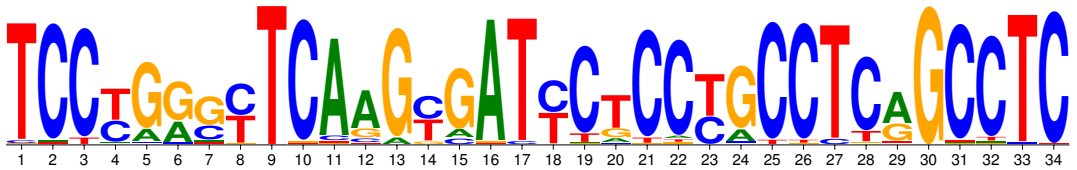


Figure 6.12: A sequence logo for TMYNRRBYTSAVRNVATYHNBBYRYCTHNGCMTC is shown. All occurrences in any of the four data sets with a Hamming distance of at most two to this IUPAC string are counted, totaling 95 occurrences. This long motif subsumes the motifs GGTTCAAGNRATTC (Positions 6 to 19), TCAAGNGATYCNCC (Positions 9 to 22), ATTCTCCTGYCTCA (Positions 16 to 29), and CCTRCCTCAGCCTC (Positions 21 to 34).

Table 6.13: The number of occurrences of the three motifs shown in Figures 6.10, 6.11, and 6.12 in different parts of the human genome are listed. The given numbers are the absolute occurrence counts of the IUPAC strings given in the caption of the respective figure. Numbers in parentheses give the average number of occurrences per one million base pairs. The column *Upstream of genes* refers to regions of 1 000 base pairs upstream of genes. The column *Repeats* refers to the regions defined by the "RepeatMasker" track as downloaded from the UCSC genome browser (`http://genome.ucsc.edu`). It contains repeats identified by the *RepeatMasker* program using the *RepBase* database (Jurka et al., 2005).

Motif	Upstream of genes	Repeats	Whole Genome
`AMMCCYYRTY...TAGCHKKGYR`	460 (15.1)	45 078 (32.0)	45 260 (14.6)
`YBNNRCYYYN...TNARHCNYND`	2 501 (82.1)	297 895 (211.4)	300 506 (96.7)
`TMYNRRBYTS...YCTHNGCMTC`	4 843 (159.0)	567 485 (402.8)	567 532 (182.6)

7 Conclusions

We summarize the achieved results and point to further research questions arising from this thesis.

The main goal of the present work was to develop a practical motif discovery algorithm able to find motifs with optimal p-values with respect to complex background text models. This goal has been achieved.

On the way towards it, Chapter 2 introduces the concepts of *deterministic arithmetic automata* and *probabilistic arithmetic automata*. We prove that they allow calculating the distributions of values resulting from deterministic computations on random texts with respect to arbitrary finite-memory text models. This technique is a valuable tool in multiple contexts. In this thesis, we use it three times; first, to compute the distribution of the number of occurrences of a pattern in a random string, second, to compute the distribution of the running time cost incurred by window-based pattern matching algorithms, and, third, to compute the distribution of clump sizes. All of these applications are interesting theoretical topics in themselves and, in all three cases, our results go beyond those known previously.

A promising topic for future work is to study the *asymptotic behavior of PAAs*. Establishing precise conditions under which the distribution of values computed by the PAA is asymptotically *normal* would further increase the utility of the PAA framework. For the second application of analyzing window-based pattern matching algorithms, this could allow proving that the number of character accesses is normally distributed for (asymptotically) long texts. Such results are already known for Horspool's algorithm (Mahmoud et al., 1997; Smythe, 2001; Tsai, 2006) but apparently not for B(N)DM and BOM.

In Section 2.7, we give a general DAA construction applicable to all window-based pattern matching algorithms. Whether there exists an algorithm to *construct the minimal DAA* directly, that is, in time linear in the number of states, is another interesting open research question.

We address a similar question in Chapter 3. There, we give algorithms to construct *simple NFAs*. We prove that these NFAs are transformed into minimal DFAs by the classical subset construction and show how they can be built from (sets of) generalized strings and from consensus strings with a Hamming neighborhood. These results do not guarantee that the resulting minimal DFA is small, but avoid the need to construct more DFA states than necessary. This efficient construction procedure is useful in the context of the present thesis as DFAs accepting instances of a motif need to be constructed in order to compute the motif's p-value. In future work, ways of *constructing simple NFAs for other pattern types* might be explored. For instance, we might investigate whether simple NFAs accepting *sets of* generalized strings with a Hamming neighborhood or even with a neighborhood defined by the *edit distance* can

efficiently be built.

The two main contributions made in Chapter 4 are the derivation of a formula for the *expected clump size* of motifs and the use of the PAA framework to compute the whole *distribution of clump sizes*, both for arbitrary finite-memory text models. From the exact distribution of clump sizes, a *compound Poisson* approximation to the distribution of the number of motif occurrences can be constructed. Again, the value of these results is twofold. On the one hand, they are interesting from a theoretical point of view and extend the field of motif statistics. On the other hand, they are of central importance to the motif discovery algorithm developed in Chapter 5.

The largest part of Chapter 5 is devoted to developing bounds for the p-values of partially known motifs and putting them to work in a branch-and-bound motif discovery algorithm. This approach becomes possible by approximating a motif's p-value using the compound Poisson distribution developed in Chapter 4. The most difficult part is to obtain upper bounds for the expected clump size of partially known motifs, which we succeed to do based on the exact formula for the expected clump size obtained in Chapter 4. Finally, we arrive at an algorithm solving Problems 1 and 2, that is, at an algorithm able to discover a motif with optimal p-value, either with respect to the total number of occurrences or with respect to the number of sequences the motif occurs in. To the best of our knowledge, this is the first practical motif discovery algorithm that finds motifs with provably optimal p-values with respect to high-order text models.

Moreover, we show how the algorithm can be generalized to take the double-stranded nature of DNA into account and simultaneously search for a motif and its reverse complement. Giving special care to this problem is important as the statistical properties of such a joint motif are strongly affected by its overlapping structure, meaning that the naive approach of searching the input sequences plus their reverse complements is not accurate.

In contrast to many other published motif discovery algorithms, implementations of the methods developed in this thesis are publicly available. They are part of the MoSDi software package distributed under the terms of the GNU General Public License.

In Chapter 6, we use the carefully crafted benchmark suite of Sandve et al. (2007) to compare MoSDi against Weeder (Bailey and Elkan, 1994) and MEME (Pavesi et al., 2004). Fauteux et al. (2008) use the same benchmark suite and report on the performance of their algorithm Seeder and additionally on the performance of BioProspector (Liu et al., 2001), GibbsSampler (Lawrence et al., 1993), and MotifSampler (Thijs et al., 2001). MoSDi outperforms all these algorithm in terms of the average nucleotide-level correlation coefficient (nCC). Comparing it with MEME, which can be seen as the most mature and established motif discovery tool, MoSDi's nCC scores were better by 36.1 % and 44.5 % for the benchmark suites *Algorithm Markov* and *Algorithm real*, respectively, but at the cost of using hours of CPU time instead of using seconds. Therefore, MEME is an excellent choice for a quick scan for motifs. Keeping in mind, however, that many data sets to be searched for motifs are the result of tedious (and sometimes costly) experimentation, it seems appropriate to invest the computational time needed by MoSDi.

The implementation of MoSDi allows us to distribute the workload across multiple

computers, which permits it to be run on compute clusters. Additionally, it seems promising to port the algorithm to *massively parallel hardware* like GPUs to fully exploit the resources found in today's computers. On the usability side, a *web frontend* to the current command-line tools would probably broaden MoSDi's use.

Besides benchmarking MoSDi against other algorithms, we applied it to the non-coding regions of *M. tuberculosis* and to four data sets of CpG-rich regions in the human genome. Both applications were selected for their biological relevance. *M. tuberculosis* is a human pathogen and CpG islands are known to be involved in gene (dys)regulation (Suzuki and Bird, 2008) and to have an important role in the development of cancer (Jones and Baylin, 2007).

Motif discovery revealed 24 motifs in non-coding regions of *M. tuberculosis*. Five of them could be attributed to known functions. The biological relevance (if any) of the other motifs remains unknown to us and should be made subject of further investigations. The interpretation of the results could possibly be aided by devising a method to compute the probability of observing a certain *increase* in the number of occurrences when a Hamming neighborhood is added. Motif 22 shown on Page **??**, for instance, occurs 30 times exactly and 50 times when a Hamming distance of one is allowed. Now it might be interesting to compute the probability that this increase from 30 to 50 happens by chance. This problem can be approached by using the PAA framework.

In the data sets of tissue-specifically methylated CpG-rich regions, we discovered three strikingly strong motifs. These motifs turned out not to be specific to these regions, but are present throughout the whole human genome, including regions upstream of genes. Elucidating the role of these motifs is a fascinating topic for future research. A thorough analysis of their *placement* and a search for similar sequence features in genomes of *related species* might serve as starting points.

On the side of algorithmics, many further topics can be pursued starting from the motif discovery algorithm developed in this work. Here, we mainly focused on Markovian text models. Using *arbitrary finite-memory text models* is possible, as long as bounds for certain conditional probabilities as given by Inequality (5.9) can be provided. One possible way of obtaining such bounds for arbitrary finite-memory text models is sketched in Remark 5.19 on Page 103. The quality of these bounds and the runtime of motif discovery when they are used should be studied in the future.

One interesting modification of the algorithm would be to score a motif by counting the *number of clumps* instead of the number of its occurrences. As the Poisson distribution accurately approximates the distribution of the number of clumps, this would simplify the calculation of p-values. To implement a feasible motif discovery algorithm, however, the problem of devising an index structure able to efficiently determine the number of clumps of a given IUPAC motif in the input sequences would have to be solved first.

Furthermore, the algorithm can be extended to perform *multi-objective* optimization and return a *Pareto set* instead of a single optimal solution. A Pareto set with respect to multiple objective functions is a set of solutions such that no solution in the set dominates another solution in the set, where one solution is said to dominate another if its scores are superior with respect to *all* considered objective functions. Algorithm 5.1

can straightforwardly be extended to find the optimal Pareto set for multiple objectives. Then, a motif prefix is skipped if bounds on the scores for all objective functions imply that an already-known solution dominates all continuations of this prefix. A wealth of combinations of objectives can be thought of. For example, the number of total occurrences and the number of matching sequences might be optimized simultaneously. Another option is to simultaneously optimize p-values with respect to multiple text models. Then, for instance, we might discover motifs that are overrepresented with respect to text models estimated from coding sequences, intergenic regions, *and* CpG islands.

In this thesis, motif discovery is performed on sequences alone without taking any further information into account. There are multiple resources of additional information that could be or have already been shown to be helpful for discovering biologically relevant motifs. These include evolutionary conservation, DNA methylation, and histone modifications. All three can be modeled as *position-specific numerical annotations* of the primary sequences. In the case of methylation, for instance, we might associate 1 with all methylated nucleotides and 0 with all unmethylated nucleotides. In the case of evolutionary conservation, we might use a (discretized) position-specific score giving the degree of conservation for each nucleotide. We could then consider the sum of these annotations at positions covered by motif occurrences and compute a p-value for this annotation sum. Especially in combination with multi-objective optimization, this approach should (further) be explored.

Implementations of multi-objective motif discovery and objective functions using sums of sequence annotations have already been integrated into MoSDi, but require further evaluation, formal proofs of correctness, and studies about the effects of clumping on annotation sum scores. Preliminary results suggest this approach to be a promising area of future research.

The present thesis shows that exact and statistically sound motif discovery is indeed feasible. This result is encouraging and should be understood as a starting point to extend the developed techniques beyond primary sequence data. Hopefully, the methods researched in this work can make a small contribution to the ongoing quest of understanding living cells.

A Appendix

A.1 MoSDi Software

The MoSDi software is available at

`http://mosdi.googlecode.com/`

under the terms of the *GNU General Public License* (GPL). At the time of writing, the current version is MoSDi 1.2.

In Section A.1.1, we give an overview on the functionality implemented in MoSDi, while common use cases are discussed in Section A.1.2.

A.1.1 Subcommands

MoSDi contains four command-line tools:

- `mosdi-stat`,
- `mosdi-discovery`,
- `mosdi-pm-analysis`, and
- `mosdi-utils`.

Each command must be followed by one of the available subcommands listed below. To get further information on precise usage, each subcommand can be called without parameters.

Subcommands of `mosdi-stat`	
`count-dist`	Calculates the distribution of a IUPAC pattern's occurrence count, either exactly (see Section 2.6) , using a Poisson, or a compound Poisson approximation (see Section 4.3). A Markovian background model is estimated from the input sequences or from a table of q-gram frequencies. This subcommand can consider reverse complements and add a Hamming neighborhood to IUPAC motifs.
`empiric-clump-stat`	Estimates the distribution of clump sizes and the expected clump size by simulation. Random texts are sampled from a text model supplied in the form of a table of q-gram frequencies.

`comp-poisson-eval`	Compares a compound Poisson approximation against the exact distribution of occurrence counts for given motifs. This subcommand was used to generate Figures 4.5 and 4.6.
`min-max-table`	Calculates a table with the minimum and maximum probabilities (over arbitrary conditions imposed on history or future) for each IUPAC character. The maximum values displayed correspond to $P_{\max}(g\|\Sigma)$ defined in Equation (5.18) on Page 101.
`exp-clump-size`	Calculates a motif's expected clump size.
`annotation-sum-dist`	Assuming that a real-valued annotation is given for each character in the input sequences, this subcommand computes the (exact or approximate) distribution of the sum of such annotation at positions corresponding to motif instances.
`clump-size-bounds`	Enumerates motifs and calculates bounds for the expected clump sizes for all their prefixes. This subcommand was used to generate Figure 5.3.

Subcommands of `mosdi-discovery`

`discovery`	Runs motif discovery on (constrained) IUPAC patterns. This subcommand implements Algorithm 5.1. It can be configured to optimize with respect to the total number of occurrences or with respect to the number of input sequences containing at least one motif instance. Furthermore, constraints on the use of wildcard characters or on the motif's expectation can be given. To allow parallel execution (possibly on different hosts), a range of motifs to be searched may also be set. Subcommand `iupac-ranges` of `mosdi-utils` can be used to split a motif space into equally-sized portions.
`local-search`	Tries to improve a given motif using an (iterated) neighborhood search. In each iteration, all motifs in a neighborhood are evaluated and the one with best p-value is retained. Stops when no further improvement can be achieved.
`calc-scores`	Calculates scores for given motifs. Useful for re-evaluation on other sequences or using other text models. Additionally, given the true answers, calculates true/false positives/negatives and further statistics.
`p-value-bounds`	Enumerates motifs and calculates p-value bounds for all motif prefixes. This subcommand can be used to evaluate the quality of p-value bounds.

Subcommands of `mosdi-pm-analysis`	
`cost-distribution`	Computes the cost distribution for a given algorithm and a given pattern using the techniques introduced in Section 2.7. A Markovian text model can be supplied as a table of q-gram frequencies.
`automata-sizes`	Prints the number of states of unminimized DAA, minimized DAA, and PAA for a given algorithm and for all patterns of a given length. When run in *verbose* mode, the DAAs are also given in textual representation (command line: `mosdi-pm-analysis -v2 automata-sizes`).

Subcommands of `mosdi-utils`	
`iupac-gen`	Enumerates IUPAC strings that meet given constraints on the multiplicity of wildcards.
`iupac-ranges`	Splits a motif space into ranges.
`iupac-abelian-gen`	Enumerates abelian patterns of IUPAC characters.
`annotate`	Creates an annotation track in EMBL format that contains all matches of a given IUPAC pattern.
`count-matches`	Reports the number of matches of a given IUPAC pattern.
`generate-pfm`	Given a IUPAC pattern and sequences, creates a position frequency matrix (PFM) for all matches of the pattern in the sequences.
`pfm-to-pwm`	Given a position frequency matrix (PFM) and a character distribution, outputs a position weight matrix (PWM) containing log-odds scores.
`pwm-to-iupac`	Given a position weight matrix (PWM) and a threshold t, generates a set of IUPAC strings such that they match a string if and only if the PWM score for this string is larger than t.
`random-text`	Generates a random text according to a given text model.
`random-copy`	Generates random sequences similar (in length, number, and composition) to template sequences given in FASTA format.
`cut-out-motif`	Removes a given motif from given sequences. Optionally, the motifs can be masked by a user-specified character instead of being excised.
`count-qgrams`	Counts q-grams in a given set of sequences.

`qgram-expectations`	Computes expectation of all q-grams of given length with respect to a given text model.

A.1.2 Use Cases

In this section, we give examples of common use cases. MoSDi is designed as a collection of command-line tools rather than one monolithic application. Some (but not all) tasks require it to call several subcommands. In the following, we give examples of such MoSDi invocations.

Distribution of Occurrence Counts

The distribution of the number of occurrences of a IUPAC motif, say $p =$ `AYAYT`, in a random text of a given length can be computed using the subcommand `count-dist` of `mosdi-stat`. That means it computes the probability $\mathbb{P}\left(occ_p\left(S_0 \cdots S_{m-1}\right) = k\right)$ for all values of k from zero to a user-specified maximum `<max>`.

```
mosdi-stat count-dist -n <length> -m <max> exact AYAYT
```

where `<length>` is the length of the random text.

The parameter `exact` says that the exact distribution should be computed. This can be slow when the text length is large. Adding the option `-C` lets the program estimate runtimes for the basic and the doubling algorithm given in Section 2.4.2 and uses the favorable one. By default, the distribution is computed with respect to a uniform i.i.d. text model. To use a different text model, a table of q-gram frequencies can be supplied. Using a third order Markovian text model estimated from the human genome can be done as follows.

```
mosdi-utils count-qgrams hg18.fa 4 > hg18.4grams
mosdi-stat count-dist -q hg18.4grams -n <length> -m <max> exact AYAYT
```

To compute the *compound Poisson* approximation that is discussed in Section 4.3, the parameter `exact` is replaced by `comp-poisson`.

Compute Motif Significance

The p-value for the number of occurrences is obtained as the tail probability of the distribution of occurrence counts discussed above. When the option `-p` is given to the `count-dist` subcommand, the number of motif instances k in the supplied text is determined and the tail probability $\mathbb{P}\left(occ_p\left(S_0 \cdots S_{m-1}\right) \geq k\right)$ is computed.

```
mosdi-stat count-dist -p input.fasta -r -H 1 comp-poisson AYAYT
```

Here, we have also added the options `-r` and `-H 1` to consider `AYAYT` jointly with its reverse complement and to also count matches within Hamming distance 1.

Run Motif Discovery

We now run Algorithm 5.1 to discover motifs. Assume we want to scan our input sequences for motifs of length eight and optimize the p-value with respect to the total occurrence count.

```
mosdi-discovery -v discovery 8 occ-count input.fasta
```

The verbosity switch `-v` is useful to get progress information. By default, the program `mosdi-discovery` allows patterns composed of {`A`, `C`, `G`, `T`} and at most `<length>`/2 instances of the wildcard `N`. To allow other wildcards (at the expense of running time), the switch `-M` is used.

```
mosdi-discovery -v discovery -E 5.0 -M 8,2,0,3 8 occ-count input.fasta
```

Here we allow eight characters from {`A`, `C`, `G`, `T`}, two from {`W`, `S`, `R`, `Y`, `K`, `M`}, zero from {`B`, `D`, `H`, `V`}, and three `N`s. Furthermore, we have added `-E 5.0` to restrict the search to patterns with expectation smaller than or equal to five. In most practical settings, we are not interested in patterns with high expectation and restricting the search space in this way can significantly reduce running time. Another option to speed up the search is to provide an initial p-value threshold.

```
mosdi-discovery -v discovery -I 1e-30 8 occ-count input.fasta
```

Then, parts of the search space that cannot contain motifs with a p-value better than 10^{-30} are skipped. If no motif with a better p-value exists, none are returned. If one exists, the output is the same as without option `-I`. To extract *all* motifs with a p-value better than a threshold, the option `-T` can be used.

```
mosdi-discovery -v discovery -T 1e-50 8 occ-count input.fasta
```

This option should be used thoughtfully as the output can be large for high thresholds.

In order to optimize the p-value with respect to the number of sequences a motif occurs in, the argument `occ-count` is replaced by `seq-count`.

```
mosdi-discovery -v discovery 8 seq-count input.fasta
```

By default, an i.i.d. text model estimated from the input sequences is used as a background model. To use a higher-order model, the parameter `-O` is used to specify the desired order. If the background should not be estimated from the input sequence it can also given in the form of a table of q-gram frequencies using the option `-q`.

For large motif spaces, long input texts and high-order text models, exact motif discovery can be slow. In this case, we might want to parallelize the search. To distribute the work load, `mosdi-discovery` can be instructed to search only parts of the search space using the options `-l` and `-u`:

```
mosdi-discovery -v discovery -l C -u CCGG 8 occ-count input.fasta
```

Now, it only scans motifs lexicographically between C and CCGG. To split a motif space into equally-sized chunks, we use the `iupac-ranges` subcommand.

```
mosdi-utils iupac-ranges -M 8,2,0,3 100 8
```

This results in a list of hundred ranges that contain all motifs of length eight with the specified maximal number of wildcards.

After an optimal motif has been discovered, we might want to search for other motifs. One procedure to do this is to excise or mask all instances of the optimal motif from the input sequence and repeat the search on the resulting sequences.

```
mosdi-utils cut-out-motif input.fasta AYAYT > new-input.fasta
```

A.2 IUPAC Codes for DNA

Code	Set	Explanation	(Watson-Crick) Complement
A	{A}	Adenine	T
C	{C}	Cytosine	G
G	{G}	Guanine	C
T	{T}	Thymine	A
R	{A, G}	puRine	Y
Y	{C, T}	pYrimidine	R
S	{C, G}	Strong bond	S
W	{A, T}	Weak bond	W
K	{G, T}	Keto	M
M	{A, C}	aMino	K
B	{C, G, T}	not A	V
D	{A, G, T}	not C	H
H	{A, C, T}	not G	D
V	{A, C, G}	not T	B
N	{A, C, G, T}	aNy	N

A.3 Matrix Theory

In this section, we gather some definitions and facts on matrices. A good reference on the topic is the quite exhaustive book by Bernstein (2009). We adopt the same nomenclature as used there.

Definition A.1 (Matrix norm)**.** A function $\|\cdot\| : \mathbb{R}^{n\times m} \to \mathbb{R}_0^+$ is called *norm* if it satisfies the following conditions:

1. $\|A\| \geq 0$ for all $A \in \mathbb{R}^{n\times m}$,
2. $\|A\| = 0$ if and only of $A = \mathbf{0}$,
3. $\|\alpha A\| \leq |\alpha|\|A\|$ for all $\alpha \in \mathbb{R}$ and $A \in \mathbb{R}^{n\times m}$,
4. $\|A + B\| \leq \|A\| + \|B\|$ for all $A, B \in \mathbb{R}^{n\times m}$.

$\Diamond$

Definition A.2 (Submultiplicative norm)**.** A norm $\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}_0^+$ is called *submultiplicative*, if $\|AB\| \leq \|A\|\|B\|$ for all $A, B \in \mathbb{R}^{n\times n}$. $\Diamond$

Definition A.3. Let $A_i \in \mathbb{R}^{n\times m}$ for $i \in \mathbb{N}_0$. If the series $\sum_{i=0}^{\infty} \|A_i\|$ converges, $\sum_{i=0}^{\infty} A_i$ is said to be *absolutely convergent*. $\Diamond$

Lemma A.4. *If $\sum_{i=0}^{\infty} A_i$ is absolutely convergent, i.e. the series $\sum_{i=0}^{\infty} \|A_i\|$ converges, then $\sum_{i=0}^{\infty} A_i$ converges.*

Proof. Bernstein (2009), Proposition 10.2.9. □

Definition A.5 (Spectrum, spectral radius)**.** Let $A \in \mathbb{R}^{n\times n}$. The set of all *eigenvalues* of A is called *spectrum* of A, written $\mathrm{spec}(A)$. That means, $\mathrm{spec}(A)$ is the set of all $\lambda \in \mathbb{R}$ for which a non-zero vector $|x\rangle \in \mathbb{R}^n$ with $A|x\rangle = \lambda|x\rangle$ exists. The *spectral radius* of A, written $\mathrm{sprad}(A)$, is defined as

$$\mathrm{sprad}(A) := \max\left\{|\lambda| : \lambda \in \mathrm{spec}(A)\right\}.$$

$\Diamond$

Lemma A.6. *Let $A \in \mathbb{R}^{n\times n}$ be given such that $\mathrm{sprad}(A) < 1$. Then, there exists a submultiplicative norm $\|\cdot\|$ such that $\|A\| < 1$. Furthermore, the series $\sum_{k=0}^{\infty} A^k$ converges absolutely and*

$$\sum_{k=0}^{\infty} A^k = (\mathbf{1} - A)^{-1}.$$

Proof. Bernstein (2009), Proposition 9.4.13. □

Lemma A.7. *Let $A \in \mathbb{R}^{n\times n}$ be given such that $\mathrm{sprad}(A) < 1$. Then, $\sum_{k=1}^{\infty} kA^{k-1}$ converges absolutely and*

$$\sum_{k=1}^{\infty} kA^{k-1} = (\mathbf{1} - A)^{-2}.$$

Proof. Let $\|\cdot\|$ be a submultiplicative norm with $\|A\| < 1$. Such a norm exists according to Lemma A.6. We first verify that the series converges absolutely:

$$\sum_{k=1}^{\infty} \|kA^{k-1}\| = \sum_{k=1}^{\infty} k\|A^{k-1}\| \overset{(i)}{\leq} \sum_{k=1}^{\infty} k\|A\|^{k-1} = \sum_{k=1}^{\infty} \Big(\underbrace{e^{\frac{\log k}{k-1} + \log \|A\|}}_{<1 \text{ for large } k} \Big)^{k-1} < \infty \,,$$

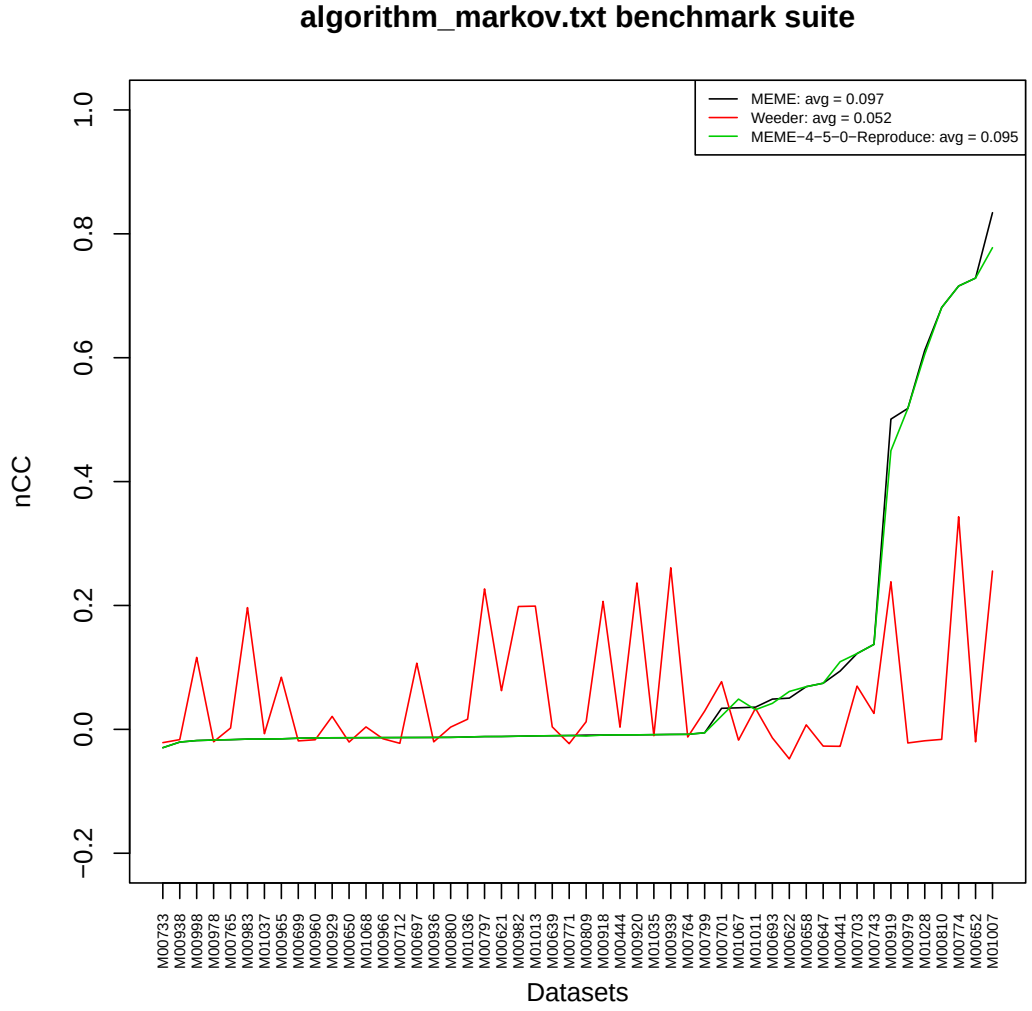
where (i) holds as $\|\cdot\|$ is submultiplicative. By applying Lemma A.6, we obtain

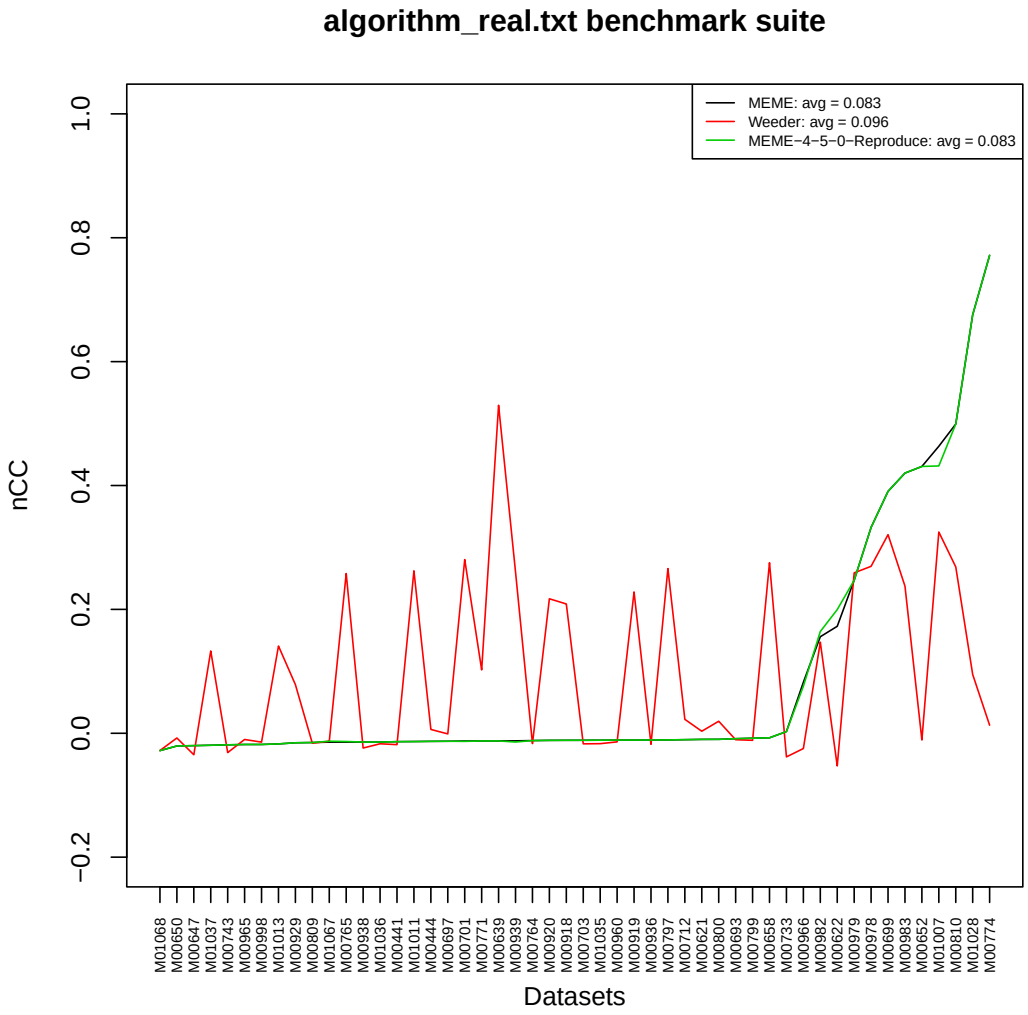
$$(\mathbf{1} - A)^{-2} = \left(\sum_{k=0}^{\infty} A^k \right)^2 = \sum_{k=1}^{\infty} kA^{k-1} \,.$$

□

A.4 Verification of MEME Results of Sandve et al. (2007)

As discussed in Section 6.2, we re-ran MEME with default parameters to reproduce the results of Sandve et al. (2007). The following two plots were generated by their website and show the results they obtained for MEME (black line) and Weeder (red line) in comparison to the uploaded predictions (green line).




algorithm_real.txt benchmark suite
MEME: avg = 0.083
Weeder: avg = 0.096
MEME−4−5−0−Reproduce: avg = 0.083
nCC
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
M01068
M00650
M00647
M01037
M00743
M00965
M00998
M01013
M00929
M00809
M01067
M00765
M00938
M01036
M00441
M01011
M00444
M00697
M00701
M00771
M00639
M00939
M00764
M00920
M00918
M00703
M01035
M00960
M00919
M00936
M00797
M00712
M00621
M00800
M00693
M00799
M00658
M00733
M00966
M00982
M00622
M00979
M00978
M00699
M00983
M00652
M01007
M00810
M01028
M00774
Datasets

A.5 Contributions to Co-Authored Articles

As required by §8(2) of the *Promotionsordnug der Universität Dortmund für den Fachbereich Informatik vom 26.11.2003*, a declaration of the author's contributions to co-authored articles that are part of this thesis follows.

I am the main author of the following articles co-authored with Sven Rahmann. As my advisor, Sven Rahmann participated in and advised me at all stages of research, from discussing the initial ideas to writing up the articles.

- Marschall and Rahmann (2008). *Probabilistic Arithmetic Automata and their Application to Pattern Matching Statistics*.
- Marschall and Rahmann (2009). *Efficient Exact Motif Discovery*.
- Marschall and Rahmann (2010a). *Exact Analysis of Horspool's and Sunday's Pattern Matching Algorithms with Probabilistic Arithmetic Automata*. An extended version has been submitted. A preprint is available from arXiv (Marschall and Rahmann, 2010c).
- Marschall and Rahmann (2010b). *Speeding up Exact Motif Discovery by Bounding the Expected Clump Size*.

The following article is a unifying survey on the framework of *Probabilistic Arithmetic Automata* and its applications.

> Marschall, Herms, Kaltenbach, and Rahmann (submitted). *Probabilistic Arithmetic Automata and their Applications*. A preprint is available from arXiv (Marschall et al., 2010).

Sections 2, 3, 4, 5, 6, and 7 of this article have mainly been written by me, while Sections 8, 9, and 10 contain contributions mainly made by my co-authors. All authors have participated in preparing the manuscript.

List of Symbols

$\langle x\|$	Row vector, page 7
$\|x\rangle$	Column vector, page 7
$*$	Convolution of two probability distributions, page 6
$[\![A]\!]$	Iverson bracket, $[\![A]\!] = 1$ if the statement A is true and $[\![A]\!] = 0$ otherwise, page 6
$s \lhd g$	True if $s \in \Sigma^*$ matches the generalized string $g \in \mathcal{G}^*$, page 10
2^K	Power set of the set K, page 6
$s[..i]$	Prefix of $s \in \Sigma^*$, i.e. $s[..i] = s[0 \dots i]$, page 6
$\overleftarrow{s}$	Reverse of $s \in \Sigma^*$, $\overleftarrow{s} = s[\|s\| - 1] \dots s[0]$, page 6
$s[i \dots j]$	Substring of s, i.e. $s[i \dots j] = s[i] \dots s[j]$, page 6
$s[i..]$	Suffix of $s \in \Sigma^*$, i.e. $s[i..] = s[i \dots \|s\| - 1]$, page 6
$\mathbf{0}$	Zero matrix, page 7
$\mathbf{1}$	Identity matrix, page 7
$\|0\rangle$	Vector composed of zeros, page 7
$\|1\rangle$	Vector composed of ones, page 7
$\mathcal{B}_{n,p}$	Binomial distribution, where n is the number of trials and p the success probability of a single trial, page 21
$\mathcal{C}$	Set of contexts, part of a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$, page 24
$\mathcal{CP}_{\lambda,\Psi}$	Compound Poisson distribution with expected clump number λ and clump size distribution Ψ, page 22
C_t	Random variable with values in $\mathcal{C}$ giving the text model state after t steps, page 24
E_t	Random variable with values in $\mathcal{E}$ giving the emission of a PAA in step t, page 30
$\mathcal{E}$	Emission set, part of a PAA, page 28
F	Set of accepting states of a DFA, $F \subset \mathcal{Q}$, page 19
$\mathcal{G}$	Generalized alphabet, $\mathcal{G} = 2^\Sigma \setminus \{\emptyset\}$, page 10
K	Overlap matrix for the set of words $\mathcal{W}$, page 72
$\mathcal{L}(X)$	Distribution (law) of the random variable X, page 6
ℓ	Length of a pattern, page 6

$\mathcal{M}$	Motif space, $\mathcal{M} \subseteq \mathcal{G}^\ell$, page 85
N	Total length of input strings, $N := \sum_{s \in \mathcal{S}} \lvert s \rvert$, page 6
$\mathbb{N}$	Set of natural numbers (excluding zero), page 6
$\mathbb{N}_0$	Set of natural numbers (including zero), page 6
$\mathcal{P}_\lambda$	Poisson distribution with expectation λ, page 22
$\mathbb{P}$	Probability measure over a random text, page 7
Q_t	Random variable with values in $\mathcal{Q}$ giving the state of a PAA after t steps, page 30
$\mathcal{Q}$	Set of states of an automaton, for example of a DFA, NFA, DAA, or PAA, page 19
$\mathbb{R}$	Set of real numbers, page 6
$\mathbb{R}^+$	Set of strictly positive real numbers, page 6
$\mathbb{R}_0^+$	Set of positive real numbers including zero, page 6
$\mathbb{R}^{n \times m}$	Set of $n \times m$ matrices over $\mathbb{R}$, page 7
S_t	Random variable giving the t-th character in a random text, page 7
S_t^ℓ	Substring of a random text, $S_t^\ell = S_t \cdots S_{t+\ell-1}$, page 7
$\mathcal{S}$	Finite set of input strings, $\mathcal{S} \subset \Sigma^*$, page 6
T	Transition function of a PAA, $T : \mathcal{Q} \times \mathcal{Q} \to [0, 1]$, page 28
V_t	Random variable with values in $\mathcal{V}$ giving the value computed by a PAA after t steps, page 30
$\mathcal{V}$	Value set, part of a PAA, page 28
$\mathcal{W}$	Non-empty set of words $\mathcal{W} \subset \Sigma^\ell$ constituting a motif, page 70
$X_n^{A,p}$	Random variable giving the number of character accesses made by algorithm A to search for the pattern p in the random text $S_0 \cdots S_{n-1}$, page 44
X_t	Joint random variable of word and context, $X_t := (S_t^\ell, C_t)$, i.e. $X_t = (w, c)$ means that word $w \in \Sigma^\ell$ starts at position t in S, and before generating its first letter, we are in context c, page 71
$\mathcal{X}$	All pairs $(w, c) \in \mathcal{W} \times \mathcal{C}$ with positive (asymptotic) probability, page 71
$Z_i^{\mathcal{W}}$	Random variable giving the size of the i-th clump of words from $\mathcal{W}$ in the random text $(S_t)_{t \in \mathbb{N}_0}$, the superscript $\mathcal{W}$ may be omitted, page 71
δ	Transition function of a DFA and/or DAA, $\delta : \mathcal{Q} \times \Sigma^* \to \mathcal{Q}$, page 19
Δ	Non-deterministic transition function of an NFA, $\Delta : \mathcal{Q} \times \Sigma^* \to 2^{\mathcal{Q}}$, page 19
$\Delta_\circlearrowleft$	Modified transition function of an NFA where self-transitions have been added to all start states, page 56
ε	Empty string, page 6

μ_q	Emission distribution associated with state $q \in \mathcal{Q}$ of a PAA, $\mu_q : \mathcal{E} \to [0,1]$, page 28
φ	Transition function, $\varphi : \mathcal{C} \times \Sigma \times \mathcal{C} \to [0,1]$, part of a finite-memory text model $(\mathcal{C}, c_0, \Sigma, \varphi)$, page 24
ψ_p	Expected clump size of pattern p, page 71
Ψ_p	Clump size distribution for a pattern p, page 76
Σ	Finite alphabet, page 6
Σ^*	Set of all strings of finite length over alphabet Σ, page 6
θ_q	Operation associated with state $q \in \mathcal{Q}$ of a PAA, $\theta_q : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$, page 28
ϑ_n	Maximal number of different PAA values, $\vartheta_n = \max_{0 \le t \le n} \lvert\operatorname{range}(V_t)\rvert$, page 30
$\xi^{A,p}(w)$	Cost caused by algorithm A when searching window w for pattern p, page 45
ζ_p	Expectation of the number of occurrences of the pattern p in a random text of length $\lvert p\rvert$, i.e. $\zeta_p := \mathbb{P}\left(S_0^\ell \lhd p\right)$, page 89

Bibliography

M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

P. Agarwal and V. Bafna. Detecting non-adjoining correlations within signals in DNA. In *Proceedings of the Second Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 2–8, 1998.

A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Fifth edition, Garland Science, 2007.

C. Allauzen, M. Crochemore, and M. Raffinot. Efficient experimental string matching by weak factor recognition. In *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 51–72, 2001.

A. Apostolico and C. Tagliacollo. Incremental discovery of the irredundant motif bases for all suffixes of a string in $o(n^2 \log n)$ time. *Theoretical Computer Science*, 408(2-3): 106–115, 2008.

A. Apostolico, M. E. Bock, S. Lonardi, and X. Xu. Efficient detection of unusual words. *Journal of Computational Biology*, 7(1-2):71–94, 2000.

A. Apostolico, M. E. Bock, and S. Lonardi. Monotony of surprise and large-scale quest for unusual words. *Journal of Computational Biology*, 10(3–4):283–311, 2003.

R. A. Baeza-Yates and M. Régnier. Average running time of the Boyer-Moore-Horspool algorithm. *Theoretical Computer Science*, 92(1):19–31, 1992.

R. A. Baeza-Yates, G. H. Gonnet, and M. Régnier. Analysis of Boyer-Moore-type string searching algorithms. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 328–343, 1990.

T. L. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 28–36, 1994.

Y. Barash, G. Elidan, N. Friedman, and T. Kaplan. Modeling dependencies in protein-DNA binding sites. In *Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 28–37, 2003.

A. D. Barbour and O. Chryssaphinou. Compound Poisson approximation: a user's guide. *The Annals of Applied Probability*, 11(3):964–1002, 2001.

R. Barrangou, C. Fremaux, H. Deveau, M. Richards, P. Boyaval, S. Moineau, D. A. Romero, and P. Horvath. CRISPR provides acquired resistance against viruses in prokaryotes. *Science*, 315(5819):1709–1712, 2007.

F. Bassino, J. Clément, J. Fayolle, and P. Nicodème. Constructions for clumps statistics. In *Proceedings of the Fifth Colloquium on Mathematics and Computer Science*, pages 179–194, 2008.

K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

D. S. Bernstein. *Matrix Mathematics*. Second edition, Princeton University Press, 2009.

C. Bi. A Monte Carlo EM algorithm for de novo motif discovery in biomolecular sequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(3): 370–386, 2009.

P. Billingsley. *Probability and Measure*. Third edition, Wiley Interscience, 1995.

V. Boeva, J. Clément, M. Régnier, M. A. Roytberg, and V. J. Makeev. Exact p-value calculation for heterotypic clusters of regulatory motifs and its application in computational annotation of cis-regulatory modules. *Algorithms for Molecular Biology*, 2: 13, 2007.

R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen. Predicting gene regulatory elements in silico on a genomic scale. *Genome Research*, 8(11):1202–1215, 1998.

P. Brémaud. *Markov Chains*. Springer, 1999.

J. Buhler and M. Tompa. Finding motifs using random projections. *Journal of Computational Biology*, 9(2):225–242, 2002.

A. M. Carvalho, A. T. Freitas, A. L. Oliveira, and M. F. Sagot. An efficient algorithm for the identification of structured motifs in DNA promoter sequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):126–140, 2006.

B. C. Chang, A. Ratnaweera, S. K. Halgamuge, and H. C. Watson. Particle swarm optimisation for protein motif discovery. *Genetic Programming and Evolvable Machines*, 5(2):203–214, 2004.

C. Chen, B. Schmidt, L. Weiguo, and W. Müller-Wittig. GPU-MEME: Using graphics hardware to accelerate motif finding in DNA sequences. In *Proceedings of the Third IAPR International Conference on Pattern Recognition in Bioinformatics (PRIB)*, pages 448–459, 2008.

F. Y. L. Chin and H. C. M. Leung. Voting algorithms for discovering long motifs. In *Proceedings of the Third Asia-Pacific Bioinformatics Conference (APBC)*, pages 261–271, 2005.

E. Cinlar. Markov additive processes I. *Z. Wahrscheinl. verw. Geb.*, 24(2):85–93, 1972a.

E. Cinlar. Markov additive processes II. *Z. Wahrscheinl. verw. Geb.*, 24(2):95–121, 1972b.

S. T. Cole, R. Brosch, J. Parkhill, T. Garnier, C. Churcher, D. Harris, S. V. Gordon, K. Eiglmeier, S. Gas, C. E. Barry, F. Tekaia, K. Badcock, D. Basham, D. Brown, T. Chillingworth, R. Connor, R. Davies, K. Devlin, T. Feltwell, S. Gentles, N. Hamlin, S. Holroyd, T. Hornsby, K. Jagels, A. Krogh, J. McLean, S. Moule, L. Murphy, K. Oliver, J. Osborne, M. A. Quail, M.-A. Rajandream, J. Rogers, S. Rutter, K. Seeger, J. Skelton, R. Squares, S. Squares, J. E. Sulston, K. Taylor, S. Whitehead, and B. G. Barrell. Deciphering the biology of Mycobacterium tuberculosis from the complete genome sequence. *Nature*, 393:537–544, 1998.

A. Cornish-Bowden. Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984. *Nucleic Acids Research*, 13(9):3021–3030, 1985.

M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter. Speeding up two string-matching algorithms. *Algorithmica*, 12(4–5): 247–267, 1994.

M. Defrance and J. van Helden. info-gibbs: a motif discovery algorithm that directly optimizes information content during sampling. *Bioinformatics*, 25(20):2715–2722, 2009.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

F. Eckhardt, J. Lewin, R. Cortese, V. K. Rakyan, J. Attwood, M. Burger, J. Burton, T. V. Cox, R. Davies, T. A. Down, C. Haefliger, R. Horton, K. Howe, D. K. Jackson, J. Kunde, C. Koenig, J. Liddle, D. Niblett, T. Otto, R. Pettett, S. Seemann, C. Thompson, T. West, J. Rogers, A. Olek, K. Berlin, and S. Beck. DNA methylation profiling of human chromosomes 6, 20 and 22. *Nature Genetics*, 38(12):1378–1385, 2006.

E. Eskin. From profiles to patterns and back again: a branch and bound algorithm for finding near optimal motif profiles. In *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 115–124, 2004.

E. Eskin and P. A. Pevzner. Finding composite regulatory patterns in DNA sequences. *Bioinformatics*, 18(1):354–363, 2002.

F. Fauteux, M. Blanchette, and M. V. V. Strömvik. Seeder: discriminative seeding DNA motif discovery. *Bioinformatics*, 24(20):2303–2307, 2008.

M. Federico, P. Valente, M. Leoncini, M. Montangero, and R. Cavicchioli. An efficient algorithm for planted structured motif extraction. In *Proceedings of the First ACM Workshop on Breaking Frontiers of Computational Biology*, pages 1–6, 2009.

W. Feller. *An Introduction to Probability Theory and its Applications*. Wiley, 1968.

P. Ferragina, R. González, G. Navarro, and R. Venturini. Compressed text indexes: from theory to practice. *Journal of Experimental Algorithmics*, 13:1.12:1–1.12:31, 2009.

G. B. Fogel, D. G. Weekes, G. Varga, E. R. Dow, H. B. Harlow, J. E. Onyia, and C. Su. Discovery of sequence motifs related to coexpression of genes using evolutionary computation. *Nucleic Acids Research*, 32(13):3826–3835, 2004.

M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. First edition, Addison-Wesley, 1999.

E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. MotifCut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

D. J. Galas, M. Eggert, and M. S. Waterman. Rigorous pattern-recognition methods for DNA sequences: analysis of promoter sequences from Escherichia coli. *Journal of Molecular Biology*, 186(1):117–128, 1985.

M. Gardiner-Garden and M. Frommer. CpG islands in vertebrate genomes. *Journal of Molecular Biology*, 196(2):261–282, 1987.

R. Giegerich and S. Kurtz. A comparison of imperative and purely functional suffix tree constructions. *Science of Computer Programming*, 25(2-3):187–218, 1995.

R. van Glabbeek and B. Ploeger. Five determinisation algorithms. In *Proceedings of the 13th International Conference on Implementation and Applications of Automata (CIAA)*, pages 161–170, 2008.

I. Grissa, G. Vergnaud, and C. Pourcel. The CRISPRdb database and tools to display CRISPRs and to generate dictionaries of spacers and repeats. *BMC Bioinformatics*, 8 (1):172, 2007.

D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

J. van Helden, B. André, and J. Collado-Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology*, 281(5):827–842, 1998.

J. van Helden, A. F. Rios, and J. Collado-Vides. Discovering regulatory elements in non-coding sequences by analysis of spaced dyads. *Nucleic Acids Research*, 28(8): 1808–1818, 2000.

I. Herms. *Probabilistic Arithmetic Automata: Applications of a Stochastic Computational Framework in Biological Sequence Analysis*. PhD thesis, Bielefeld University, Germany, 2009.

G. Z. Hertz and G. D. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15(7-8):563–577, 1999.

J. M. Heumann, A. S. Lapedes, and G. D. Stormo. Neural networks for determining protein specificity and multiple alignment of binding sites. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 188–194, 1994.

J. E. Hopcroft. An $n \log n$ algorithm for minimizing the states in a finite automaton. In *The Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

R. N. Horspool. Practical fast searching in strings. *Software-Practice and Experience*, 10: 501–506, 1980.

J. D. Hughes, P. W. Estep, S. Tavazoie, and G. M. Church. Computational identification of cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. *Journal of Molecular Biology*, 296(5):1205–1214, 2000.

R. Illingworth, A. Kerr, D. DeSousa, H. Jørgensen, P. Ellis, J. Stalker, D. Jackson, C. Clee, R. Plumb, J. Rogers, S. Humphray, T. Cox, C. Langford, and A. Bird. A novel CpG island set identifies tissue-specific methylation at developmental gene loci. *PLoS Biology*, 6(1):e22, 2008.

E. S. Jackson and W. J. Fitzgerald. A sequential Monte Carlo EM approach to the transcription factor binding site identification problem. *Bioinformatics*, 23(11):1313–1320, 2007.

K. L. Jensen, M. P. Styczynski, I. Rigoutsos, and G. N. Stephanopoulos. A generic motif discovery algorithm for sequential data. *Bioinformatics*, 22(1):21–28, 2006.

P. A. Jones and S. B. Baylin. The epigenomics of cancer. *Cell*, 128(4):683–692, 2007.

J. Jurka, V. V. Kapitonov, A. Pavlicek, P. Klonowski, O. Kohany, and J. Walichiewicz. Repbase update, a database of eukaryotic repetitive elements. *Cytogenetic and Genome Research*, 110(1-4):462–467, 2005.

H.-M. Kaltenbach. *Statistics and Algorithms for Peptide Mass Fingerprinting*. PhD thesis, Bielefeld University, Germany, 2006.

O. D. King and F. P. Roth. A non-parametric model for transcription factor binding sites. *Nucleic Acids Research*, 31(19):e116, 2003.

D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

T. Knuutila. Re-describing an algorithm by Hopcroft. *Theoretical Computer Science*, 250 (1-2):333–363, 2001.

D. C. Kozen. *Automata and Computability*. Springer, 1999.

G. Kucherov, L. Noé, and M. Roytberg. A unifying framework for seed sensitivity and its application to subset seeds. *Journal of Bioinformatics and Computational Biology*, 4 (2):553–569, 2006.

C. Kunsch, S. M. Ruben, and C. A. Rosen. Selection of optimal κB/Rel DNA-binding motifs: interaction of both subunits of NF-κB with DNA is required for transcriptional activation. *Molecular and Cellular Biology*, 12(10):4412–4421, 1992.

J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 633–642, 1999.

C. Lawrence, S. Altschul, M. Boguski, J. Liu, A. Neuwald, and J. Wootton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262(5131):208–214, 1993.

C. E. Lawrence and A. A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins: Structure, Function, and Genetics*, 7(1):41–51, 1990.

H. C. M. Leung and F. Y. L. Chin. Finding exact optimal motifs in matrix representation by partitioning. *Bioinformatics*, 21(Suppl 2):ii86–ii92, 2005.

L. Li, R. L. Bass, and Y. Liang. fdrMotif: identifying cis-elements by an EM algorithm coupled with false discovery rate control. *Bioinformatics*, 24(5):629–636, 2008.

C. Linhart. The degenerate primer design problem. Master's thesis, School of Computer Science, Tel-Aviv University, 2002.

C. Linhart and R. Shamir. The degenerate primer design problem: theory and applications. *Journal of Computational Biology*, 12(4):431–456, 2005.

R. Lister, M. Pelizzola, R. H. Dowen, R. D. Hawkins, G. Hon, J. Tonti-Filippini, J. R. Nery, L. Lee, Z. Ye, Q. Ngo, L. Edsall, J. Antosiewicz-Bourget, R. Stewart, V. Ruotti, A. H. Millar, J. A. Thomson, B. Ren, and J. R. Ecker. Human DNA methylomes at base resolution show widespread epigenomic differences. *Nature*, 462(7271):315–322, 2009.

D. Liu, X. Xiong, B. DasGupta, and H. Zhang. Motif discoveries in unaligned molecular sequences using self-organizing neural networks. *IEEE Transactions on Neural Networks*, 17(4):919–928, 2006.

X. Liu, D. L. Brutlag, and J. S. Liu. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. *Pacific Symposium on Biocomputing*, 6:127–138, 2001.

M. Lladser, M. D. Betterton, and R. Knight. Multiple pattern matching: A Markov chain approach. *Journal of Mathematical Biology*, 56(1-2):51–92, 2008.

M. Lothaire. *Applied Combinatorics on Words*, volume 105 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2005.

B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. *SIAM Journal on Computing*, 39(4):1432–1443, 2009.

B. Ma, J. Tromp, and M. Li. PatternHunter - faster and more sensitive homology search. *Bioinformatics*, 18:440–445, 2002.

H. M. Mahmoud, R. T. Smythe, and M. Régnier. Analysis of Boyer-Moore-Horspool string-matching heuristic. *Random Structures and Algorithms*, 10(1-2):169–186, 1997.

L. Marsan and M. F. Sagot. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *Journal of Computational Biology*, 7(3-4):345–362, 2000.

T. Marschall. Pattern matching statistics – an approach based on finite automata. Master's thesis, Universität Bielefeld, Germany, 2007.

T. Marschall. Construction of minimal deterministic finite automata from biological motifs. *Theoretical Computer Science*, 412:922–930, 2011.

T. Marschall and S. Rahmann. Probabilistic arithmetic automata and their application to pattern matching statistics. In *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 95–106, 2008.

T. Marschall and S. Rahmann. Efficient exact motif discovery. *Bioinformatics*, 25(12): i356–364, 2009.

T. Marschall and S. Rahmann. Exact analysis of Horspool's and Sunday's pattern matching algorithms with probabilistic arithmetic automata. In *Proceedings of the Fourth International Conference on Language and Automata Theory and Applications (LATA)*, pages 439–450, 2010a.

T. Marschall and S. Rahmann. Speeding up exact motif discovery by bounding the expected clump size. In *Proceedings of the 10th International Workshop on Algorithms in Bioinformatics (WABI)*, pages 337–349, 2010b.

T. Marschall and S. Rahmann. Exact analysis of pattern matching algorithms with probabilistic arithmetic automata. *arXiv*, 1009.6114, 2010c.

T. Marschall, I. Herms, H.-M. Kaltenbach, and S. Rahmann. Probabilistic arithmetic automata and their applications. *arXiv*, 1011.5778, 2010.

G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings*. Cambridge University Press, 2002.

P. Ng and U. Keich. GIMSAN: a Gibbs motif finder with significance analysis. *Bioinformatics*, 24(19):2256–2257, 2008.

P. Nicodème, B. Salvy, and P. Flajolet. Motif statistics. *Theoretical Computer Science*, 287: 593–617, 2002.

G. Nuel. Pattern Markov chains: optimal Markov chain embedding through deterministic finite automata. *Journal of Applied Probability*, 45:226–243, 2008.

L. Parida, I. Rigoutsos, A. Floratos, D. Platt, and Y. Gao. Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 297–308, 2000.

G. Pavesi, G. Mauri, and G. Pesole. An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics*, 17(Supplement 1):S207–S214, 2001.

G. Pavesi, P. Mereghetti, G. Mauri, and G. Pesole. Weeder Web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. *Nucleic Acids Research*, 32(Web Server Issue):W199–203, 2004.

G. Pesole, N. Prunella, S. Liuni, M. Attimonelli, and C. Saccone. WORDUP: an efficient algorithm for discovering statistically significant patterns in DNA sequences. *Nucleic Acids Research*, 20(11):2871–2875, 1992.

P. A. Pevzner and S. H. Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 269–278, 2000.

N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot. Bases of motifs for generating repeated patterns with wild cards. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):40–50, 2005.

N. Pisanti, A. Carvalho, L. Marsan, and M.-F. Sagot. RISOTTO: fast extraction of motifs with mismatches. In *Proceedings of the Seventh Latin American Symposium on Theoretical Informatics (LATIN)*, pages 757–768, 2006.

M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

S. Rahmann, T. Müller, and M. Vingron. On the power of profiles for transcription factor binding site detection. *Statistical Applications in Genetics and Molecular Biology*, 2(1):Article 7, 2003.

S. Rahmann, T. Marschall, F. Behler, and O. Kramer. Modeling evolutionary fitness for DNA motif discovery. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 225–232, 2009.

B. Raphael, L. Liu, and G. Varghese. A uniform projection method for motif discovery in DNA sequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(2):91–94, 2004.

T. E. Reddy, B. E. Shakhnovich, D. S. Roberts, S. J. Russek, and C. DeLisi. Positional clustering improves computational binding site detection and identifies novel cis-regulatory sites in mammalian GABAA receptor subunit genes. *Nucleic Acids Research*, 35(3):e20, 2007.

M. Régnier. A unifed approach to word occurrence probabilities. *Discrete Applied Mathematics*, 104:259–280, 2000.

G. Reinert and S. Schbath. Compound Poisson and Poisson process approximations for occurrences of multiple words in Markov chains. *Journal of Computational Biology*, 5(2):223–253, 1998.

G. Reinert, S. Schbath, and M. S. Waterman. Probabilistic and statistical properties of words: An overview. *Journal of Computational Biology*, 7(1-2):1–46, 2000.

E. Rocke. Using suffix trees for gapped motif discovery. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 335–349, 2000.

E. Rocke and M. Tompa. An algorithm for finding novel gapped motifs in DNA sequences. In *Proceedings of the Second Conference on Research in Computational Molecular Biology (RECOMB)*, pages 228–233, 1998.

H. Roider, A. Kanhere, T. Manke, and M. Vingron. Predicting transcription factor affinities to DNA from a biophysical model. *Bioinformatics*, 23(2):134–141, 2007.

E. Roquain and S. Schbath. Improved compound Poisson approximation for the number of occurrences of multiple words in a stationary Markov chain. *Advances in Applied Probability*, 39(1):128–140, 2007.

F. P. Roth, J. D. Hughes, P. W. Estep, and G. M. Church. Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nature Biotechnology*, 16(10):939–945, 1998.

M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. In *Proceedings of the Third Latin American Symposium on Theoretical Informatics (LATIN)*, pages 374–390, 1998.

A. Sandelin, W. Alkema, P. G. Engström, W. W. Wasserman, and B. Lenhard. JASPAR: an open access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Research*, 32(Database Issue):D91–94, 2004.

G. K. Sandve, O. Abul, V. Walseng, and F. Drabløs. Improved benchmarks for computational motif discovery. *BMC Bioinformatics*, 8:193, 2007.

S. Schbath. Compound Poisson approximation of word counts in DNA sequences. *ESAIM: Probability and Statistics*, 1:1–16, 1995.

S. J. Schultheiss, W. Busch, J. U. Lohmann, O. Kohlbacher, and G. Rätsch. KIRMES: kernel-based identification of regulatory modules in euchromatic sequences. *Bioinformatics*, 25(16):2126–2133, 2009.

M. Schulz, D. Weese, T. Rausch, A. Döring, K. Reinert, and M. Vingron. Fast and adaptive variable order markov chain construction. In *Proceedings of the Eighth International Workshop on Algorithms in Bioinformatics (WABI), Karlsruhe, Germany*, pages 306–317, 2008.

K. Shida. Hybrid Gibbs-sampling algorithm for challenging motif discovery: GibbsDST. *Genome informatics*, 17(2):3–13, 2006.

S. Sinha and M. Tompa. A statistical method for finding transcription factor binding sites. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 344–354, 2000.

S. Sinha and M. Tompa. Discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Research*, 30(24):5549–5560, 2002.

S. Sinha and M. Tompa. YMF: a program for discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Research*, 31(13):3586–3588, 2003.

R. T. Smythe. The Boyer-Moore-Horspool heuristic with Markovian input. *Random Structures and Algorithms*, 18(2):153–163, 2001.

R. Staden. Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Research*, 12(1):505–519, 1984.

R. Staden. Methods for discovering novel motifs in nucleic acid sequences. *Computer Applications in the Biosciences*, 5(4):293–298, 1989.

V. Stefanov, S. Robin, and S. Schbath. Waiting times for clumps of patterns and for structured motifs in random sequences. *Discrete Applied Mathematics*, 155(6–7): 868–880, 2007.

G. D. Stormo and G. W. Hartzell. Identifying protein-binding sites from unaligned DNA fragments. *Proceedings of the National Academy of Sciences of the United States of America*, 86(4):1183–1187, 1989.

D. M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, 33 (8):132–142, 1990.

M. M. Suzuki and A. Bird. DNA methylation landscapes: provocative insights from epigenomics. *Nature Reviews Genetics*, 9(6):465–476, 2008.

S.-H. Sze and X. Zhao. Improved pattern-driven algorithms for motif finding in DNA sequences. In *Revised Selected Papers of the Joint Annual RECOMB 2005 Satellite Workshops on Systems Biology and on Regulatory Genomics*, pages 198–211, 2006.

D. Takai and P. A. Jones. Comprehensive analysis of CpG islands in human chromosomes 21 and 22. *Proceedings of the National Academy of Sciences of the United States of America*, 99(6):3740–3745, 2002.

G. Thijs, M. Lescot, K. Marchal, S. Rombauts, B. D. Moor, P. Rouzé, and Y. Moreau. A higher-order background model improves the detection of promoter regulatory elements by Gibbs sampling. *Bioinformatics*, 17(12):1113–1122, 2001.

G. Thijs, K. Marchal, M. Lescot, S. Rombauts, B. De Moor, P. Rouzé, and Y. Moreau. A Gibbs sampling method to detect overrepresented motifs in the upstream regions of coexpressed genes. *Journal of Computational Biology*, 9(2):447–464, 2002.

M. Tompa. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 262–271, 1999.

M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, V. J. Makeev, A. A. Mironov, W. S. Noble, G. Pavesi, G. Pesole, M. Regnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23(1):137–144, 2005.

T. Tsai. Average case analysis of the Boyer-Moore algorithm. *Random Structures and Algorithms*, 28(4):481–498, 2006.

L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer, 2003.

M. Waterman, R. Arratia, and D. Galas. Pattern recognition in several sequences: Consensus and alignment. *Bulletin of Mathematical Biology*, 46:515–527, 1984.

M. S. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes*. Chapman & Hall/CRC, 1995.

E. Wingender, P. Dietze, H. Karas, and R. Knüppel. TRANSFAC: a database on transcription factors and their DNA binding sites. *Nucleic Acids Research*, 24(1): 238–241, 1996.

C. T. Workman and G. D. Stormo. ANN-Spec: a method for discovering transcription factor binding sites with improved specificity. *Pacific Symposium on Biocomputing*, pages 467–478, 2000.

E. Zaslavsky and M. Singh. A combinatorial optimization approach for diverse motif finding applications. *Algorithms for Molecular Biology*, 1:13, 2006.

J. Zhang, B. Jiang, M. Li, J. Tromp, X. Zhang, and M. Q. Zhang. Computing exact p-values for DNA motifs. *Bioinformatics*, 23(5):531–537, 2007.

M. Zhang and T. Marr. A weight array method for splicing signal analysis. *Computer Applications in the Biosciences*, 9(5):499–509, 1993.