The jPREdictor and its Application to Motif Discovery, Motif Clustering and the Prediction of Polycomb/Trithorax Response Elements

Dissertation zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
der Technischen Fakultät der Universität Bielefeld

vorgelegt von
Thomas Fiedler

April 2008

Acknowledgements

First of all, I would like to thank my supervisor Marc Rehmsmeier for his guidance and support throughout my time as a graduate student in Bielefeld. Thanks for giving me the opportunity to work in the interesting bioinformatics field of fly genetics and to do what I like most, designing, implementing, and operating a software.

Furthermore, I like to thank my office mates, Arne Hauenschild and Jia Ding, for using and exhaustively testing my *jPREdictor* program, for many discussion and debates, and for their patience with me when I bothered them with questions. I have had a very pleasant time with you two around and I thank you for that.

Many thanks go to colleagues and friends from the M3 floor, especially Sven Hartmeier, Susanne Konermann, Jan Krüger, Jens Reeder, and Alexander Sczyrba, for the comfortable breaks during work and for keeping the coffee flowing. In addition, I thank Sven Hartmeier and Jan Krüger from the BiBiServ team for their support with the internet presence of the *jPREdictor* program, and the Gaus sisters for careful proof-reading of this thesis.

Finally, I would like to thank my family for their constant support, backup, and encouragements, and Katharina for putting up with me over the past years and for bringing joy and spirit into my life.

Contents

List of Figures

List of Tables

1 Introduction

Over the past two decades, estimates on the number of protein coding genes in a genome dropped greatly. While early estimations gave numbers above $100,000$ for the human genome, today, it appears to house only 20,000-25,000 protein-coding genes [1]. The genome of the fruit fly, *Drosophila melanogaster*, underwent the same drop in gene coverage. It encodes approximately $13,600$ genes [2]. But with the decreasing number of genes in a genome, the intergenic regions became larger and larger. Once believed to be mostly junk, in fact, the space is a place for many structural elements.

The most important of these structural elements serve in controlling and regulating the transcription level of their attached genes. These so-called *cis*-regulatory elements are fragments of the DNA and can be found mostly upstream of the controlled genes. Regulatory elements are, for instance, promoter elements for switching the gene transcription on or off, respectively. Additionally, enhancer and silencer elements boost or restrain transcription to a certain degree. Another class of regulatory elements are insulators, which separate and isolate the gene-regulatory element-complexes from each other. A third and developmentally essential class of regulatory elements with direct influence on transcription are Polycomb/Trithorax response elements, PRE/TREs. Their purpose is to maintain transcriptional repression or activation, even after the transcription factors bound foremost to the promoter or enhancer element are long gone [3]. Moreover, the transcription level is maintained over many cell cycles.

All of these elements play their role in controlling and regulating the expression of genes. Identification and functional analysis of such elements is a most important task, because it leads to a greater understanding of the regulatory network of cells. Common to most if not all *cis*-regulatory DNA elements is that they are characterized by short binding sites. Through these sites, the elements are bound by transcription factors or other sequence-specific proteins. Often, multiple copies of binding sites for one factor as well as for different factors are present within one element. This has been investigated for enhancer/silencer and promoter elements [4] as well as for PRE/TREs [5]. Consequently, knowledge and identification of these binding sites is the basis for detecting *cis*-regulatory elements and thus gaining greater understanding of the gene regulation network.

In this thesis, a framework is presented in order to work with motifs and to apply them to certain tasks. The reason for working with motifs is that binding sites for one transcription factor incur sequence variability. Therefore, such binding sites are aligned into and represented as a motif and one motif is the computational representative of all binding sites for one transcription factor. Since binding sites are fairly short, motifs are too. Different representation types of motifs exist, which are presented in Chapter 2.2. Motifs can be used for searching binding sites in the genome *in silico*. This is the foundation for the prediction of regulatory elements. Due to their shortness, however, they occur in the genome frequently, and not every occurrence hints at a regulatory element. Therefore, two other features of such elements are incorporated in the predictory process. First, binding sites are enriched, and second, binding sites are clustered together in regulatory elements. These features are incorporated into the prediction by combining the motifs to double motifs as well as by weighting the motifs. Sequence fragments are scanned for the motif patterns and their corresponding weights are added up. High sum-scores hint at PRE/TREs because many motifs are tightly clustered together. In Chapter 4, an analysis of the complex prediction pipeline is presented, together with a new prediction of PRE/TREs, based on already known motifs.

Transcription factors are not limited to bind to only one gene. One such factor may regulate the transcription of many genes. High-throughput techniques such as ChIP-on-chip (Chromatin Immunoprecipitation on chips, [6]) allow biologists to monitor protein-DNA interactions *in vivo* for any given protein. In the case of a transcription factor, the discovery of interactions hints at the regulation of nearby genes by this factor. The commonly accepted assumption is that such co-regulated genes share similarities in their regulatory mechanism. This leads to the hope that the upstream regions of these co-regulated genes have transcription factor binding sites in common, which only need to be detected. Upon success, this would either result in the identification of the target binding site for a transcription factor or it would enable the prediction of other similar regulatory elements. Nevertheless, accurate identification of motifs is difficult because they are short signals in the midst of a great amount of statistical noise. Some algorithms incorporated into many computational tools were developed for the de-novo identification of motifs. In Chapter 5 an “evolutionary algorithm” is added to the number of algorithms. This novel approach makes use of the ideas behind evolution strategies in order to scan the space of possible motifs for over-representation in model sequences with respect to background sequences. The search space is not so large for basic binding sites, but it increases with the degeneration level of the motif and with the complexity of motif patterns. In particular, the evolutionary approach was developed to evolve double motifs applicable in the prediction of PRE/TREs. Discovery of such co-localized motifs and other high-complexity motifs is a novel feature, which was not addressed to this extend before.

The motif discovery based on the evolutionary algorithm results in many over-represented motifs, which are often very similar. In order to reduce the amount of similar motifs, a motif clustering was developed and is presented in Chapter 5. This agglomerative hierarchical clustering combines similar motifs to clusters and either stops in case the predefined number of clusters is reached or when a threshold similarity is exceeded. If neither restriction is given, the outcome will be one super cluster comprising all input motifs.

All three applications, the prediction of PRE/TREs, de-novo motif discovery based on evolution, and the clustering of motifs, are incorporated into a software, the *jPREdictor*. In the first place, it was developed to provide an easy and versatile way to define and represent motifs, and, moreover, to use these motifs in the prediction of PRE/TREs and other *cis*-regulatory elements. Therefore, it incorporates the functions, user interfaces, and parameters in order to define and work with a large number of motifs. Considerable efforts were put into the development of the program, and great care was taken to ensure easy maintenance and extendability. This paid off, when the motif evolution and the motif clustering were incorporated into the program. By now, the *jPREdictor* is a powerful tool for working with motifs. Simple tasks like searching are equally easy to perform as a motif discovery. Weighting as well as clustering motifs presents no computational challenge anymore. The *jPREdictor* is available on the following website:

```
http://bibiserv.techfak.uni-bielefeld.de/jpredictor
```

Organization of this thesis

In Chapter 2, the background on motifs and all three applications is presented. For the motifs, not only definition and representation issues are addressed, but also how the occurrence probability is calculated with respect to the type of the background sequence. This was done with one problem in mind: for a predicted DNA fragment with a certain score, what is the probability of reaching this or a higher score merely by chance (*p*-value calculation). The task presented considerable difficulties and was only solved for very simple motif settings. Following the background on PRE/TREs, the background on motifs, motif discovery and motif clustering is given.

In Chapter 3, the *jPREdictor* program is presented. This chapter gives an overview on how the *jPREdictor* is used and on the internal structures. At first, its history is outlined briefly, followed by a general overview of the *jPREdictor* organization. After addressing how motifs and sequences are represented in the program, the processing pipeline, user interfaces and the option file is presented.

In Chapter 4, the prediction of PRE/TREs as a class of *cis*-regulatory elements is presented. Different motifs, motif sets, weighting, and scoring schemes are reviewed with the goal of a reliable prediction with a high specificity.

Chapter 5 deals with the discovery of novel motifs via an evolutionary algorithm. This is combined with a clustering approach for fusing similar motifs into one consensus motif. After introducing both procedures, case studies are presented, where the single and the double motif dicovery are applied to artificial sequence sets. In these studies, planted binding sites have to be identified. This chapter ends by applying the motif evolution to the discovery of double motifs and the use of the resulting double motifs in a prediction of PRE/TREs.

This thesis is concluded with a discussion and some implications of this work.

Previous work

The *jPREdictor* and its application in PRE/TRE prediction have already been published in the scientific literature:

- **Thomas Fiedler** and Marc Rehmsmeier. *jPREdictor*: a versatile tool for the prediction of *cis*-regulatory elements. *Nucleic Acids Research*, 34:W546-W550, 2006.

2 Background

2.1 Polycomb/Trithorax Response Elements

Polycomb/Trithorax response elements (PRE/TREs, PREs for short) are *cis*-regulatory DNA elements. In general, *cis*-regulatory elements are fragments of the DNA that are recognized by special protein complexes, which bind to the DNA at these positions. Thereby, they influence the expression of nearby genes. Such regulatory fragments are called *cis*, because they are located on the same strand and, in addition, near or in the vicinity of their corresponding gene. The distance to the transcription start site may be a few to several thousend nucleotides. *Cis*-regulatory elements other than PREs are promoters for switching the transcription on or off for its gene or gene sequence, respectively, and enhancer and silencer elements that boost or repress transcription to a certain degree. Another class of regulatory elements are insulators, which separate and isolate the gene-regulatory element-complexes from each other. PREs are the third and developmentally essential class of regulatory elements with direct influence on transcription. Their purpose is to maintain transcriptional repression or activation, even after the transcription factors bound foremost to the promoter or enhancer element are long gone [3]. Moreover, the transcription level is maintained over many cell cycles.

Two groups of proteins with antagonistic function make use of PREs in a competitional manner. PcG (Polycomb group) proteins act as silencers, or, in other words, they maintain the previously established silenced state of a gene. On the other hand, proteins of the TrxG (Trithorax group) maintain the active state of a gene. For genes regulated by PcG/TrxG proteins the transcriptional level is preserved over many cell cycles, thus providing a transcriptional memory to and determining the fate of a cell.

PcG proteins are highly conserved throughout different species and were initially discovered in *Drosophila* ([7] and references therein). Up to date, no PREs have been identified in vertebrates, although they likely exist, since many PcG and TrxG proteins have functional homologues. Nevertheless, the following description of PRE functionality will focus on flies, especially the model organism *Drosophila melanogaster*.

The maintenance of a cellular memory involves dynamic interactions between the PcG and TrxG proteins and their target genes, via PRE elements. The complicated theories and models behind the necessary interactions are reviewed in [3]. In a first step, DNA binding proteins of both PcG and TrxG must find the PRE while the determining transcription factors are still present at the corresponding gene's promoter. This recruitment is mainly accomplished by the proteins binding to their corresponding DNA sites located on the PRE. Such sites are recognized and bound by different proteins, namely GAGA factor (reviewed in [8]), Pipsqueak, Zeste, Pleiohomeotic, and Pho-like ([3] and references therein). Recent publications reported some more proteins to play a role in recruitment [9, 10, 11]. Binding sites for members of the zinc-finger SP1/KLF protein family can be found in many confirmed PREs and are in fact bound *in vivo* [9]. The Dorsal switch protein 1 binds to the Ab-*Fab* confirmed

PRE and mutating the corresponding binding sites leads to a loss of silencing functionality [10]. Grainyhead (Grh) was reported to not only bind to the *iab*-7 confirmed PRE but also to cooperate with Pho in order to increase both their binding affinities [11]. Recent studies suggest other recruiting strategies involving non-coding RNAs [12, 13]. TrxG proteins like TRX and Ash1 are recruited to *Ubx* PREs by non-coding RNAs originating from the actively transcribed *Ubx* complex [12, 13]. Early models also hold histone modifications and chromatin binding responsible for PcG and TrxG protein recruitment, but this is challenged by findings of the core PRE sequences being devoid of histones and such modifications being long-range [14]. Nevertheless, PcG protein complexes recruited to the chromatin flanking the PRE sequence have been reported to be based on modified histones as well as sequence specificity [15]. For the PRE to accomplish other functions than mere recruiting, the binding proteins are embedded into large protein complexes. In this sense, PREs serve as "billboards" not only by bringing DNA binding proteins into proximity but also by bringing the attached proteins together.

In a second step, before the determining transcription factors disappear, the PcG and TrxG proteins have to assess the activity of the promoter in order to maintain the transcription level [3]. Little is known about this. It appears that protein recruitment is independent of the present transcription level, since some proteins of both groups, PcG as well as TrxG, are bound to any PRE. However, PREs are thought to act as silencers per default, and it seems that the TrxG proteins TRX and Ash1 counteract the repression ([3] and references therein). As a consequence, assessing the active state would be sufficient in order to circumvent repression. Such an actively transcribed gene construct produces the non-coding RNAs necessary to recruit TRX and Ash1 to the PRE [12, 13]. Nevertheless, this easy mechanism is challenged by recent findings and models proposing a role for non-coding RNAs not only in activation but also in repression through transcriptional interference [13, 16].

The third step is that the proteins recruited to a PRE install their own system that reproduces the assessed transcriptional level. For this, the chromatin is remodeled and several epigenetic marks are set ([3] and references therein). Proteins of the PcG group methylate histone 3 at several lysines, namely H3K9 and H3K27. In addition, methylation is performed at H4K20 ([17] and references therein). On the other hand, TrxG proteins, especially the histone methyltransferases TRX and Ash1, methylate histone 3 at lysine 4 and 9 (H3K4, H3K9) and histone 4 at lysine 20 (H4K20). Other TrxG proteins acetylate histone 3. Acting contrary, proteins of the PcG group possess deacetylation functionality. In this light, the difference between repression and activation seems to be the presence or absence of methylated H3K27, respectively, and the absence or presence of acetylated H3, respectively. Nevertheless, the influence of H3K27 methylation is questionable because this modification is typically present across the whole coding region [17, 18]. In addition, acetylation seems to play no part at all because a presented working model [17] ignores this modification event entirely. In this model, which aims at explaining the maintenance of the *Ubx* gene's activity [17], the OFF state is characterized by the presence of trimethylated H3K27, H3K9, and H4K20 across the whole *Ubx* gene, while in the ON state, trimethylated H3K27, H3K9, and H4K20 is only present throughout the upstream control region, but is largely absent in the promoter and coding region and H4K20me1 and H3K4me3 are present instead. All in all, as Ringrose and Paro [3] pointed out, the chromatin is in a fluid state and the TrxG proteins are constantly battling for keeping the active state up.

The last step in PRE functionality is that after each cell division, this same transcriptional state has to be correctly reinstalled. Ringrose and Paro [3] discuss some potential mechanisms, which give a head start in the possibly correct direction. The most attractive candidate is histone lysine methylation, since it seems to be a very stable modification. Modified histones are distributed equally between daughter strands after replication and may serve as a local mark for the history of the piece of DNA that bears them. Recent analyses comfirm the subsequent formation of heterochromatin in order to install a silenced state on the basis of lysine-modified histones [19, 20]. This could be the missing link to remembering the transcriptional state over many cell cycles.

Concluding from the above, the task of orchestrating cellular memory of many different genes in hundreds of different cell types is quite formidable and may involve many variations on this basic four-step theme [3]. To increase the difficulty even further, the compositions and properties of PcG and TrxG complexes are different at different target loci and, furthermore, expression profiles seem to be tissue-specific and dependent on the current developmental stage.

Nevertheless, PcG and TrxG protein complexes have many target genes in flies, among them many segmentation and developmental genes ([3] and references therein). In this light, discussing potential new roles for PcG proteins is the next logical step (reviewed in [7]). It is a challenging task to identify all targets and to compare their activation level in different cell types and across different genomes. Recent protein binding studies (reviewed in [21]) indicate that up to 5% of all genes in flies are PcG targets. Of further interest are the binding studies of Tolhuis et al. [22] and Negre et al. [23], as this thesis uses the sequences they provided. Both groups published *in vitro* scans for Polycomb binding domains in the *D. melanogaster* genome. Tolhuis et al. [22] provided 131 sequences with an overall length of 3.7 Mb, Negre et al. [23] provided 41 sequences with overall length around 195 kb. A conclusion from the review [21] is that although these studies provide us with an extensive list of PcG targets, they do not add to the number of TrxG targets. In addition, it becomes clear that cell differentiation is accompanied by shifts in the PcG target gene activity, which may explain the missing overlap in the reviewed studies. Therefore, in order to understand the dynamic PcG regulation, cell fate transitions in intact organisms have to be studied.

In this thesis, a sensitive approach for the computational prediction of PRE/TREs is presented. Locating these regulatory elements in the genome is the first step in identifying target genes and it also gives the starting point for further studies.

2.2 Motifs and motif probabilities on sequences

In genetics, motifs are short patterns, which are widespread and have a biological significance. Motifs in this work are always built from DNA/RNA sequences. They may be $5-15$ nucleotides long, rarely shorter, sometimes longer. Due to their shortness they occur very often in the genome and act as profiles to be recognized by binding proteins. In genomic regions with regulatory function, however, binding sites may be accumulated, since said regions rely on proteins to bind there preferably.

Motifs can be searched on sequences by matching them to sequence sections. A sequence S is defined as a number of consecutive letters $s_1 s_2 \dots s_{|S|}$, where $|S|$ denotes the length of the sequence and every letter is taken from the finite alphabet Σ. For DNA sequences the alphabet may be $\Sigma_{DNA} = \{A, C, G, T\}$, for RNA sequences the alphabet $\Sigma_{RNA} = (\Sigma_{DNA} \backslash \{T\}) \cup \{U\}$ is used. A section of the sequence is referred to as $S_{[i,j]}$ with s_i as first and s_j as last letter ($i \leq j$). For every letter of Σ the frequency $fr_{\sigma \in \Sigma}$ gives the probability of finding the letter σ in S. The sequence may be generated under the model of a zeroth order Markov chain (denoted MC0, [24]), which is a Markov chain with no memory and non-uniform distribution for the letters of the alphabet. For example, the genome of *Drosophila melanogaster* has the non-uniform distribution $p(A) = fr_A = 0.2877$, $fr_C = 0.2124$, $fr_G = 0.2124$, $fr_T = 0.2877$. This means if a letter is picked randomly from the genome it is an A in 29% of all cases.

For every motif type an equation is given to calculate the probability to find this motif merely by chance. Different equations are necessary to address different background models. The probabilities are given and used to solve the task of how likely it is to encounter a certain sum-score for a sequence fragment. Sum-scores are calculated from weighted motifs, after they were found on a sequence. This task presented some difficulties and was only solved for equal-weighting simple motifs. No solution can be given for arbitrary-weighting motifs as well as motif patterns.

2.2.1 Sequence motifs

Sequence motifs are motifs which consist of only one well-defined short sequence. They are composed of the same letters used in genomic sequences. Thus $M \in \Sigma^k$, where M denotes the motif and k its length. An example for this is *TATAAA*, the core motif of the TATA-box (also called Goldberg-Hogness box according to its discoverers [25]). In *Drosophila melanogaster* genes the *TATAAA* motif, or the motif with one mismatch allowed, is enriched in promoter regions. This shows its importance for gene regulation. It is present in 43% of 205 core promoters [26] or, according to another study, in 33% of 1941 potential promoters [27]. Sequence motifs can be defined as $M = m_1 m_2 \dots m_k$ with $m_i \in \Sigma$ for all $1 \leq i \leq k$. In the *TATAAA* example m_1 and m_3 map to a T, all other single letters are A's.

A match of a sequence motif on a genomic sequence is defined as index i where the constraints $s_i = m_1, s_{i+1} = m_2, \dots, s_{i+k-1} = m_k$ hold, thus i is the starting position and $S_{[i,i+k-1]} = M$ (the cutout from S is exactly M). If mismatches are allowed, the required number of constraints to hold is reduced by the number of mismatches allowed. However, if mismatches occur $S_{[i,i+k-1]} = M$ is not valid anymore. In this case, $S_{[i,i+k-1]} \simeq M$ is used to state a match. Obviously, when a sequence motif has to be searched on a genomic sequence the probability to find it relies on the motif itself. Calculating that probability under the terms of different sequence types has often been investigated [28, 29, 30]. For sequences generated under a zeroth order Markov model (MC0) and matches without mismatches allowed it can be given as

$$p(S_{[i,i+k-1]} = M) = \prod_{i=1}^{k} fr_{m_i} . \tag{2.1}$$

If all letters in Σ are equally distributed, above equation simplifies to $p(S_{[i,i+k-1]} = M) = \frac{1}{|\Sigma|^k} = \frac{1}{4^k}$. Note that Equation 2.1 holds under the assumption that all positions in both sequence and motif are independent. For generated sequences this might be correct, but it is not obvious that this assumption is fulfilled for arbitrary motifs. If, for instance, the motif *AAAA* is found on a DNA sequence (generated with uniformly distributed letters) starting at s_i, the probability of a match at position s_{i+1} is rather $\frac{1}{4}$ and not $\frac{1}{256}$, because three of four letters already match. On the other hand, in case the motif *ACGT* was found at s_i the probability of a match starting at position s_{i+1}, s_{i+2}, s_{i+3} is zero. Thus, one would assume motifs which overlap themselves to occur more often than motifs that are not periodic. But the paradox regarding the waiting time before the occurrence of the next motif acts contrary [31, 32]. In the mean, the next motif *ACGT* occurs after $4^4 = 256$ characters. In contrast,

Table 2.1: Occurrences and waiting times for three different motifs. The random DNA sequences are either generated with uniform nucleotide distribution, or with the base distribution of the *D. melanogaster* genome. Both are of length 3×10^9.

Base distribution	Motif	Occurrence probability	Found occurrences	Variation	Mean waiting time overlap	Mean waiting time no overlap
uniform	*AAAAAA*	$4,096^{-1}$	$731,411$	$-1,011$	$4,100$	$5,463$
uniform	*ACGTAC*	$4,096^{-1}$	$732,690$	-468	$4,098$	$4,114$
uniform	*YGAGNCTCY*	$16,384^{-1}$	$182,250$	855	$16,367$	$16,367$
D.mel.	*AAAAAA*	$\approx 1,775^{-1}$	$1,694,144$	$3,540$	$1,763$	$2,474$
D.mel.	*ACGTAC*	$\approx 4,384^{-1}$	$684,238$	101	$4,382$	$4,399$
D.mel.	*YGAGNCTCY*	$\approx 23,705^{-1}$	$127,023$	-665	$23,793$	$23,795$

the waiting time for the next motif *AAAA* gives 340 characters due to its overlapping. Thus, periodic motifs occur in clusters, e.g. the *GAGA-factor* binding site *GAGAG*. Since both effects rely on periodicity they neutralize themselves. Therefore, Equation 2.1 holds for arbitrary motifs, which can be seen experimentally proven in Table 2.1.

Equation 2.1 is easily adapted to other sequence-generating models, e.g. Markov chains of higher order than zero. For this, probabilities for tuples, triples and quadruples of nucleotides must be known. For instance, generating the background as a second order Markov chain changes Equation 2.1 to

$$p(S_{[i,i+k-1]} = M) = fr_{m_1} \cdot fr_{m_1 m_2} \cdot \prod_{i=3}^{k} fr_{m_{i-2} m_{i-1} m_i} \,. \tag{2.2}$$

If mismatches are allowed to occur in a match, above equations are both insufficient. Intuitively, given e the number of mismatches allowed, the equation $p(S_{[i,i+k-1]} \simeq M, e) = \frac{1}{|\Sigma|^{k-e}} = \frac{1}{4^{k-e}}$ seems correct at a first glance. With this equation, the probability of finding the motif *ACGTAC* in a sequence with uniformly distributed nucleotides and one mismatch allowed would be $\approx 1 \cdot 10^{-3}$. This does not apply, as can be seen in Table 2.2, because the real probability to find motif *ACGTAC* with one mismatch allowed is almost five times higher. Therefore, the following equation for calculating the probability of a sequence motif with up to e mismatches allowed is proposed:

$$p(S_{[i,i+k-1]} \simeq M, e) = \frac{1}{4^k} \cdot \sum_{i=0}^{e} \binom{k}{i} \cdot 3^i \,. \tag{2.3}$$

Equation 2.3 is valid under a uniform nucleotide distribution. The term $\binom{k}{i} \cdot 3^i$ gives the number of possible sequences, if exactly i nucleotides out of k are exchanged by one of the other 3 bases from the alphabet. This equation was practically proven (Table 2.2) and was actually derived from the much more complicated one, which assumes a zeroth order Markov chain as background model:

$$\begin{aligned} p(S_{[i,i+k-1]} \simeq M, e) = & \qquad\qquad (2.4) \\ \sum_{e_0=0}^{e} \left[\underbrace{\sum_{e_1=1}^{k-e_0+1} \sum_{e_2=e_1+1}^{k-e_0+2} \cdots \sum_{e_{e_0}=e_{e_0-1}+1}^{k}}_{e_0\, sums} \underbrace{\frac{(1-fr_{m_{e_1}})}{fr_{m_{e_1}}} \cdot \frac{(1-fr_{m_{e_2}})}{fr_{m_{e_2}}} \cdot \ldots}_{e_0\, times} \cdot \prod_{i=1}^{k} fr_{m_i} \right] . \end{aligned}$$

Table 2.2: Occurrences of the motif *ACGTAC* with different number of nucleotide mismatches allowed for a match. The random DNA sequences are either generated with uniform nucleotide distribution, or with the base distribution of the *D. melanogaster* genome. Both are of length 3×10^9.

Base distribution	Max Errors	Found occurrences	Probability
uniform	0	$731,846$	$2.4 \cdot 10^{-4}$
uniform	1	$13,918,325$	$4.6 \cdot 10^{-3}$
uniform	2	$112,794,210$	$3.8 \cdot 10^{-2}$
D.mel.	0	$683,957$	$2.3 \cdot 10^{-4}$
D.mel.	1	$13,376,330$	$4.5 \cdot 10^{-3}$
D.mel.	2	$110,716,131$	$3.7 \cdot 10^{-2}$

The bracket-enclosed sum terms in Equation 2.4 enumerate over the $\binom{k}{e_0}$ sequences that are possible by exchanging e_0 nucleotides in M with other nucleotides from Σ. Equation 2.4 was practically proven as well (see Table 2.2). For higher order Markov models no formula is given here, because these models do not work under the assumption of independent positions. In this case, an enumeration approach as discussed for matrices (next chapter) is more appropriate.

The sequence motifs described above use the 4-letter alphabet $\Sigma = \{A, C, G, T\}$. But, in cases where two or more bases are permitted at a particular position, an enhanced alphabet, the IUPAC one letter code [33, pp. 123–126], may be used. The one letter code maps single-letter symbols to a set of nucleotides. This nomenclature may also be applied where uncertainty exists as to extent and/or identity. This enhanced alphabet comprises the 4-letter alphabet and adds 11 distinct letters, one for every combination of two to four nucleotides. Possible combinations of two nucleotides are named $R = \{A, G\}$ (pu*R*ine), $Y = \{C, T\}$ (p*Y*rimidine), $M = \{A, C\}$ (a*M*ino), $K = \{G, T\}$ (*K*eto), $S = \{C, G\}$ (*S*trong interaction, 3 H bonds) and $W = \{A, T\}$ (*W*eak interaction, 2 H bonds). Three nucleotides are comprised into the letters $B = \{C, G, T\}$ (not A), $D = \{A, G, T\}$ (not C), $H = \{A, C, T\}$ (not G) and $V = \{A, C, G\}$ (neither T nor U). All four nucleotides are denoted by the letter N (a*N*y). The letter N is most often used to specify fixed-sized gaps occurring within motifs.

Motifs built from the IUPAC one letter alphabet are called regular expression motif. Because this IUPAC alphabet comprises the 4-letter alphabet, all sequence motifs are a subtype of the regular expression motifs. An example is the DNA binding site of the protein *Zeste*, *YGAGYG*. For calculating the occurrence probability every letter from the IUPAC alphabet may be replaced by its mapped characters set (similar to motif class 9 in [28]), for *Zeste* it would be $[ct]GAG[ct]G$. The Y character was replaced by C, T. The motifs characters are now indexed as $m_{i,j}$, where i is the position within the motif and j is the position in the set. The length of the set at a position is referred to as l_i. Clearly, the probability for a position to match a letter of the sequence is the sum over the permitted nucleotides' frequencies, thus Equation 2.1 is extended to be used with sets of characters:

$$p(S_{[i,i+k-1]} \simeq M) = \prod_{i=1}^{k} \sum_{j=1}^{l_i} fr_{m_{i,j}} . \tag{2.5}$$

Regular expression motifs bridge the gap between sequence motifs and more complicated ones like matrix motifs (see next chapter). And even though the IUPAC code might need some getting used to, it is easier to read and its letters are easier to recognize than matrix motifs. Matching them against each other and against genomic sequences can be done by hand, if a first glimpse is needed. And since they can be represented as strings, simpler programs and algorithms like string/pattern matching can be applied to them, making them easier to use. Nevertheless, regular expression motifs should only be used in case the uncertainty in one position is not biased toward a specific nucleotide. If one nucleotide at a certain position is preferred over another one, but both are valid, a nucleotide distribution vector shall be defined for that position. With this in mind, a regular expression motif can easily be converted into a matrix motif, namely a position specific probability matrix. Or, more simple, single IUPAC letters have their counterpart as a probability vector. However, it is not recommended to mix up such letters and vectors to form a motif, since the search issue is different. While IUPAC letters match nucleotides on an all-or-nothing basis, vectors can match nucleotides partially and even as a fraction. Rudimentary, this can be seen in Equation 2.5, where the fr vector (background) is used for every position of the regular expression motif.

2.2.2 Position specific matrices

This motif type is represented by a matrix $M = (m_{i,j})$ with $i \in \{1, \ldots, k\}$ and $j = \{1, \ldots, |\Sigma|\}$, where k is the length of the motif and Σ is the alphabet. Thus, every position within the motif is specified by a vector, namely m_i, with numbers mapped to the letters of the alphabet. The numbers can be interpreted either as occurrence counts, occurrence probabilities or weights. Typically, matrices found in TRANSFAC consist of occurrence counts, while for instance Down et al. [34] give 30 position weight matrices (PWMs) derived from FlyReg 2.0 [35] as occurrence probabilities. This latter example shows that occurrence counts and probabilities can be handled as weights, on the other hand, the opposite is not valid since weights may be negative. Additionally, careful differentiation between weights and counts/probabilities has to be made with regards to searching and matching. This topic will be covered below.

In this work, a motif represented by a matrix filled with occurrence counts/probabilities is denoted as position specific probability matrix (PSPM), m^{PSPM}. Similarly, if the matrix values are considered to be weights or scores the motif is denoted as position specific score matrix (PSSM), m^{PSSM}.

Position specific matrices are defined for binding factors which are rather lax in recognizing and binding DNA sequences. Typically, few well-defined nucleotides make up the core sequence, often trailed left and right by degenerated parts. In order to analyze which sites are bound by a transcription factor, either DNase I footprinting [36] or *in vitro* binding site selection experiments [37] can be used. Both yield a bunch of binding sites, and it depends on the resolution whether the resulting sequences are pleasantly short or too long, either leading to an effective alignment or making motif discovery algorithms like Gibbs sampling approaches [38, 39, 40] necessary. The Pho transcription factor was analyzed in depth, and different groups have published slightly mismatching sites bound by the factor [41, 42, 43, 44, 45]. In Table 2.3 these sites are listed, the core sequence *CCAT* is marked in blue. It has to be noted that most often in literature (e.g. [5]) and in this work, too, the core sequence for Pho is given as *GCCAT*, since only one binding site was reported to differ. Based on that 5-base core, the Pho motif has four preceding and five consecutive degenerated positions, thus, its length is 14 nucleotides. In order to build a matrix from all 23 binding sites the number of nucleotides has to be counted in every position. Dividing the resulting occurrence counts by the number of binding sites gives occurrence probabilities (Table 2.4). Positions are named after Mihaly et al. [42]. For better

Table 2.3: Known and confirmed binding sites for the Pho transcription factor aligned after the minimal core *CCAT*.

No	Binding site sequence	Source in *D. mel.* genome	Literature
1	GGCAG CCAT TTTCC	*en* gene, -550..-531	[43, 45]
2	CGCAG CCAT TTTCC	*D. vir. en* gene, -725..-712	[41]
3	GTCGG CCAT TAAAA	PBX region	[44]
4	GGAAG CCAT AACGG	PRE_D binding site 1	[44, 45]
5	CGCAG CCAT TATGG	PRE_D binding site 2	[44, 45]
6	GTTAG CCAT CTCGC	PRE_D binding site 3	[44, 45]
7	CGTCG CCAT AACTG	PRE_D binding site 4	[44, 45]
8	AACGA CCAT TACGA	PRE_D binding site 5	[44, 45]
9	TGAGG CCAT CTCAG	PRE_D binding site 6	[44, 45]
10	GGCAG CCAT GTTGG	iab-2 (a)	[42]
11	GGCGG CCAT TGCGG	iab-2 (b)	[42]
12	GGCAG CCAT CAATG	Mcp	[42]
13	GTCGG CCAT CTTGG	iab-6	[42]
14	CTCGG CCAT CATGG	iab-7 (a)	[42]
15	GGCAG CCAT CATGG	iab-7 (b)	[42]
16	CTCTG CCAT CAGAG	iab-8	[42]
17	GTCAG CCAT TTTGG	Scr 10 Xba (a)	[42]
18	TTCAG CCAT TATTG	Scr 10 Xba (b)	[42]
19	CTCCG CCAT CTGCG	Scr 10 Xba (c)	[42]
20	ATCCG CCAT GGTAG	Scr 7.6 H III	[42]
21	TACTG CCAT TACAG	Scr 7.6 H III/6.5 KS	[42]
23	TGCCG CCAT ATTAT	ph 418	[42]

Table 2.4: Position specific matrix for the binding site of the Pho transcription factor, with occurrence counts and occurrence probabilities derived from 23 known binding sites. The 5-base core sequence *GCCAT* is shaded in blue.

Position	*A*		*C*		*G*		*T*	
-4	3	(0.1304)	6	(0.2609)	10	(0.4348)	4	(0.1739)
-3	2	(0.087)	0	(0)	12	(0.5217)	9	(0.3913)
-2	2	(0.087)	19	(0.8261)	0	(0)	2	(0.087)
-1	11	(0.4783)	4	(0.1739)	6	(0.2609)	2	(0.087)
1	1	(0.0435)	0	(0)	22	(0.9565)	0	(0)
2	0	(0)	23	(1)	0	(0)	0	(0)
3	0	(0)	23	(1)	0	(0)	0	(0)
4	23	(1)	0	(0)	0	(0)	0	(0)
5	0	(0)	0	(0)	0	(0)	23	(1)
6	3	(0.1304)	8	(0.3478)	2	(0.087)	10	(0.4348)
7	11	(0.4783)	0	(0)	3	(0.1304)	9	(0.3913)
8	2	(0.087)	7	(0.3043)	2	(0.087)	12	(0.5217)
9	6	(0.2609)	3	(0.1304)	10	(0.4348)	4	(0.1739)
10	2	(0.087)	3	(0.1304)	17	(0.7391)	1	(0.0435)

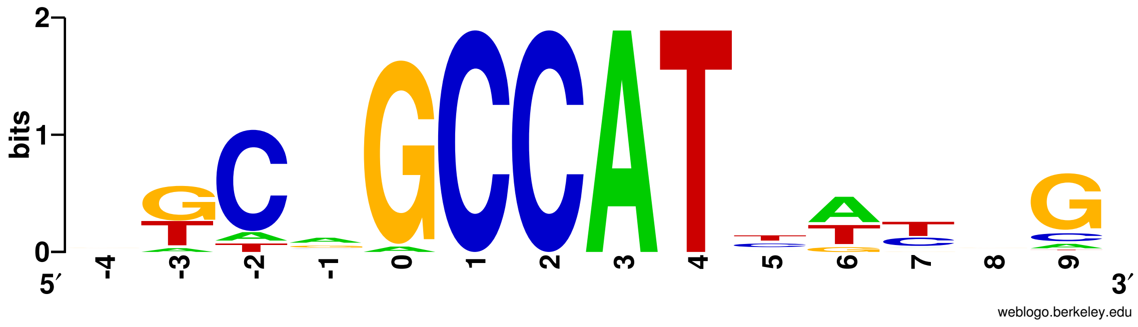


Figure 2.1: Sequence logo for the Pho transcription factor, generated from 23 confirmed binding sites. The core sequence *GCCAT* can easily be spotted at position 0 to 4.

readability a sequence logo [46] was generated. In every position the height of each nucleotide is made proportional to its occurrence probability, whereas the height of the entire stack is adjusted to the information content. For the Pho sequence logo, which was generated with *WebLogo* [47], the 5-letter core is easily spotted (see Figure 2.1).

In a last step occurrence probabilities are recalculated to weights, which are defined as log-odd-scores. For this purpose, an overall nucleotide distribution acting as background is necessary, here denoted as fr. The formula for every position in this new weight matrix is

$$m_{i,j}^{PSSM} = \ln\left(\frac{m_{i,j}^{PSPM}}{fr_j} + c\right), \tag{2.6}$$

where i denotes the position within the motif, j the corresponding nucleotide in the positional vector. The value $c > 0$ denotes a small pseudo-count value, which is added to the fraction in order to avoid infinite values. Applying Equation 2.6 to the Pho occurrence probabilities (Table 2.4) with $c = 0.01$ yields the weights in Table 2.5. Two different background nucleotide distributions were used, the genome-wide one from *D. melanogaster*, $fr_A = fr_T = 0.2877$, $fr_C = fr_G = 0.2123$, and a uniform distribution, where every nucleotide has a probability of one fourth to occur. This results in different weights depending on the used background distribution. In the *D. melanogaster* genome, the nucleotides A and T occur more often than nucleotides C and G. Henceforth, even if two positions have the same probability, they might have different weights. The nucleotide, whose background is less frequent, is assigned a higher weight. This reflects its significance to the whole motif.

Recalculating probabilities to weights might seem inappropriate and unnecessary, but working with weight matrices has some advantages over probability matrices. First of all, interpretation and readability is much more straight forward, since weights are not limited to $[0,1]$. Second, since computing probabilities means multiplying them, the resulting product uses only the exponent of floating point numbers, which can easily overflow, especially, when working with “floats” (8 bit exponent). On the contrary, scores are summed up.

Interpretation of weights is always done in relation to the background distribution. Finding a nucleotide in one position with a probability equal to the background probability yields a weight of slightly above zero. Since c is added to all fractions, weights are shifted toward higher positive values. Nucleotides which may not occur in one position have a zero or very low probability (but always

Table 2.5: Position specific score matrix representing the binding sites of the Pho transcription factor, with log-odd-scores (weights) derived from recalculating nucleotide occurrence probabilities. Two background nucleotide distribution were used: the one found in the *D. melanogaster* genome (left), and a uniform one (right, in parenthesis). The 5-base core sequence *GCCAT* is shaded in blue.

Position	*A*		*C*		*G*		*T*	
-4	-0.77	(-0.63)	0.21	(0.05)	0.72	(0.56)	-0.49	(-0.35)
-3	-1.16	(-1.03)	-4.61	(-4.61)	0.90	(0.74)	0.32	(0.45)
-2	-1.16	(-1.03)	1.36	(1.2)	-4.61	(-4.61)	-1.16	(-1.03)
-1	0.51	(0.65)	-0.19	(-0.35)	0.21	(0.05)	-1.16	(-1.03)
1	-1.83	(-1.69)	-4.61	(-4.61)	1.51	(1.34)	-4.61	(-4.61)
2	-4.61	(-4.61)	1.55	(1.39)	-4.61	(-4.61)	-4.61	(-4.61)
3	-4.61	(-4.61)	1.55	(1.39)	-4.61	(-4.61)	-4.61	(-4.61)
4	1.25	(1.39)	-4.61	(-4.61)	-4.61	(-4.61)	-4.61	(-4.61)
5	-4.61	(-4.61)	-4.61	(-4.61)	-4.61	(-4.61)	1.25	(1.39)
6	-0.77	(-0.63)	0.50	(0.34)	-0.87	(-1.03)	0.42	(0.56)
7	0.51	(0.65)	-4.61	(-4.61)	-0.47	(-0.63)	0.32	(0.45)
8	-1.16	(-1.03)	0.37	(0.2)	-0.87	(-1.03)	0.60	(0.74)
9	-0.09	(0.05)	-0.47	(-0.63)	0.72	(0.56)	-0.49	(-0.35)
10	-1.16	(-1.03)	-0.47	(-0.63)	1.25	(1.09)	-1.83	(-1.69)

smaller than the background), and thus have weights smaller than zero. They will be avoided at that specific position and are also called underrepresented. The opposite are nucleotides with high occurrence probability. They are called over-represented and yield weights greater zero. An example for nucleotide over-representation are the weights for the core sequence of Pho, they are all above one.

Searching and matching with matrices is not as straight forward as with regular expression motifs. A matrix motif is searched on a sequence by summing up the positional weights for the letters of the sequence. In case the values are handled as probabilities they are multiplied. The process of searching yields a sum-score or a product for every starting position on the sequence (of length l), respectively, thus giving a total of $l-k+1$ results. Note that the sequence has to be at least as long as the matrix for matching. This sum-score is named the motif match score by Bailey and Gribskow and they show in a picture how to calculate it for one position [48]. Figure 2.2 gives the sum-scores of Pho calculated for two sequences at every position. For both random and non-random sequence the sum-scores behave the same: they accumulate above the minimum sum-score, and are rarely close to the maximum. Additionally, from one position to the next, sum-scores may change rapidly and it is impossible to derive one sum-score from a previous one. Admittedly, their might be occasions where this derivation is possible, e.g., when a matrix is searched on a sequence consisting of only the same letter, but, generally spoken, it is not.

The time consumption for searching with matrices is increased in comparison to searching with sequence motifs (strings). Optimal string matching has a time complexity (after processing the search string) of $O(n)$, with n being the length of the sequence the motif is searched on. Searching with matrices has a time complexity of $O(k \cdot n)$. Some tricks, e.g. look-ahead scoring [49], were developed for speeding up searching, but they function only in conjunction with encountering a match. A match of the matrix motif on a sequence is defined as the starting position, where the corresponding sum-

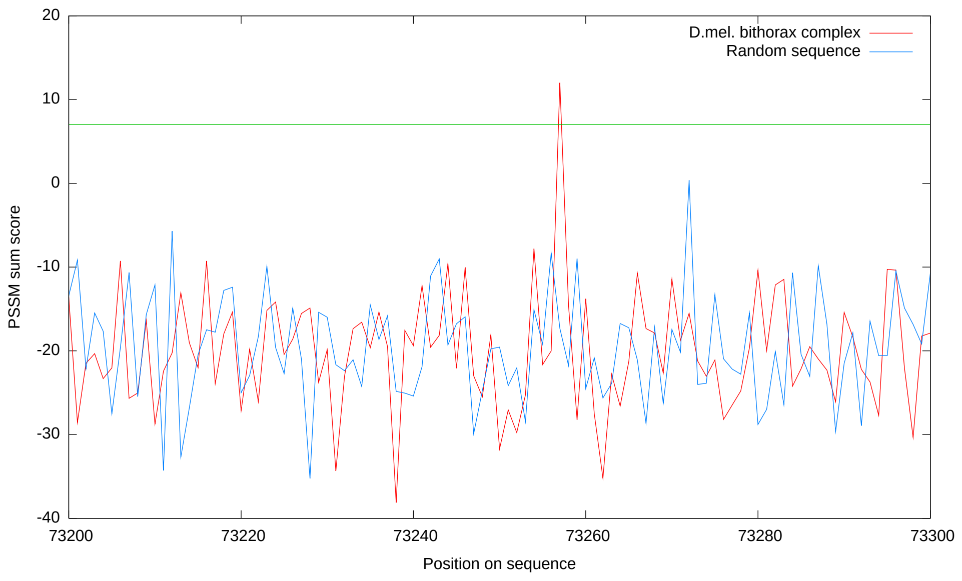


Figure 2.2: Matching the Pho position specific score matrix to 100 positions of two sequences, red, the *D. melanogaster bithorax complex*, and blue, a randomly generated sequence. For every starting position a sum-score is yielded. The green line gives the threshold. High sum-score exceeding this threshold are considered to be hits.

score exceeds a given threshold:

$$S_{[i,i+k-1]} \simeq M \Rightarrow \sum_{j=1}^{k} m_{j,S_{i+j-1}} \geq threshold\,. \tag{2.7}$$

One big problem in working with matrices is to define an appropriate threshold. On the one hand, it must be low enough to find real and potential binding sites, on the other hand, it must be high enough to not report too many false positives. When searching for an appropriate threshold some guidelines can be applied:

1. An intuitive approach takes the maximum possible sum-score, and sets the threshold for occurrence at some percentage of that maximum score.

2. If the binding sites used to generate the matrix motif are known, choose the highest threshold such that all sites match.

3. If a complete binding profile for the matrix motif is available, including additional sites which are tested to not be bound by the binding factor, choose the minimal threshold such that none of the avoided sites match. Also apply guideline two and take the higher threshold.

4. Decide on a frequency (E-value) or a probability (p-value) how often the matrix motif is to be found in a randomly generated sequence and calculate the threshold in accordance to this value.

The first guideline was shown to have merit by Tronche et al. [50], who pointed out that Hepatocyte nuclear factor 1 (HNF1) binding sites with a sum-score greater than 83% of the maximum possible sum-score of the corresponding matrix showed experimental evidence of binding. Nevertheless, this idea is not applied here, since it is expected that different binding factors will have different percentages of maximum scores allowed for binding. The matrix for the Pho motif yields a maximal possible sum-score of approximately 14.2, for which the matrix matches exactly one site. The minimal sum-score is -43.2, for which the motif would match every position in a sequence. Applying the 83% to 14.2 gives a threshold of 11.8. This seems very high, only around 300 different sites (out of $4^{14} \approx 2.7 \cdot 10^8$ possible ones) can reach this sum-score.

In order to follow the second guideline, binding sites used to build up the matrix motif are applied. Nevertheless, experience has shown that the threshold defined this way is often to low, finding too many matches in unknown sequences. To obtain the threshold by applying the third guideline, a complete binding profile for the factor is needed, with sites known to be bound and other short sequences known to be avoided. With binding site selection experiments [37] sequences can be tested for a weak or not-at-all binding. It works by slightly altering known binding sites and checking whether the factor binds or not. Unfortunately, very few binding factors were analyzed in such depths, since this is costly and time-consuming. Fortunately, for Pho it was done. Besides the 23 sites the Pho motif was found to bind (Table 2.3), 10 sites were reported, where Pho does explicitly not bind: *GGCAG CACG TTTCC* [43], a mutated engrailed binding site, *TTTTG CCAT GGCTA*, *ATGCG CCAT AAAAA*, *AAGGT CCAT AATCT*, *TTGCA ACAT CTATA*, *CTTTG ACAT TTGCC*, *ATGAG CCAT AAAAC*, and *CGTCG CACG AACTG* [44], *CTTTC TAGA CATGG*, and *GGAGA TATT CATGG* [51]. Searching the Pho motif on these sites results in a minimal threshold of 2.0, for which all of them are not matched. The maximum threshold such that all 23 binding sites match is 4.2. Since 4.2 is higher than 2.0 the threshold of choice would be 4.2. Comparison with the maximum threshold of 14.2 gives the impression that a threshold of 4.2 is rather low. Therefore, the last guideline is applied here, too.

To be able to follow the fourth guideline, a way to transform thresholds to p-values and vice versa has to be found. If a threshold is given, the corresponding p-value is the probability on how likely it is to have a match on a random sequence generated under a certain model, i.e. to have a sum-score equal or even exceeding the threshold. The naive way to obtain the corresponding occurrence probability is to enumerate all possible sequences with length k and test for them to be equal or above the threshold:

$$p_{threshold}(M\,matches) = \frac{|\{S \in \Sigma^k \mid \sum_{i=1}^{k} m_{i,s_i} > threshold\}|}{|\{S \in \Sigma^k \mid \sum_{i=1}^{k} m_{i,s_i}\ exists\}|}\,, \tag{2.8}$$

where $S = (s_1, \ldots, s_k) \in \Sigma^k$ are permutations over the alphabet. If all sum-scores exist, i.e. no weight is infinity or undefined, the denominator resolves to $|\Sigma|^k$. Since every enumerated sequence S has the same probability to occur, Equation 2.8 gives the correct probability only in case of random sequences generated under a uniform nucleotide distribution model. For sequences generated as a zeroth order Markov chain with a non-uniform letter distribution, Equation 2.8 has to be changed to

$$p_{threshold}(M\,matches) = \sum_{S \in \Sigma^k} \frac{1\{\sum_{i=1}^{k} m_{i,s_i} > threshold\}}{\prod_{i=1}^{k} fr_{s_i}}\,. \tag{2.9}$$

Again, all possible sequences are enumerated, but now the indicator function in the enumerator is normalized by the probability of each permutated sequence. The denominator can easily be adapted to other sequence-generating models, e.g. Markov chains of higher order than zero, if probabilities for tuples, triples and quadruples of nucleotides are known. For instance, for sequences generated as a first order Markov chain, the denominator changes to $fr_{s_1} \cdot \prod_{i=2}^{k} fr_{s_{i-1}s_i}$.

Both equations, 2.8 and 2.9, have a time complexity of $O([\Sigma]^k)$ and a space complexity of $O(k)$. This exponential time complexity is far too slow, because limits are already reached using the DNA alphabet and a motif of length 16. In this case roughly four billion samples have to be generated and scored. For getting only an estimate of the probability, one approach could be to generate a limited number of samples and apply Equation 2.8 or 2.9, respectively. However, as Barash et al. [52] pointed out, the main drawback of this method is the huge number of samples needed for the exact estimation of the probability, due to the sparseness of high-scoring samples, e.g. if estimating a probability in the order of 10^{-3} the number of samples must be around 10^5. Nevertheless, this randomly generating and scoring samples is the same as randomly drawing from an unknown *p*-value distribution. The compound importance sampling approach from Barash et al. [52] also draws a limited amount of weighted samples from a known *p*-value distribution and estimates the unknown *p*-value distribution. This approach compares well to the full enumeration approach even for high thresholds (and therefore low *p*-values), but it gets worse with the threshold approaching zero. This again reflects the sparseness of high-scoring samples. Another approach would be to use generating functions [28, 53] and similar methods based on the same idea [54, 48]. Again, these approaches yield a *p*-value distribution rather than a single *p*-value. Nevertheless, they require the weights from the matrix motif to be of limited precision, e.g. to be integer numbers. This way, sum-scores for partial sequences will result in the same value, maybe reducing the number of distinct partial results greatly, and tabulation can be applied to store the number of sum-score occurrences. However, if dealing with arbitrary numbers or if a high precision is wished, generating functions worst-case time complexity stays $O(|\Sigma|^k)$. And worse, since partial results need to be tabulated, space complexity becomes $O(|\Sigma|^k)$ as well.

Figure 2.3 shows the *p*-value distribution for the Pho motif. It was created with the *jPREdictor* using parameter "—PSSMprobs" and default settings (see Chapter 3.8). The above mentioned threshold of 2.0 corresponds to a *p*-value of $1.4 \cdot 10^{-3}$, and 4.2 corresponds to $4.7 \cdot 10^{-4}$. Therefore, in a sequence of one million nucleotides, Pho is expected to be found either $1,400$ times or 470 times, respectively.

It is clear that a threshold cannot be defined independently from the matrix motif, whereas a *p*-value can. Therefore, *p*-values or derived *E*-values (expected number of occurrences) are sufficient and the threshold is adapted accordingly. Knowing the complete *p*-value distribution makes it easy to obtain the threshold in question, but some other publications deal with this specialized problem as well. Again, sampling approaches, e.g. used by MatInspector [55] or PRIMA [56], have to deal with the sparseness problem. Beckstette et al. [57] present an approach called *LazyDistrib*, which is part of their *PoSSuMsearch* package and mixes limited weight precision with look-ahead scoring for precise threshold computation. Unfortunately, they assume independently distributed background nucleotides.

Now that *p*-values can be recalculated to thresholds, the question arises what a "good" *p*-value is. A "good" *p*-value has the same requirements to fulfill as a threshold, stringent (close to zero) to minimize false positives, but also high enough to find many true positives. If no a priori knowledge about the expected number of motifs in a sequence is known (*E*-value), comparison to regular expression motifs is useful. They can be seen as a precursor to matrix motifs. Therefore, one valid alternative to obtain a *p*-value is to reconstruct the matrix motif to a regular expression motif, and use its calculated occurrence probability (Equation 2.5) for the *p*-value. If done automatically, it will probably fail, since

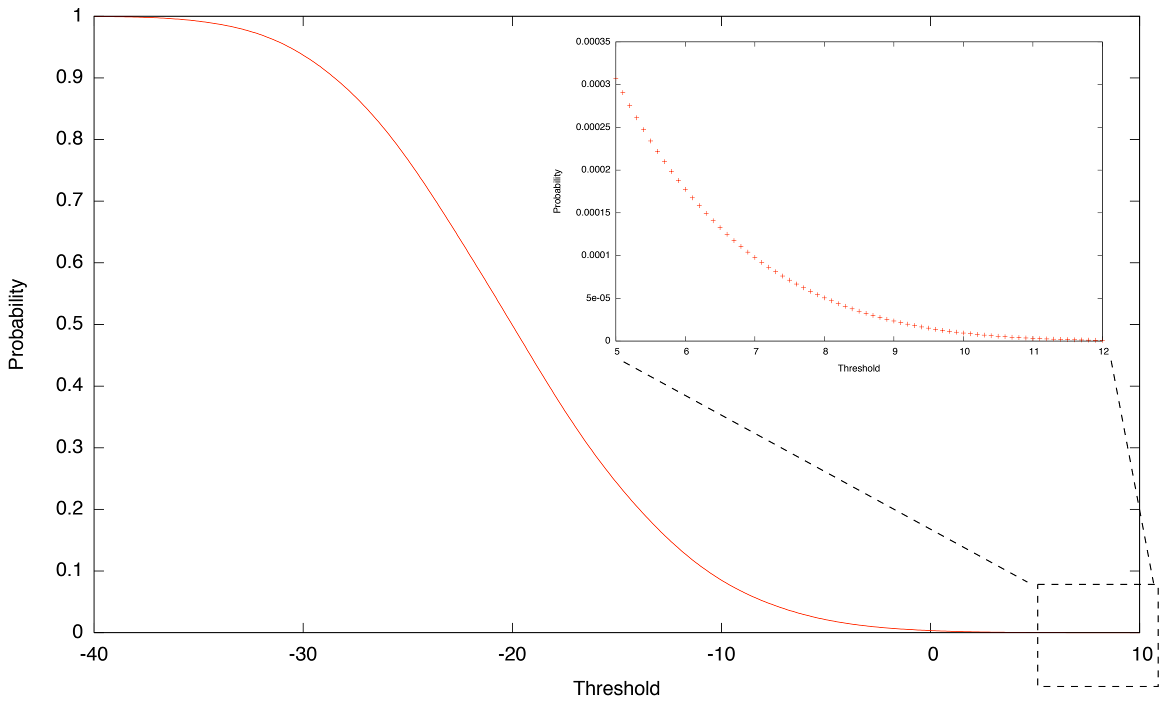


Figure 2.3: p-value distribution for the Pho PSSM under a zeroth order Markov model. The p-value for a threshold gives the probability to find the motif in a randomly generated sequence with this specific or higher threshold.

the matrix may be too biased, i.e. too many positions allow most or even all nucleotides to occur. It will cause the resulting sequence motif to consist of many N nucleotides, for which the probability to occur is always one. Therefore, the task is done manually. For every position a number between zero and one reflecting the weight distribution is assigned. Non-degenerated positions receive a one, fully degenerated positions a zero. A number close to zero is assigned to every positional vector in-between these two extrema. The numbers are added and the p-value is calculated as $1/4^n$. For the Pho matrix motif (Table 2.5) the resulting count was approximately 6.6, which in effect can be interpreted as the new non-degenerated length of the matrix motif. The corresponding p-value was calculated as 10^{-4}, and therefore the chosen threshold was 7.0.

2.2.3 Other single motif types

Two major drawbacks with all previously described motifs exist, which make them insufficient to use in special cases. Keeping in mind that a motif is meant to exhaustively represent all binding sites for one binding factor, the first drawback occurs if the binding factor recognizes sites which allow for gaps with variable length. Fixed-length gaps present no problem, since they can be modeled with the N nucleotide in regular expression motifs, or with the background vector in PSPMs, or with a zero-weight vector in PSSMs. One approach to solve this problem is to split the motif into fixed-size blocks, and to search these blocks separately. Afterwards they are combined to one motif by applying the gap constraints. For their index based search problem, Beckstette et al. (unpublished, personal communication) use the local chaining approach [58] to combine lists of single PSSM hits to the full length motifs. The program *fragrep2* [59] applies a dynamic programming approach to the problem even allowing for a deletion of single blocks. With the *jPREdictor* gaps of variable length can be modeled building higher order motifs, called MultiMotifs (see Chapter 2.2.5). A MultiMotif describes a motif pattern comprising single motifs linked together by gap constraints.

The second drawback occurs when a binding factor recognizes sites whose sequences are not alignable, e.g. *AAACCC* and *CCCAAA*. If an alignment would be forced, the consensus site would be *MMMMMM*. This consensus motif is very degenerated and can therefore also match sequences like *ACACAC*, which may not be biological binding sites. To solve this problem, the corresponding motif could be modeled to comprise alternative sites without aligning them. In the *PatSearch* program [60] with its regular expression-like language, it is possible to specify such a motif. In the *jPREdictor*, however, non-alignable binding sites must be split and defined as separate motifs.

Both these problems, variable-sized gaps and alternative sites, can effectively be modeled using Hidden Markov models (HMM). Nevertheless, in Bioinformatics, HMMs are particularly used for modeling protein families, the *Pfam* (protein families, [61, 62]) database being the most prominent example. This is because binding sites of nucleotides can often be modeled with simpler motifs.

2.2.4 Probabilities of sum-scores

In this chapter, a theoretical problem arising from the prediction of PRE/TREs (see Chapter 4) is investigated. In the prediction, motifs are searched in a sequence fragment, and for all found motifs the corresponding weights add up to a sum-score for the fragment. The theoretical task behind the prediction is then to assign a probability to the sum-score (*p*-value) that tells how likely it is to obtain this or a higher score merely by chance. An approach on how to calculate this probability is shown here, even though it will not solve the task to its full extend.

At first, a probability is given to encounter a certain motif m exactly x times. Let $p(m)$ be the probability for a single motif $m \in M$ to occur on a sequence merely by chance. The equation to calculate this probabilities is given in one of the previous chapters. Under the assumption of independent positions, the decision is made for every position along the sequence whether the motif occurs or not. This can be modeled as a Bernoulli experiment with success probability $p(m)$. The probability of a motif to occur exactly x times within a sequence of size l follows a binomial distribution with

$$p_{m,x} = \binom{l-|m|}{x} \cdot p(m)^x \cdot (1-p(m))^{l-x-|m|}, \tag{2.10}$$

where $|m|$ denotes the length of motif m. Given a set of motifs $M = (M_1, M_2, \ldots, M_N)$ and specific occurrence counts for these motifs on a sequence, $n = (n_1, n_2, \ldots, n_N)$, allows for the calculation of a probability to see this merely by chance:

$$P_{M,n} = \prod_{i=1}^{N} p_{M_i,n_i}. \tag{2.11}$$

Using n, M and a previously defined weight function w, which assigns weights to motifs, the score for the sequence can be calculated as: $score_{M,n} = \sum_{i=1}^{N} w(M_i) \cdot n_i$. Any calculated score for a given n can directly be assigned to a corresponding probability $P_{M,n}$ using Equation 2.11. Therefore, the function $: n \rightarrow (score_{M,n}, P_{M,n})$ holds. However, without a given n, no one-to-one relationship between score and probability exists, and thus, the functions : $score_{M,n} \rightarrow P_{M,n}$ as well as : $P_{M,n} \rightarrow score_{M,n}$ cannot be defined. This is true, since some permutations over n yield the same score as well as the same probability.

As a consequence of a missing one-to-one relation between score and probability, the relationship has to be defined over n. The task defined was to obtain a probability reaching or exceeding a defined cut-off score c. The naive way to obtain the probability is to enumerate over all possible settings for $n = (n_1, n_2, \ldots, n_N)$, to calculate the sum-score, and to check whether this sum-score exceeds the cut-off c:

$$P(score_M(S) \geq c) = \frac{\sum_{n=(n_1=0,n_2=0,\ldots,n_N=0)}^{(l,l,\ldots,l)} P_{M,n} \cdot 1\{score_{M,n} \geq c \wedge \sum_{i=1}^{N} n_i \leq l\}}{\sum_{n=(n_1=0,n_2=0,\ldots,n_N=0)}^{(l,l,\ldots,l)} P_{M,n} \cdot 1\{\sum_{i=1}^{N} n_i \leq l\}} . \tag{2.12}$$

For computing this equation the time complexity is $O(l^N \cdot N)$, thus not practicable, the space complexity would be $O(N)$.

It becomes more practical if only the *expected* overall-score (mean score) of an arbitrary sequence is of interest. This can directly be calculated from the expected occurrences of the single motifs within the window, thus

$$\mathbb{E}(score) = \sum_{i=1}^{N} \mathbb{E}_l(M_i) \cdot w(M_i) = \sum_{i \geq 1} (l - |M_i| + 1) \cdot p(M_i) \cdot w(M_i) . \tag{2.13}$$

For instance, the motifs *AAAAAA*, *ACGTAC*, and *YGAGNCTCY* are searched on a uniformly distributed DNA sequence, therefore with occurrence probabilities given in Table 2.1. Let the weights of the motifs be ln(4), ln(3) and ln(2), respectively. The sequence length may be $l = 500$. From Equation 2.13 follows: $\mathbb{E}(score) = 0.3211$. Simulations with a sliding window on a sequence of length $2 \cdot 10^9$ yielded a mean score of 0.3204. Applying a sequence with *Drosophila melanogaster* character distribution (for the probabilities of the single motifs refer to Table 2.1) Equation 2.13 gives $\mathbb{E}(score) = 0.5250$, whereas simulation yielded 0.5257.

The sum-of-faces problem was thought to give insights into how the probability for a certain cut-off might be calculated. The problem definition is similar to the sum-of-weights problem. In addition, the presented solution uses generating function, which can be applied to the sum-of-weights problem as well. Prerequisite for the sum-of-faces problem is a die, which is rolled k times and the numbers produced are summed. Applied to the sum-of-weights problem it would mean that motif occurrences are rolled and the weights are summed. In the sum-of-faces problem, the probability that the die-sum becomes exactly m is of interest. To calculate this probability, generating functions are used. Let $f(m)$ be the number of sequences of length k with all terms in $\{1,2,3,4,5,6\}$ and with sum m. Let F be the polynomial in X with the coefficient of X^m being $f(m)$. Let $G = X + X^2 + X^3 + X^4 + X^5 + X^6 = \frac{X(1-X^6)}{1-X}$ the generating function to get a number between one and six (in the powers) in only one throw. Now the die is thrown k times, thus

$$\begin{aligned} F &= G^k \\ &= X^k \cdot (1 - X^6)^k \cdot (\sum_{i=0}^{\infty} X^i)^k . \end{aligned}$$

The first term X^k means that the minimal sum to get is k, which occurs only once. Further transformation of the above equation yields

$$= X^k \cdot \sum_{i=0}^{k} (-1)^k \cdot \binom{k}{i} \cdot X^{6i} \cdot \sum_{i=0}^{\infty} \binom{i+k-1}{k-1} \cdot X^i$$

$$= X^k \cdot \sum_{i=0}^{\infty} \sum_{j=0}^{\min(k, \lfloor \frac{i}{6} \rfloor)} (-1)^j \binom{k}{j} \binom{i-6j+k-1}{k-1} \cdot X^i \, .$$

This gives $f(m) = \sum_{j=0}^{\min(k, \lfloor \frac{m-k}{6} \rfloor)} (-1)^j \binom{k}{j} \binom{m-6j-1}{k-1}$. The probability is then $P(sum_k = m) = \frac{f(m)}{6^k}$.

Applying the sum-of-faces problem to the sum-of-weights problem, the corresponding generating function is

$$G = (1 - \sum_{i=0}^{N} p_i) + p_1 \cdot x^{w_1} + p_2 \cdot x^{w_2} + \ldots + p_N \cdot x^{w_N} \, , \tag{2.14}$$

with $p_i = p(M_i)$ and $w_i = w(M_i)$. Calculating G^l, which means that maximal l single motifs would fit on the sequence, yields the probability to reach exactly that sum-score (in the coefficient) for every possible sum-score c (in the power). This is written as $[x^c]G^N$. To make c a cut-off the probabilities to exceed that cut-off have to be summed-up, thus $[x^{\geq c}]G^N = \sum_{i=c}^{\infty} [x^i]G^N = 1 - [x^{<c}]G^N = \sum_{i=0}^{c-1} [x^i]G^N$. Unfortunately, weights might be arbitrarily real numbers and not integers, thus computing G^N together with keeping track of all scores in all steps leads to a time/space complexity of $O(l^N)$.

Therefore, the problem is simplified. All weights are set to one. This obviously disrupts the concept behind the weights of motifs, but at least, the simplification leads to a solution. Let $p = \sum_{i=0}^{N} p_i$ and $q = 1 - p$. The generating function becomes $G = q + p \cdot x$ and calculating this to the power of l (binomial theorem) gives

$$G^l = (q + p \cdot x)^l = \sum_{n=0}^{l} \binom{l}{n} q^n p^{l-n} x^{l-n} = \sum_{n=0}^{l} \binom{l}{n} p^n x^n (1-p)^{l-n} \, . \tag{2.15}$$

Finding a weight exactly c, which is the same as encountering exactly c single motifs, gives $[x^c]G^l = \binom{l}{c} p^c (1-p)^{l-c}$. The probability to reach a cut-off c or higher is given by $[x^{\geq c}]G^l = \sum_{n=c}^{l} \binom{l}{n} p^n (1-p)^{l-n}$.

In practice, for every sequence fragment the numbers of occurrences of all motifs are known, because the motifs are searched on the sequence fragment. Therefore, the probability of encountering exactly this number of occurrences can easily be calculated using Equation 2.11. However, this is not the probability of exactly obtaining the corresponding sum-score, because other combinations of occurrences might produce the same sum-score. Therefore, the actual probability of approaching exactly this sum-score is higher. This exact probability can be calculated, if all weights are one. Because the motifs are assigned other weights, the problem for arithmetically calculating the question on how likely it is to exceed a specific sum-score for a sequence fragment remains unsolved.

2.2.5 MultiMotifs

MultiMotifs are motifs of higher order, which comprise basic motifs and distance constraints. Basic motifs are all kinds of single motifs as well as other MultiMotifs. Formally, they can be described in the form $M = M_1G_1M_2G_2 \dots M_{k-1}G_{k-1}M_k$. This is simply a concatenation of arbitrary motifs linked by gap specifications, i.e. minimal and maximal distance. Distance is always measured from the end of one to the beginning of the subsequent motif in terms of positions on a sequence. Examples for MultiMotifs can be found in Chapter 4.1, where they are used in the prediction of PRE/TREs.

The expected number of one MultiMotif in a long sequence S can be calculated using equations introduced in [28]. For a MultiMotif consisting of N basic motifs and with known distance constraints between every two single motifs (G^{max} maximal distance, G^{min} minimal distance), the expected number of occurrences is $E = \prod_{i=1}^{N} P_iR_i$ with $R_1 = |S|$ and $R_i = G_i^{max} - G_i^{min}$.

In statistical evaluation, MultiMotifs break the linear relationship between occurrences on a sequence and counting these occurrences. While each basic motif found on a sequence counts one as occurrence, MultiMotifs depend on several motifs to occur cooperatively. A double motif, for instance, is only found if both basic motifs not only occur, but also occur nearby, thus fulfilling the distance constraints. Note that the distance constraints prevent occurrences of a double motif in case only one basic motif is actually found. If a double motif contains the same motif twice, and if two instances of this basic motif are found, the double motif is reported once. If a third instance of the basic motif is found, the double motif is reported thrice already. A fourth instance rises this number to 6, a fifth to 10, and so on. Actually, under the assumption that all distance constraints are fulfilled, the number of reported double motifs is calculated as $\binom{k}{2}$, with k being the number of found basic motifs. This is a quadratic increase. With triple motifs, the increase is to the power of three: $\binom{k}{3}$.

As a consequence, even small assemblies of basic motifs in one spot on the sequence will produce large numbers of matching MultiMotifs. If the number of reported MultiMotifs is rewarded, as in the prediction process, enrichment and clustering of basic motifs are encouraged. However, if the distance constraints are restrictive, more emphasis is placed on densification. With more relaxed distances, at least the enrichment of basic motifs in the whole sequence fragment is strongly encouraged.

2.3 Basic motifs used in this thesis

In this chapter, the basic motifs are listed, which are used in this thesis. Basic motifs are sequence motifs or matrix motifs, but not motif patterns or composite motifs. Normally, motifs and corresponding transcription factor binding sites (TFBS) are curated and assembled in databases. This is the first source for generating a motif from binding sites. The Regulatory Element Database (REDfly, [63]) is a collection of known *Drosophila* transcriptional *cis*-regulatory modules (CRMs) and transcription factor binding sites. Since October 2007 the *Drosophila* DNase I footprint database (FlyReg) has merged into REDfly, adding 1.365 DNase footprints obtained from systematic curation and genome annotation [35]. The *Drosophila* Transcription Factor Database (FlyTF, [64]) stores characterized and putative site-specific transcription factors. Like FlyBase [65], it lacks information about the motifs bound by the transcription factors. FlyMine, on the other hand, is an integrated database of genomic, expression and protein data, not only for *Drosophila*, but also for *Anopheles* and *Caenorhabditis elegans* [66]. Integrating data makes it possible to run sophisticated data mining queries that span domains of biological knowledge. TRANSFAC is a database on eukaryotic cis-acting regulatory DNA elements and trans-acting factors [67].

For PRE/TRE regulatory elements, significant motifs, among others, are the binding site for the GAGA factor (GAF) [68] and Pipsqueak (PSQ), for the Zeste (Z) protein [69], and for Pleiohomeotic

Table 2.6: List of all basic motifs used in the prediction of PRE/TREs and in robustness tests, together with their consensus sequence, number of errors allowed for a match (not applicable, n.a., in case of a PSSM), number of occurrences in the *D. melanogaster* genome and the corresponding log-odd against a random sequence generated under a zeroth order Markov model.

Name	Description	Motif / Consensus sequence	Errors	Occurrence	Odd
En1	Engrailed 1	GSNMACGCCCC	1	20,209	1.51
G10	GAF long	GAGAGAGAGA	1	35,493	2.40
GA/ GAF	GAF short	GAGAG	0	184,542	-0.02
Grh	Grainyhead PSSM	BRRWCYGGTTTT	n.a.	113,464	0.13
HB	Hunchback PSSM	TTTTTTRTKVB	n.a.	1,049,224	1.16
Kni	Knirps PSSM	AACKAGAKCA	n.a.	31,111	0.21
PF	Pho trail	GCCATHWY	0	62,473	0.39
Pho- DSP1	Pho-DSP1 double	GCCAT-(0,40)-GAAAA	n.a.	119,119	2.39
PM	Pho long	CNGCCATNDNND	0	41,263	0.51
Pho	Pleiohomeotic PSSM	SKCRGCCATYWYGG	n.a.	40,552	0.56
PS	Pho core	GCCAT	0	290,570	0.44
SP1	SP1/KLF	RRGGYG	0	319,722	0.12
Z	Zeste	YGAGYG	0	193,733	0.17

(Pho, see Chapter 2.2.2). For the Pho motif binding experiments were performed [41, 43, 42, 44, 51, 45] resulting in a position specific score matrix (PSSM) for the motif. Recent publications reported some more motifs to play a role in PRE/TRE functionality: SP1/KLF [9], Dorsal switch protein 1 (DSP1) [10] and Grainyhead (Grh) [11]. In Table 2.6, the sequence representations for these motifs are listed.

Some motifs are reported ambiguous. In this case, the decision has to be made, which motif to use. GAF and PSQ, for instance, bind repeated binucleotides, $(GA)_n$, where n can vary between 1.5 and 5 [70]. *In vitro*, GAF requires at least 2.5 repeats to be bound properly [71, 72]. Nevertheless, reports on GAF binding to only the trinucleotide *GAG* exist [70], but this may be due to additional GAF binding sites or other unrelated motifs nearby [73]. Consequently, two GAF motifs are used in this thesis, a short one, $(GA)_{2.5}$, and a long one, $(GA)_5$, the latter with one mismatch allowed (Table 2.6). The Zeste protein binding site was reported as *YGAGYG* [74] as well as *BGAGTGV* [75]. Here, *YGAGYG* will be used as Zeste motif, because this one is well studied, whereas the other one was reported and used only once [9].

The PSSM motifs used in this thesis are Pho, Grh, HB, and Kni (Table 2.6). Pho was already explained in Chapter 2.2.2. With a threshold of 7.0 it was designed to recognize 21 out of 23 reported binding sites and to avoid 10 out of 10 sequence fragments reported to be not bound by Pho. Grainyhead (Grh) was analyzed by Blastyak et al. [11] for its importance to PRE functionality. They report Grh to interact with the *iab-7* PRE and to work cooperatively with Pho. Consequently, the Grh motif as presented in Table 2.7 was used in further studies.

Both, Hunchback (HB) and knirps (Kni) play a role in enhancer/silencer functionality. They are included into PRE/TRE analysis as a kind of robustness test. The PSSM for HB (Table 2.8) as well

Table 2.7: The Grainyhead (Grh) motif as a PSSM. The most significant positions are shaded in blue. On the right side, the corresponding encoding into an option file format is shown. The binding sites were reported in [11] and aligned and cut to a length of 12. A threshold of 5.9 corresponds to roughly 422 matches in a sequence of one million nucleotides.

Position	*A*	*C*	*G*	*T*
1	-4.61	0.57	0.17	0.27
2	0.27	-0.51	0.57	-0.81
3	0.78	-4.61	0.17	-0.81
4	0.27	-4.61	-0.51	0.56
5	-0.81	1.08	-0.51	-0.81
6	-4.61	1.08	-4.61	0.27
7	-4.61	-4.61	1.26	-0.13
8	-4.61	-4.61	1.55	-4.61
9	-4.61	-4.61	-4.61	1.25
10	-0.81	-0.51	-4.61	0.96
11	-0.81	-0.51	-4.61	0.96
12	-4.61	-0.51	-4.61	1.12

```
[Motif]
name = pssmGrh
motif = MULTI_SEQUENCE
CAATCCTGTTTT # FP1
TGATTCGGTATC # Ftz I
GGGAGCTGTTTT # Ftz II
TCGGATGGTTTT # Ftz IIIa
CTTTCTGGTTTT # Ftz IV
CAATCTGGTTTT # Ubx
GAAACCGGTTAT # Ddc be-I
TGAACCGGTCCT # Ddc be-II
background =
0.2877 0.2124 0.2123 0.2876
threshold = 5.9
```

as for Kni (Table 2.9) is the same as generated and used by Berman et al. [76, 77]. Note that current versions of TRANSFAC as well as REDfly list additional binding sites, which were not taken into account by Berman et al. However, both PSSMs are not recreated for reasons of comparability.

Very often it is not sufficient to use basic motifs, because distance or co-occurrence constraints with other binding sites might exist, no matter if it is of the same or of another type. In this case, motifs are combined to higher order motifs. Dejardin et al. [10] reported the DSP1 motif to occur near the Pho motif. Therefore, the double motif $GCCAT-(0,40)-GAAAA$ was created, combining the Pho core motif with DSP1 (Table 2.6). Other reports on co-occurrence constraints and context-dependent binding exist for GAF [73] and Zeste [78, 79]. This is one reason for combining basic motifs to double motifs in the prediction process.

2.4 Motif discovery and clustering

2.4.1 Motivation behind the de-novo motif discovery

In the process of transcription and translation genes act as templates for proteins. Regulation of the protein production enables cells and organisms to react to changes in the environment and still fulfill their needs. Therefore, the genetic machinery has means to activate, deactivate, up- and down-regulate the expression of single genes. A major challenge in biology is to understand the regulation of gene expression. Regulatory sequences such as promoters, enhancers/silencer, and PRE/TREs, control the transcription process, and biologists and bioinformaticians strive to identify these regulatory elements.

Transcriptional control is often accomplished by the binding of multi protein factors to regulatory sequences and thereby determination of the transcription level. Based on this mechanism, discover-

Table 2.8: The Hunchback (HB) motif as a PSSM. The most significant positions are shaded in blue. On the right side, the corresponding encoding into an option file format is shown. The PSPM table was reported in [76]. A threshold of 4.4 corresponds to roughly 1.392 matches in a sequence of one million nucleotides.

Position	*A*	*C*	*G*	*T*
-6	-0.78	-0.48	-0.66	0.80
-5	-4.61	-4.61	-4.61	1.25
-4	-3.05	-1.55	-2.20	1.17
-3	-4.61	-4.61	-4.61	1.25
-2	-4.61	-4.61	-4.61	1.25
-1	-4.61	-2.81	-4.61	1.24
1	0.61	-0.77	0.28	-1.06
2	-0.44	-0.14	-0.48	0.57
3	-2.47	-0.08	0.82	0.06
4	0.02	0.24	0.20	-0.44
5	-1.06	0.28	0.35	0.13

```
[Motif]
name = HB
motif = TABLE_PROB
#pos A C G T
-6 12 12 10 59
-5 0 0 0 93
-4 1 4 2 86
-3 0 0 0 93
-2 0 0 0 93
-1 0 1 0 92
1 49 9 26 9
2 17 17 12 47
3 2 18 45 28
4 27 25 24 17
5 9 26 28 30
background =
0.287 0.213 0.213 0.287
threshold = 4.4
```

Table 2.9: The Knirps (Kni) motif as a PSSM. The most significant positions are shaded in blue. On the right side, the corresponding encoding into an option file format is shown. The PSPM table was reported in [76]. A threshold of 5.4 corresponds to roughly 107 matches in a sequence of one million nucleotides.

Position	*A*	*C*	*G*	*T*
-5	1.25	-4.61	-4.61	-4.61
-4	1.25	-4.61	-4.61	-4.61
-3	-0.35	1.04	-0.05	-4.61
-2	-4.61	-4.61	0.64	0.74
-1	1.03	-0.05	-4.61	-4.61
1	-4.61	-4.61	1.55	-4.61
2	0.74	-0.05	-0.05	-4.61
3	-4.61	-0.05	0.64	0.34
4	-4.61	1.55	-4.61	-4.61
5	1.25	-4.61	-4.61	-4.61

```
 [Motif]
name = Kni
motif = TABLE_PROB
#pos A C G T
-5 5 0 0 0
-4 5 0 0 0
-3 1 3 1 0
-2 0 0 2 3
-1 4 1 0 0
1 0 0 5 0
2 3 1 1 0
3 0 1 2 2
4 0 5 0 0
5 5 0 0 0
background =
0.287 0.213 0.213 0.287
threshold = 5.4
```

ing the binding sites of transcription factors would enable the identification of regulatory sequences. Binding sites are computationally represented as motifs. If the binding sites are identified, they can help to identify regulatory sequences, which are characterized by occurrences of these binding sites.

In case the motifs characterizing a regulatory element are unknown, they have to be discovered. This goes as far as having only one upstream region of a gene, in which the binding site is suspected. Dejardin et al. [10], for instance, filled the inter-motif space of the 219 nucleotides long PRE *Ab*-Fab with random nucleotides, which caused the PRE to lose its functionality. This PRE could be such a candidate for motif discovery, and, indeed, Dejardin et al. [10] came up with a novel motif, DSP1, although they used no motif discovery tools. But this is an extreme scenario, having only a single very short sequence, and to carry it even further, without information about existing motifs and about the background. Normally, a set of upstream sequences is provided, and for all of them, the corresponding gene is regulated by one or at least few transcription factors. There are many possible sources for such putatively co-regulated genes, including expression microarray experiments (e.g., following a gene knockout experiment) and functional classes from literature (gene ontology). The working assumption is that all provided sequences house shared binding sites for one or more transcription factors.

In molecular biology, either DNase I footprinting [36] or *in vitro* binding site selection experiments [37] can be used in order to analyze which sites are bound by a transcription factor. For instance, Mahmoudi et al. [80] used DNase I footprinting to reveal the binding sites of Pho, Zeste, GAF and HB in the *Ubx* PRE. However, such experiments have two major disadvantages: first, their resolution is low, and second, they fail if the transcription factor is unknown. The binding sites Mahmoudi et al. [80] revealed are more than 15 nucleotides in length. This first disadvantage can be overcome by a good multiple alignment strategy, or, if the binding sites are too long, by motif discovery strategies. Especially Gibbs sampling approaches will work well, since they assume one motif in every sequence. The disadvantage of the unknown binding factor cannot be overcome so easily. Experiments to test a set of sequences for co-regulation using all known factors are very expensive and time consuming, and can fail, if a novel, up-to-now unknown factor is involved. In this case, powerful computational algorithms for motif discovery are applied revealing the binding sites used for further biological testing.

2.4.2 Motif discovery problem

The problem of motif discovery can be formulated as follows: given a set of sequences, find an unknown motif that occurs frequently. An equivalent definition of this problem is given by Li et al. [81] who proved it to be NP-hard. Nevertheless, if such a motif is of fixed length n and if it occurs in all sequences, one single solution is to simply enumerate all possible binding sites of length n and to report the ones fulfilling the constraints. However, the motif to discover usually is of unknown length. This could be overcome by starting the enumeration approach with different lengths. The real problem is that from one sequence to the next, the binding sites might change slightly due to mutations and varieties of nucleotides. In this case the enumeration approach fails.

The unknown length of the motif and the level of degeneration result in a huge search space, which cannot be exhaustively covered by enumeration methods. Nevertheless, biological binding sites are of restricted length. Therefore, the length of the corresponding motif will not be arbitrary. Sample motifs and the ones curated in databases like TRANSFAC suggest that motifs are 5 to 15 nucleotides long, rarely longer. Motifs can occur on both strands of the DNA. Computationally, this is solved by matching the motif on the sequences either as it is, or after reversing and complementing (inversion) the motif. The number of binding sites on one sequence of the set is not limited. It can have zero, one, or multiple copies of a motif.

2.4.3 Available tools

The discovery of motifs in a given set of sequences has been addressed often before and a large number of algorithms have been developed. Das and Dai [82] survey several motif finding algorithms and conclude that, despite considerable effort, the problem remains a complex challenge. Motif discovery algorithms can be categorized into two major classes: (1) word-based methods, and (2) probabilistic models such as Gibbs-sampling approaches. Since word-based methods most often exhaustively enumerate the search space, they guarantee global optimality. They are a good choice for finding totally preserved binding sites. However, this is the weak spot as they ignore degenerated positions or only allow for few positions to be degenerated. Nevertheless, Sinha and Tompa [83] argue that, since the number of well conserved nucleotides is usually in the range of 6 to 10, and since positional variations are rare and can be covered mostly by the letters of the IUPAC code, the use of enumerative methods can be afforded. Consequently, they present their Yeast Motif Finder (YMF, [84]), which enumerates motifs over 8 letters of the IUPAC code, and allows for spacers in the middle of the motif.

Probabilistic approaches represent a motif as a position weight matrix (PSPM or PSSM). *MEME* (Multiple EM for Motif Elicitation, [85]) is its most famous member and uses an expectation maximization approach. The software Consensus [86], on the other hand, uses a greedy approach for aligning sequence fragments into a position probability matrix. With every cycle of the algorithm, the matrices with the highest information content are kept.

Among the probabilistic approaches, Gibbs sampling is used most extensively. It was introduced by Lawrence et al. in 1993 [38]. The central idea is to construct a position weight matrix from one short fragment randomly chosen out of every sequence from the set. In every cycle, one sequence is drawn and the pattern is constructed excluding the fragment of the chosen sequence. Every position in that sequence is then matched by the pattern, forming a distribution over the sequence (similar to Figure 2.2). The starting position for the new fragment from the left-over sequence is then drawn in accordance to the normalized probabilities of the distribution.

The main criticism with probabilistic approaches is that they often get stuck in local optima. This is a criticism for most, if not all, hill-climbing strategies. Great care must be taken in order to allow an algorithm to jump over the search space and try to climb many hills. For sampling approaches this is solved by restarting them multiple times. With every restart they are randomly initialized.

Algorithms based on Gibbs sampling are AlignACE (Aligns Nucleic Acid Conserved Elements, [87]), MotifSampler [88], and BioProspector [89]. All algorithms more or less extend the original Gibbs sampling approach, mostly in terms of improving the background model to higher-order Markov chains, but also in terms of masking out already found motifs, in improving sampling strategies, or in judging the significance of motifs. Recent extensions are the recursive [39] and the centroid Gibbs sampler [40].

2.4.4 Motif discovery through motif evolution

In this work, a motif discovery approach based on the *in silico* evolution of motifs is presented. Starting with randomly chosen motifs or most unspecific motifs, mutations are introduced from one generation to the next. The children have to compete against their parents as well as themselves in terms of high weights. In order to avoid local optima, first, the number of motifs in one generation is practically large, and second, a simulated annealing process is applied for the number of mutations allowed in-between generations. To balance occurrences among all provided sequences, the selection is preceded with a rewarding step.

The idea behind an evolution strategy is that small changes in the properties of an object lead to an improvement of the objective function. This general optimization strategy can be visualized as hill-climbing. For motifs, the changes include base replacements, and increasing or decreasing the positional degeneration level. From one generation to the next, the motifs are mutated to identify a motif, which maximizes the weight. Jumps over the search space are possible by strongly mutating the motifs.

Applying evolutionary algorithms to motif discovery is novel. Recently, Wei and Jenson introduced a software, GAME, which utilizes a genetic algorithm to find optimal motifs in DNA sequences [90]. The tool works similar to the approach presented here in that it evolves motifs with high fitness from a population of randomly generated starting motifs. Nevertheless, it lacks the ability to give motifs as a prior knowledge. It also lacks the ability to evolve motif patterns as well as certain characteristics of motifs, e.g. the error number allowed for a match. Recently, Chan et al. [91] introduced a genetic algorithm for motif discovery. They restrict the search to one binding site per sequence like in Gibbs sampling approaches. The *jPREdictor* allows for arbitrary many motif matches in every sequence, and even zero occurrences are possible.

In addition, the *jPREdictor*'s *in silico* evolution of motifs makes use of background sequences, which only few motif discovery algorithms do. Most often, they rely on more or less powerful background models like Markov chains. Algorithms that use sequences as background are called “discriminative”. Such approaches, among others, are *ALSE* [92] and *DIPS* [93]. Recently, *DEME* [94] was introduced, adding to the number of discriminative algorithms. Redhead and Bailey [94] showed that with artificial data *DEME* is more effective than a non-discriminative approach when there are "decoy" motifs or when a variant of the motif is present in the "negative" sequences. Nevertheless, applying *DEME* to real data showed that it is as good as non-discriminative algorithms at discovering yeast transcription factor binding motifs [94].

The motif discovery incorporated into the *jPREdictor* specifically was developed to identify over-represented double motifs. The novelty is that the allowed distances in the double motifs can have almost arbitrary values up to a maximum. Another novelty is that the distance is subject to mutation, too. Other tools for the discovery of co-localized binding sites are BioProspector, when run in two-block mode [89], and a tool called HeliCis [95], which relies on periodic spacing. Nevertheless, BioProspector limits the allowed distance to 50, and HeliCis was developed with periodic distances of 10 to 11 nucleotides in mind. Additionally, the *jPREdictor* features the identification of motif patterns with higher order than double, a feature both BioProspector and HeliCis cannot afford.

2.4.5 Objective functions

The basic idea behind *in silico* motif discovery is over-representation. A binding site remains undetected, if it does not occur more frequent than a background model suggests. This over-representation is somehow expressed as an objective function, which is to be maximized throughout the motif discovery process. Li and Tompa [96] analyzed certain objective functions used in recent motif discovery tools for their usefulness: the famous log-likelihood ratio, *Z*-score, and sequence specificity. The log-likelihood compares two likelihoods: the likelihood of approaching the given set of sequences under the current motif model and the likelihood of approaching the sequences under a background model. It is a relational measure and used, for instance, by *MEME* [85]. The *Z*-score simply calculates the distance between the number of observed motifs and the number of expected motifs. It is used, for instance, by the *YMF* program [84]. The sequence specificity function emphasizes on the fact that a predicted motif should have a balanced number of binding sites over all provided sequences. This concept is carried to the extreme by Gibbs sampling approaches, which assume exactly one binding site on every provided sequence. Nevertheless, balancing out the occurrences of motifs is important, and therefore it is incorporated into the evolutionary process presented in this thesis as well. For all discussed objective functions, Li and Tompa [96] concluded that they alone are not able to separate the true motifs from the background noise. As a result, they discussed a new objective function, and tested it with the *MEME* tool (see [96]). It performed very well in terms of separation. However, the disadvantage is that it adds to the number of free parameters, for which a training step would be necessary.

In this work, the weights of motifs act as objective function. Weights are very similar to log-likelihood ratios in the way how they are calculated (see Chapter 4.4 for the implementation details into the *jPREdictor*). Nevertheless, no probabilities are used, but the weight calculation directly sets off the numbers of found motifs from both training sets. The negative training set acts as background, and the positive training set as model. No probabilities or expectations are incorporated into the equation. It is conveniently simple, the background sequences define the binding sites to avoid and the model defines the site which has to be matched frequently by the motif.

In this thesis, the evolution of motifs for the purpose of discovering over-represented motifs uses the weight function for measuring the enrichment. In other words, motifs are evolved to have maximal weight. Nevertheless, the used objective function gives rise to a dilemma briefly explained now and referred to in this thesis as the weight dilemma. The weight function is a relational function, and this relation is expressed as a fraction. As with all objective functions, occurrences of a motif in the model result in higher weights the fewer the occurrences in the background are. The dilemma is now that occurrences in the background have more impact on the weight than occurrences in the model sequences. This results in discovered motifs mainly avoided in the background.

This dilemma is of highest consequence, if motifs are to be discovered which are short and highly degenerated. Such motifs occur very frequently in the model, but also relatively often in the background. Because of its occurrences in the background, this motif will have a low weight, almost no matter how often it occurs in the model. Few motif discovery tools will find such a motif. The reason is that most tools use a relational objective function similar to the log-likelihood [96]. In contrast, tools using the *Z*-score objective function, e.g. ANN-Spec [97], and YMF [84], have the highest chance of capturing such a motif. An example might make this clear (also see Figure 2.4). A certain motif occurs 100 times in the model sequences. If it occurs 10 times in the background its weight is around

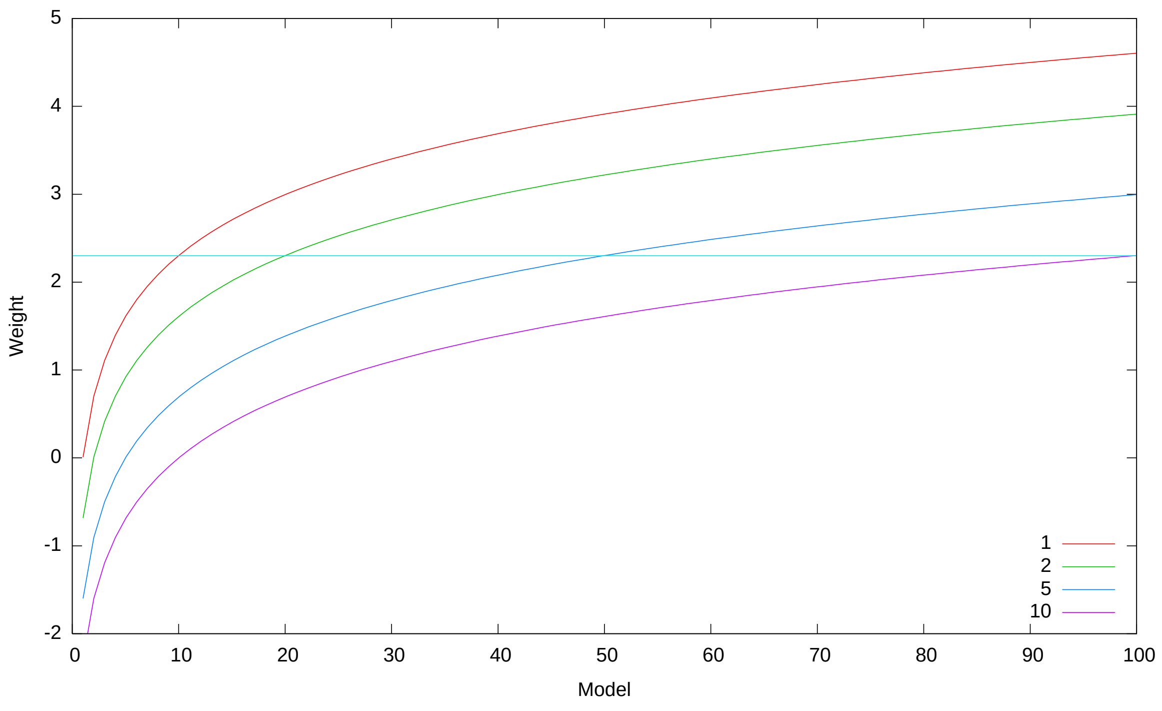


Figure 2.4: Visualization of the weight dilemma. The x-axis gives the number of occurrences of a motif in the model sequences. The motif does occur in the background sequences once (red line), twice (green line), 5 times (blue), or 10 times (violet). In order to obtain a weight of 2.3 (horizontal line), 100 occurrences of a motif in the model and 10 occurrences in the background are necessary. In contrast, if the motif occurs only once in the background, not more than 10 occurrences in the model sequences are necessary. As a result, motif discovery tools based on maximization of log-likelihoods or relative objective functions mainly report back motifs which are avoided in the background sequences.

2.3. Considering the most simple *Z*-score as the distance between both occurrences, the corresponding *Z*-score would be 90. Now note that motif discovery relies on maximizing the objective function, in case of the *jPREdictor*, any motif with a higher weight than 2.3 would be preferred. Consequently, any motif occurring only twice in the background gets a higher weight if it occurs more than 20 times in the model. For motifs occurring once in the background, 10 occurrences in the model are enough to exceed the weight of 2.3. In both latter cases the *Z*-score would be 18 and 9, respectively. This shows that the *jPREdictor* and most other tools will discover motifs almost non-present in the background, while tools using the *Z*-score would find motifs largely present in the model. In this sense, combining both objective functions would certainly be beneficial to motif discovery, but remains a task for the future [96].

2.4.6 Motif clustering background

The goal of clustering is to reduce the amount of data by categorizing or grouping similar data items together. Clustering methods can be divided into two basic (non-probabilistic) methods: hierarchical and partitional clustering. Hierarchical clustering either combines cluster objects together to form higher order clusters (agglomerative, bottom-up) or breaks up big clusters to form smaller ones (divisive, top-down). As representation of this hierarchy a dendrogram is often used. The other method is partitional clustering, which tries to directly decompose the data set into disjoint clusters (e.g. K-means and derivatives).

Motif discovery approaches, especially when combined, yield many motifs that are very similar in terms of nucleotide arrangement. Often, two motifs differ only in one or two positions. In this case, a subsequent clustering of the motifs has to be applied. Jensen et al. [98], for instance, rerun the BioProspector tool [89] multiple times with different starting parameters, followed by an optimization step via BioOptimizer [99] and the actual clustering step. Their clustering model is very complex: they combine a Bayesian model to calculate the probability for a motif to belong to a certain cluster and a Gibbs sampling algorithm to find the optimal cluster for that motif. Opposed to this probabilistic approach, hierarchical clustering methods are much simpler. Matlign [100], for instance, calculates the distance between each two motifs and clusters according to the lowest distance. However, the clustering used in Matlign is problematic, since the alignment allows for gaps. The problem is that a match of a motif is gapless and that the insertion of gaps into a motif changes the motif to a degree, where it matches to binding sites different from the ones it was discovered to bind to.

Clustering methods rely on measurements that describe in one way or another how close two objects from the data set are. This can be distances or similarities. Two objects with a low distance are preferably joined into the same cluster as compared to two objects with higher distance. Measurements also must be defined between cluster and single object and between two clusters. Calculation methods for this in turn rely on the representation of a cluster, either as a composition of single elements or with a representative, centroidic, or consensus element. Single linkage, complete linkage, average linkage, and average group linkage see a cluster as a composition of single elements, whereas median, centroid and Ward's method take the cluster's representative for distance calculation. The clustering implemented into the *jPREdictor* does not use linkage distances, because comprising motifs into a central motif is preferred. The representative of a cluster will be the average motif after alignment of all of its members. Ward's distance is also not implemented, because it favors clusters to be homogenous in size.

Test	Condition			
Outcome	*Illness*	*Health*		
Positive	True positive (TP)	False positive (FP)	$\rightarrow$	Positive predictive value $PPV = \frac{TP}{TP+FP}$
Negative	False negative (FN)	True negative (TN)	$\rightarrow$	Negative predictive value $NPV = \frac{TN}{TN+FN}$
	$\downarrow$	$\downarrow$		
	Sensitivity$= \frac{TP}{TP+FN}$	*Specificity*$= \frac{TN}{TN+FP}$		

Figure 2.5: Statistical measures for the evaluation of a test. Every sample/object is in one of two well defined conditions, e.g. illness/health. The test has to guess which condition it is. Both, condition and test outcome are binary.

Applying hierarchical clustering yields a dendrogram, where the lowest level, 0, consists of single element clusters. The highest level is occupied by only one cluster comprising all motifs. If the number of resulting clusters is not known beforehand, the decision has to be made, at what height the dendrogram is to be cut in order to yield the optimal number of clusters. This is one of the central problems both in hierarchical and non-hierarchical clustering. Visual inspection of the dendrogram is not preferred, since it is error prone and unfeasible due to the large data set. Many formal methods to get the optimal number of clusters have been proposed ([101] and references therein). Not only do they vary widely in terms of accuracy, but their performance also depends on the distance measure used. The *jPREdictor* implements three different distance measures. For the Manhattan and Euclidean distance measure, the cut-off is directly defined in terms of the measure. If the distance measure is likelihood, the information criteria is checked at every clustering level. When it decreases too much from one level to the next clustering stops.

2.5 Statistical evaluation

The statistical measures used in this thesis are shown in Figure 2.5. In a test environment, each object or sample, respectively, is in one of two well-known conditions. In case the condition is illness, the patient is either sick or not. Note that the condition is binary, there is no such thing as being a little ill. A test is applied to guess the condition of the patients. The null hypothesis is that the patient is sick. Again, the test is binary. It yields a positive, i.e. confirming, or a negative, i.e. rejecting, result, whether it tests the patient to be sick or not, respectively. The statistical outcomes of the test are true positives (abbreviated TP, the patient is really sick and the test confirms this), false positives (FP, patient is healthy, but is tested to be ill), false negatives (FN, patient is ill, but is tested to be healthy), and true negatives (TN, patient is healthy and tested accordingly).

In order to estimate the performance of the test, several statistical parameters can be calculated using the tests outcomes. Such parameters are the positive predictive value (PPV), negative predictive value, sensitivity and specificity (for the equations refer to Figure 2.5). PPV and sensitivity relate the number of true positives to the overall number of positively tested cases ($TP+FP$) and to the overall number of ill cases ($TP+FN$), respectively. On the other hand, negative predictive value and specificity relate the number of true negatives to the overall number of negatively tested cases ($TN+FN$) and to the overall number of healthy cases ($TN+FP$), respectively. Another parameter is the performance coefficient that is calculated as $PC = TP/(TP+FN+FP)$ ([102] and reference therein). This parameter relates the correctly identified ill cases (TP) to all ill and all positively tested cases.

Providing sensitivity without specificity as well as vice versa has to be taken with care. Declaring all patients as ill gives a sensitivity of 100%. The same applies to the specificity, if all patients are declared healthy. In the first case specificity and in the second case sensitivity would be zero. Testing environments strive for maximizing both characteristics. However, it is very difficult to reduce the number of false positives while at the same time reducing the number of false negatives. Therefore, fixing either sensitivity or specificity and increasing the other value is an accepted approach.

For all applications in this thesis, prediction as well as evolution and clustering, sensitivity, specificity, and positive predictive value are valuable characteristics. Another characteristic is the E-value which represents the number of expected false positives for the outcome of an application. Note that in case of the prediction, the E-value is not calculated afterwards, but the application is adjusted to yield a well-defined number of false positives. Setting the E-value fixes the specificity to high values. At the same time, in order to increase sensitivity the number of predicted PREs is tried to maximize.

For some applications the specificity cannot be calculated because the number of true negatives is unknown and cannot be determined. An example is the genome-wide prediction of PREs. This is the reason, especially in the biological literature, why the PPV is often referred to as specificity. However, in this thesis, PREs are not only predicted genome-wide but also in training data. Consequently, PPV and specificity will be used as specified in Figure 2.5.

3 jPREdictor

The *jPREdictor* is a program written in Java to support the genome-wide prediction of *cis*-regulatory regions on the basis of predefined motifs. It is fast, easy to use, and allows for great versatility. The program supports the definition of a wide spectrum of motifs, basic ones as well as more complicated, i.e. assembled, ones. Many tasks can be performed using the motifs: searching sequences, weighting them on training sets, scoring sequences, and certain others, like clustering motifs. The *jPREdictor* comes either as a command-line tool, as an applet in websites, or with a graphical front end.

A number of programs exists dealing with the task of searching motifs on sequences. Schones et al. [103], for instance, developed a tool for the detection and statistical evaluation of single transcription factor binding sites and *cis*-regulatory modules (Search Tool for Occurrences of Regulatory Motifs - STORM and MODSTORM software). It is very powerful in statistical evaluation, but cannot handle certain aspects of motif types, e.g. MultiMotifs. The same applies to the MAST software (Motif Alignment & Search Tool, [48]). Another program is *PatSearch* [60], which is very powerful in defining motifs and even allows for complicated motif patterns. However, no possibility exists to assign weights to motifs. All these programs can search sequences, thus reporting a list of found motifs, which can be statistically analyzed. In addition, the *jPREdictor* can process this list of motifs by scoring a sequence.

3.1 History

The original *PREdictor* software existed for more than two years, when the decision was made in April 2005 to rewrite the underlying computer program in order to increase versatility and user-friendliness. The *PREdictor* program was developed by Marc Rehmsmeier to predict PRE/TREs in the *Drosophila melanogaster* genome, and was successfully used by Ringrose et al. [5]. For the new program the programming language of choice was Java due to excellent portability and superior support of graphical front ends. The basic functionality was preserved and built into the new *jPREdictor*, where the first letter of the name was dedicated to the programming language it was developed in. Great efforts were put into the design of the program, and finally, in February 2006, after 10 months of development, programming, debugging, and testing the first release candidate was published.

At the same time, a paper about the *jPREdictor* and a new genome-wide prediction of PRE/TREs in *D. melanogaster* was submitted to *Nucleic Acid Research* (NAR) Webserver issue 2006. When the paper, sent to NAR in the middle of February, was finally accepted at the end of March 2006, *jPREdictor* version 1.04 was released and at the time when the NAR Webserver issue was published in July 2006 [104], version 1.1 was out. This shows that the *jPREdictor* program is under constant development. Many additional features were built in, e.g. multi file support and the cut-off calculator. Major release candidate 1.2 was published in June 2007, the latest release (April 2008) has version number 1.22.

From writing the first code line until the most recent *jPREdictor* version (1.22), the program has grown significantly. It has seen some design changes and many extensions. The most recent version contains 78 Java classes and 9 interfaces. The number of code lines including documentation (in Java doc style) is about 28, 800. In order to run the *jPREdictor* program a functional Java, at least in version 1.5, is necessary. While the early versions of the *jPREdictor* were designed to run with Java version 1.4, the recent major release will run only partially with that Java version. It is therefore strongly recommended to use Java 1.5 or higher.

With release candidate 1.01 of the *jPREdictor* and with the beginning of version numbering a history was written into the program, which contains not only a complete overview over all versions so-far, but also a list of additions, bug-fixes, and changes made to the program. In the graphical front end it can be found in the *Help* section. For the command-line interface two parameters are supported, which either output the version number or the release history, respectively:

```
--version
--history
```

Always, the latest version of the *jPREdictor* can be found in the download section of the corresponding website, which is administered and updated together with the *jPREdictor* program. Besides welcome and download site it contains a manual section to explain parts of the *jPREdictor* usage. The website can be found here:

```
http://bibiserv.techfak.uni-bielefeld.de/jpredictor
```

The *jPREdictor* was well received in the scientific community. In 2006, it was used by 65 different groups (unique internet addresses according to the download statistic). In 2007, another 86 groups tried out the program.

3.2 General overview over the *jPREdictor* organization

The *jPREdictor* is organized into four main units, responsible for the basic functionality in working together smoothly. The first unit deals with representing and handling motifs, and is described in Chapter 3.3. The second unit deals with all kinds of sequences the *jPREdictor* has to work on, and is described in Chapter 3.4. An operator is built upon the motif and sequence handler to control their working, and forms the third unit together with the communicator (Chapter 3.5).

Both operator and communicator are designed for bridging the gap between the two handlers and the several user interfaces, which build the fourth unit. In Figure 3.1 this relationship is illustrated. Shaded in blue are the basic functional units, motif and sequence handler, and the part to control them, the operator. Shaded in red and yellow are the several user interfaces together with the communicator they interact with.

The *jPREdictor* comes with several different front ends. A graphical user interface is available as both, stand-alone program and applet, the latter to be started within a website. The more powerful command-line interface requires the program to be started from a terminal. Additionally, the

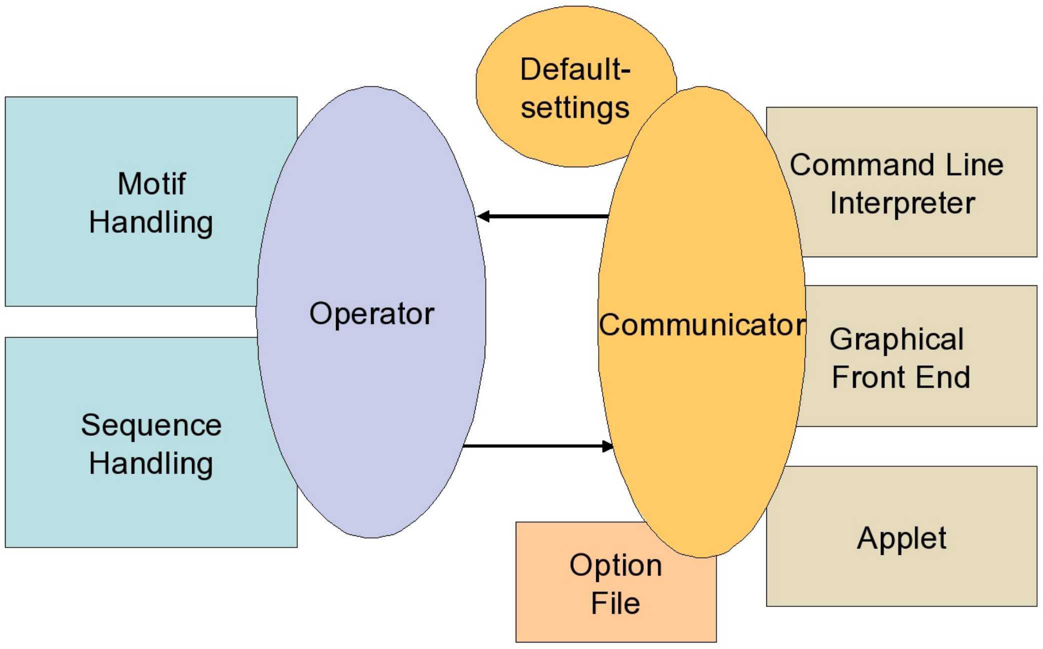


Figure 3.1: General organization of the *jPREdictor* units. Motif and sequence handler are controlled by the Operator that, together with the Communicator, bridges the gap to the various user front ends.

jPREdictor is able to recognize and evaluate option files, which allows the program to be run semi-automatically, e.g. in a computer cluster environment.

Several advantages arise from the chosen design. Every unit can be edited separately and in keeping the defined interfaces, they can be improved or updated without disrupting the basic functionality. Additionally, single units can be extended, upgraded, and tested separately. In possessing and following a clear command pipeline, i.e. operator and communicator, program errors and inconsistencies are easier to check, grasp, and correct. The *jPREdictor* is easily extendable with new units and program parts, which has proven to be a major advantage at least twice, when adding the motif evolution and the motif clustering program parts (see special task section, Chapter 3.8).

3.3 Motif representation

Motifs are the key to the functionality of the *jPREdictor*, no task is performed without defined motifs. Common to all motifs are an identifier and a description. Thus, the base class MOTIF contains getter and setter methods for these parameters. Since a motif can be reversed and complemented, the abstract method CREATEREVERSEDCOMPLEMENTED was introduced. This method is especially needed, when the motif is matched to sequences on both the plus and the minus strand. Additionally, methods for printing and representing, i.e. as a consensus string, and methods for comparing and cloning are defined.

The five supported motif types (simple sequence motif, regular expression motif, PSPM motif, PSSM motif and multi motif, see Chapter 2.2) are derived from the basic MOTIF class. Figure 3.2 illustrates these dependencies. In the class REGULAREXPRESSIONMOTIF the motif is represented in

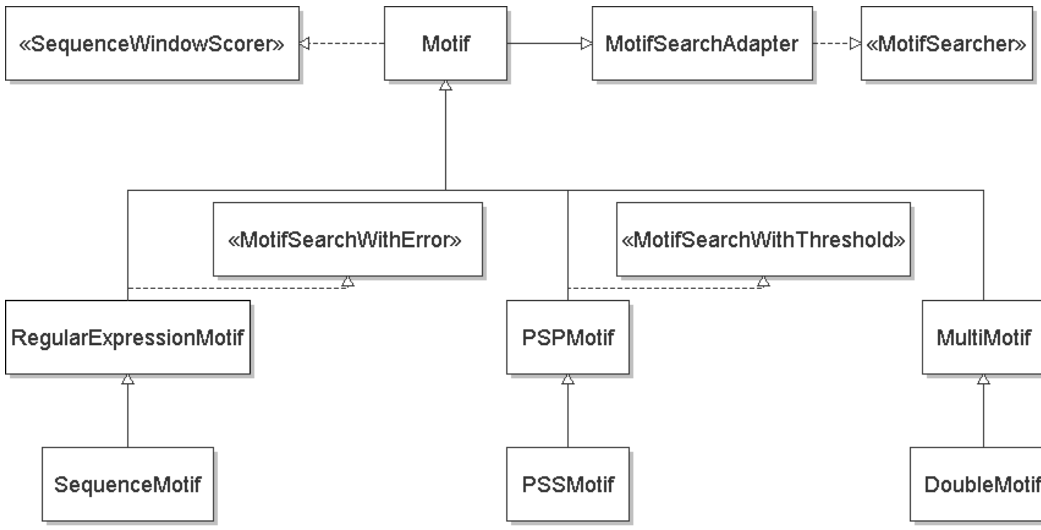


Figure 3.2: UML diagram of all supported motif types represented in the *jPREdictor*. The base class MOTIF and the derived motif types are extended by interfaces adding searching and scoring functionality. The class MULTIMOTIF can in turn contain other MOTIFs.

a STRING type, with restriction to the IUPAC one letter code. This restriction is achieved by applying a filter step to any STRING before it is accepted as a motif representation. Class SEQUENCEMOTIF in turn is derived from REGULAREXPRESSIONMOTIF, the only difference being a harsher filter step, in which the given STRING is restricted to the basic nucleotide alphabet.

The classes PSPMOTIF and PSSMOTIF (position specific probability/score matrix motif) are used for representing a motif as a matrix, either containing probabilities or log-odd-scores. This matrix is stored as an array of DOUBLE values. At every position a vector of probabilities or scores is mapped to the four nucleotides. In case both matrix motifs were created by computing an aligned set of short sequences, these sequences are stored as well. For the creation of a PSSMOTIF a vector defining the background nucleotide probabilities is necessary. This vector is stored along with the original sequences. Note that storage of all this information (sequences, background vector) other than the mere matrix is unnecessary when working with the motif, but becomes a must if the motif has to be stored back into an option file. In this case, e.g. when working with the MotifMaker in the graphical user interface, no information about how the matrix motif was originally created must get lost. The guideline is that a motif read from the file has to be equivalent to a motif written to an option file.

The class MULTIMOTIF was implemented to define and store a motif pattern composed of other motifs. Such other motifs are any of the five supported types, even of type MULTIMOTIF. Nesting MultiMotifs into other MultiMotifs allows for arbitrarily complicated motifs. They can be visualized as connected, rooted graphs, which are trees. The inner nodes are MultiMotifs and the leaves are sequence or matrix motifs. In Figure 3.3, an example is illustrated. A MultiMotif that does not contain other MultiMotifs is called flat. To define a complete motif pattern two ADD-methods are provided. The first one, ADD(MOTIF, MIN, MAX), gets a MOTIF and two distances, the minimal and maximal distance to the next motif. This ADD-method can be called as often as necessary as long as the second ADD-method is not called. By invoking the second ADD-method, ADD(MOTIF), the motif pattern is completed, since this method only takes the final MOTIF without further distance

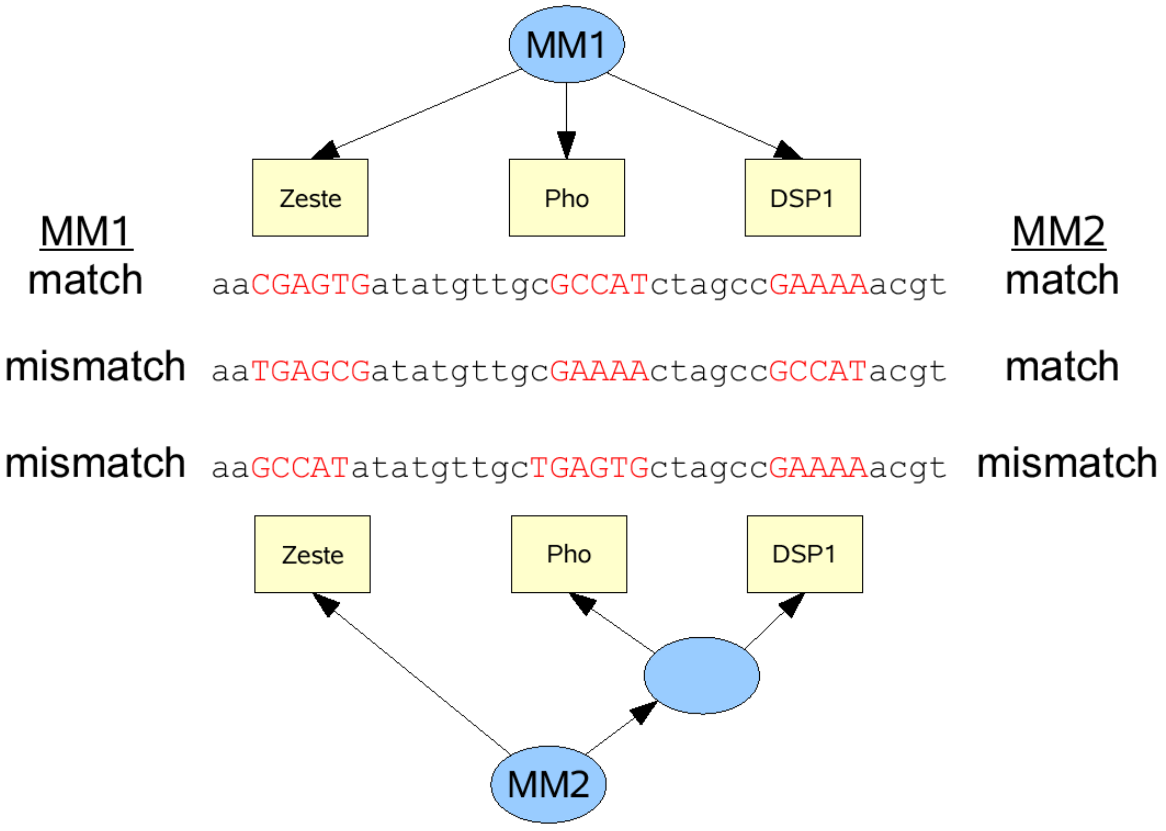

MM1
Zeste
Pho
DSP1
MM1
match
aaCGAGTGatatgttgcGCCATctagccGAAAAacgt
MM2
match
mismatch
aaTGAGCGatatgttgcGAAAActagccGCCATacgt
match
mismatch
aaGCCATatatgttgcTGAGTGctagccGAAAAacgt
mismatch
Zeste
Pho
DSP1
MM2

Figure 3.3: Example for a flat and a non-flat motif pattern both represented as MultiMotifs, and their matching behavior to three sequences with planted binding sites. MM1 and MM2 each contain the motifs Zeste (matches to *YGAGYG*), the core Pho (*GCCAT*), and DSP1 (*GAAAA*). Three sequences are drawn with the binding sites marked as red, upper-cased letters. Each sequence contains one binding site for Zeste, one for Pho, and one for DSP1 - in different orders. MM1 matches the upper sequence, but cannot match the two lower sequences, because it is flat, and Pho always has to occur in-between Zeste and DSP1. On the other hand, the non-flat MM2 matches the upper two sequences, because the motif order of the inner MultiMotif can be switched. Nevertheless, it does not match the lower sequence, because Zeste may not occur in-between Pho and DSP1.

information.

A motif pattern only stores references to the motifs it is composed of, no motif is cloned. Thus, any motif can be part of more than one MULTIMOTIF. This reusability has some advantages for functionality, especially when searching the motif on a sequence. When a motif is searched on a sequence it produces a list of match results. All MultiMotifs this motif belongs to will use this list in order to determine matches of their pattern. This means, the need for searching the motif twice or more often is eliminated. The impact of this behavior can easily be seen on the motif sets used in this thesis. The sets consist of up to 9 single motifs that are combined to up to 36 double motifs. If each motif combined in the 36 double motifs would be searched separately on the sequences, 72 of such search runs would be necessary. With the chosen approach of making a single motif part of more than one MultiMotif only 9 search runs are necessary.

While reusing motifs in more than one MultiMotif has functional advantages two problems arise. The first problem is that changing a motif will affect all motif patterns this motif is a part of. Sometimes, such a change might be intended for one pattern but unintended for another. The workaround is to clone the motif before changing it. This problem cannot occur, neither by defining motifs via option file nor by MotifMaker, because the name of every new motif has to be unique. It is only interesting to note for programmers who want to use the MOTIF classes in their programs. The second problem could occur with the *jPREdictor* prior version 1.22, but is now only interesting for programmers. It is possible, even though useless, to create a motif pattern by storing references to itself, or, if it is part of a hierarchy, to other patterns higher in the hierarchy. Such a MultiMotif would not be representable as a tree anymore, because it now contains cycles. This problem does not arise if motifs are defined via option file, because the option file is evaluated top-to-bottom. For the MotifMaker, the problem was not fixed until version 1.22 of the *jPREdictor*, when a check for cycles in motif patterns was implemented.

For all motifs but motif patterns the space consumption is $O(m)$, with m the length of the motif. MultiMotifs can be represented as an arbitrarily complicated tree, where every inner node, always a MultiMotif, has an arbitrary number of children. Such a tree may have k leaves, which are either sequence or matrix motifs. Every possible tree over k leaves can be mapped one-to-one to a binary tree, which marks the upper bound for space consumption, since $k-1$ inner nodes exist in this case. Therefore, space consumption is $O(k \cdot m + (k-1) \cdot c) = O(k \cdot m)$, with c the number of bytes per stored motif reference and corresponding distance information in a MULTIMOTIF.

The number of motifs the *jPREdictor* can theoretically handle is not limited. All motifs are stored in memory and as a disadvantage, this and the missing limit make it possible to let the program run out of memory, if too many motifs are specified. In such a case, a meaningful error message is output. With the amounts of memory available in current computers, the problem should pose no threat. In tests, all possible sequence motifs of length 8 were created and weighted, all in all, about $65,000$ motifs.

3.3.1 Matching motifs to sequences

Functionality is added to motifs by implementing certain interfaces. For searching and matching motifs on sequences, the interfaces MOTIFSEARCHER, MOTIFSEARCHWITHERROR, and MOTIFSEARCHWITHTHRESHOLD are defined. The class MOTIF extends the adapter class for the MOTIFSEARCHER interface, MOTIFSEARCHADAPTER. Note that MOTIFSEARCHADAPTER is not a stub

Zeste						Sequence						
Pattern	*A*	*C*	*G*	*A*	*G*	*T*	*G*	*T*	*G*	*T*	*G*	*C*
Y	1	0	1	1	1	0	1	0	1	0	1	0
G		2	0	2	1	2	0	2	0	2	0	2
A			3	0	3	2	3	1	3	1	3	1
G				4	0	4	2	4	1	4	1	4
Y					5	0	5	2	5	1	5	1
G						6	0	6	2	6	1	6

Figure 3.4: Shift-add algorithm for matching a pattern (here, the motif *Zeste*) to a sequence [106]. The vector at every position is recalculated from the previous one, first by shifting all values once to the end, second by adding one or zero, representing a match or mismatch at that position. A zero in the last row indicates a perfect match. In the example, only one perfect match exists starting at position 2 and ending at position 7 (shaded in blue).

(implementing an interface with methods that do nothing) like the `WindowAdapter` class for the `WindowListener` interface, but that it is a slightly distorted adapter class in the sense of design patterns [105]. It is used to pre-implement some advanced methods related to searching, i.e. the `searchAll` method, which returns all occurrences of a motif on a sequence section. Additionally, it provides a method to initialize the search, `initSearch`, and getter and setter methods for the motifs search mode, i.e. whether a motif is to be searched only on the plus strand or on the minus strand or on both representations of the given sequence. The only method the motifs themselves have to implement is `search(start, width)`, with `start` being the index on the sequence were to start the search and `width` being the width of the sequence window to search.

Sequence motifs are searched with a defined number of errors allowed in a match. Therefore, they implement the interface `MotifSearchWithError`, where the getter and setter method for the error parameter is defined. Likewise, a matrix motif is reported as a match on a sequence if a predefined threshold is exceeded. Consistently, the class `PSPMotif` implements the interface `MotifSearch-WithThreshold` defining getter and setter method for the threshold parameter.

For searching sequence motifs on a sequence, the shift-add algorithm by Baeza-Yates and Gonnet [106] was implemented. It was chosen because it is a straight forward approach without preprocessing the pattern, it is already adapted to mismatches, and can handle character sets due to a sourced-out match method. Despite a time complexity of $O(m \cdot n)$, with m and n the length of motif and sequence, respectively, the authors point out that the algorithm is about to be equally fast as an implementation of the Knuth-Morris-Pratt algorithm [107], which has a time complexity of $O(n)$. This shift-add method works by processing a vector v for every position on the sequence (denoted as s_i), which counts the number of mismatches approached so far (see Figure 3.4 for an example). The formula behind the vector reprocessing is

$$v_j^i = v_{j-1}^i + \begin{cases} 0 & p_j \simeq s_i \\ 1 & otherwise \end{cases}, \tag{3.1}$$

with j running from $m-1$ to zero, where m is the length of the motif. For $j = 0$, $v_{-1}^i = 0$. Note that in the theoretical approach v_j^i is calculated using v_{j-1}^{i-1} and not v_{j-1}^i [106]. The change was made to emphasize an implementation detail, since no vector is ever duplicated or copied from one sequence position to the next but reprocessing works in-place. Further note that the term $p_j \simeq s_i$ already implies the use of character sets, since p_j can be any character from the IUPAC one letter code. An example can be seen in Figure 3.4, where $p_0 = Y = [C, T]$ is matched to each character of the sequence. After processing the vector for a certain sequence position i, its last value states the number of mismatches occurring when matching the motif to the sequence cutout $S_{[i-m+1,i]}$. This number is then simply compared to the number of mismatches allowed, and if it is less or equal a match is reported.

Searching matrix motifs on sequences is accomplished by using a threshold. A sum-score or a product probability is calculated and if it is greater or equal to a predefined threshold a match is reported. The formula implemented into PSPMOTIF, and therefore dealing with position-specific nucleotide probabilities, is

$$S_{[i,i+k-1]} \simeq M \Rightarrow \prod_{j=1}^{k} m_{j,S_{i+j-1}} \geq threshold\,, \tag{3.2}$$

and the formula implemented into PSSMOTIF, and therefore dealing with log-odd-scores, is

$$S_{[i,i+k-1]} \simeq M \Rightarrow \sum_{j=1}^{k} m_{j,S_{i+j-1}} \geq threshold\,. \tag{3.3}$$

Look-ahead scoring [49] is used to speed up searching, which means that calculating the sum-score or the product probability is aborted, if the threshold cannot be reached anymore, even assuming the use of always the maximal values in all positions yet to come. Nevertheless, theoretical time complexity stays $O(m \cdot n)$.

Searching motif patterns was implemented into the MULTIMOTIF class by matching the distance information to the reported matches of all comprised basic motifs. The first step is that all comprised motifs report their occurrence list to the MultiMotif. Since these occurrences reported for the basic motifs are, first, in separate lists, and, second, in ascending order, a bottom-up approach over the hit lists is implemented. It works by taking the occurrences of the first motif, and then using the distance information between the first and second motif of the pattern to filter the reported matches of the second motif by discarding all occurrences not fulfilling the distance constraints. All matches of the second motif which are not discarded are processed likewise one after another, but now with respect to the third motif and the distance constraints between second and third motif. In ascending order all motifs of the pattern are processed in the same way. If approaching the last list and if at least one value remains after the filter step, a match for the MULTIMOTIF is found. Note that distance is always measured between the end of the previous and the beginning of the next motif.

Note also an advantageous implementation detail for speeding up the search. It results from the previously mentioned fact that motif patterns only store references to the motifs which are part of their pattern. Thus, even if a motif is part of more than one pattern, it is always only searched once. When working with double motifs, the advantage becomes quite obvious. Having only k single motifs results in $\binom{k}{2}$ double motifs (without self-coupling). Therefore, one basic motif is part of $k-1$ double motifs, and without this implementation detail, the search overhead would be quite large.

All search methods are implemented with respect to the search mode, which determines the orientation of a hit and which must be specified for every motif by the user. The search mode restricts the search to the plus strand, or the minus strand, or allows matches on both strand orientations. For motif patterns this might cause some non-intended behavior, because the search mode can be set individually for every motif within the pattern. Note that the order of motifs within a pattern is never disrupted. Therefore, the pattern $A-B-C$ will always be found either as $A-B-C$ (plus strand) or as $C-B-A$ (minus strand), the B motif is always located in-between A and C (see Figure 3.3). This can be overcome by creating non-flat patterns, like $A-(B-C)$. The partial pattern $B-C$ is a Multi-Motif on its own, and therefore has its individual search mode. Setting both search modes to plus as well as minus will cause four different patterns to be reported as matches, $A-(B-C)$, $A-(C-B)$, $(B-C)-A$, and $(C-B)-A$. Restricting the search mode for the inner pattern $B-C$ to plus leaves $A-(B-C)$ and $(B-C)-A$ as valid matches, and so on. But, no matter how the search mode is chosen, it is not possible to have the A motif stand between B and C.

3.3.2 Weighting motifs

Another functionality implemented into the `MOTIF` class enables it to score sequence windows. The corresponding interface is the `SEQUENCEWINDOWSCORER`, which provides a getter and setter method for the weight parameter and a method for scoring a part of the sequence, namely `SCORESEQUENCEWINDOW(START, WIDTH)`, with `START` being the index on the sequence were to start the search and `WIDTH` being the width of the sequence window. The latter method is implemented by simply multiplying the weight with the number of reported matches on the sequence.

Motif weights depict over- and under-representation in one sequence with respect to another sequence. Consequently, the weight of a motif is calculated using its occurrences on the sequences of the positive training set (denoted *pos*) versus its occurrences on the negative training set (denoted *neg*). The full formula containing all possible normalizations is given as

$$w(M) = \ln \frac{f(M\,|\,pos)}{f(M\,|\,neg)} = \ln \left(\frac{\frac{\sum_{S\in pos} \frac{fr(M|S)}{|S|} \cdot 1000 + c}{|pos|}}{\frac{\sum_{S\in neg} \frac{fr(M|S)}{|S|} \cdot 1000 + c}{|neg|}} \right), \tag{3.4}$$

with $fr(M\,|\,S)$ denoting the number of occurrences of motif M on sequence S (frequency). The first logarithm is the same as given by Ringrose et al. [5] and is stated here to emphasize the equality. Note that a training set usually comprises more than one sequence, which is the reason for the two sums.

Two normalizations are applied, first, sequence length normalization, where the number of motif occurrences is divided by the length of the sequence. Per default (as of version 1.2), this kind of normalization is switched on in the *jPREdictor* program, but can be switched off using parameter

```
-A
```

The second kind of normalization is by the number of sequences within a set. This normalization cannot be switched off in the program.

The pseudo-count value c is normally zero. But, if either sum resolves to zero, c is set to one and is added to both sums. This prevents the logarithm from becoming minus infinity.

An analysis of weights for different motifs is given in Chapter 4.4.

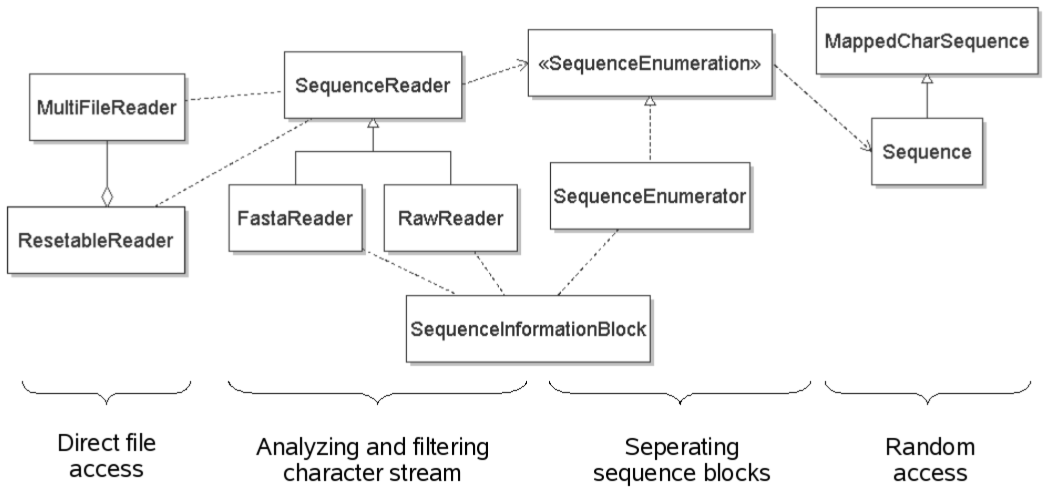


Figure 3.5: UML diagram of sequences represented in the *jPREdictor*. A file is read sequentially, its content is filtered for information and sequence blocks, which in turn are enumerated. The class MAPPEDCHARSEQUENCE allows for random access to a sequence.

3.4 Sequence representation

Sequences are used in the *jPREdictor* to match motifs against them. In contrast to motifs they are not held in memory but loaded when necessary. This reduces memory consumption to a minimum. Nevertheless, since searching motifs is done in windows on the sequence, at least this window needs to be buffered. Additionally, this sequence window must provide random access, since not only one motif after another starts searching itself at the beginning of the window, but within a motif matcher certain positions are read multiple times.

Consequently, a framework is provided within the *jPREdictor*, where sequential reads on one file or several consecutive files are mapped to a string-like structure allowing full random access. Figure 3.5 illustrates the steps needed. Within this framework, additional functionality is implemented, (1) filtering the characters read from the file to allow only valid characters to pass, (2) transforming characters, e.g. upper-casing them, and (3) analyzing the files for informative structures like headlines.

In order to provide full random access to a sequence while buffering only the last few characters read, it might become necessary to re-read a file from the beginning, i.e. performing a reset on the reader and re-open the file. This always needs to be done if a character is requested with an index position before the start of the buffer. Note that there is no class in Java providing the mentioned functionality. Therefore, RESETABLEREADER was implemented to ensure the reset functionality, which means, re-open the file if necessary and read up to a previously set mark. Additionally, the MULTIFILEREADER class was implemented to link up several files into one reader, which is processed afterwards as if it would be one big file. Again, it was ensured that reset functions properly.

Filtering and transforming functionality within the framework is provided by the SEQUENCEREADER and derived classes, FASTAREADER and RAWREADER. They filter sequence data and scan for sequence information blocks, in case of a file in FASTA format, this is any line starting with the greater-sign, “>”. Generally, in all stream data, new-lines and carriage-return characters are discarded. The raw reader does not filter the character stream any further, whereas the FASTA reader upper-cases all characters read and passes only the ones defined in the IUPAC one letter code. In order to store sequence information for every separate sequence an instance of class SEQUENCEINFORMATION-BLOCK is created and written. The raw reader does not know separate sequences and therefore writes only one such block. The FASTA reader starts writing a sequence information block whenever a line beginning with the greater-than character, ">", is approached.

The software design was chosen to ensure easy extendability. Additional filters for sequence formats can be included by deriving the SEQUENCEREADER class, and overwriting the READ-FILTERED method.

The information stored in the blocks is evaluated, when all sequences are enumerated one after another. For this, the SEQUENCEENUMERATION interface was implemented by the SEQUENCE-ENUMERATOR class, its only method, NEXTSEQUENCE, returning instances of the SEQUENCE class (see Figure 3.5). This ensures that dependent classes can correctly count number and length of single sequences. These characteristics are used for instance for weight calculation and normalization.

Each separated sequence is then mapped to a character sequence, providing random access to its characters. For this, the MAPPEDCHARSEQUENCE buffers a certain amount of characters and if a character is requested from a position before the start of the buffer a reset is performed on the sequence ensuring that the requested character comes into focus once more. The size of the buffer, the *jPREdictor* is working with, is chosen as 64 kB.

The framework can handle files of sizes up to 16 Million TB (tera bytes) due to 64 bit file pointers. Nevertheless, character sequences supported by Java are restricted to 32 bits, therefore the size is limited to four GB. As a consequence, files may be of any size, while each separate (FASTA) sequence is limited to four GB.

3.5 Communicator and Operator - Processing pipeline

Both communicator and operator make up the third unit in the *jPREdictor* organization. They were designed to bridge the gap between the user interfaces and the motif and sequence handlers (Figure 3.1). The advantage of separating user interfaces and the units performing the basic functions is that the definition of higher-order tasks or applications built upon the basic functions is possible, without the need for the user interfaces to control their execution.

The communicator itself has one major task to perform, the storage of all possible parameters, such as the size of the sequence window and the list of motifs. The default settings are stored in the STATES class. By extending the STATES class and by implementing the ICOMMUNICATOR interface, which defines getter and setter methods and other accessible methods and objects, the DEFAULTCOMMUNICATOR class can fulfill the storage task.

The operator was designed for initializing and starting the threads dealing with the main functionality, which is searching motifs on sequences, weighting motifs by searching them on the sequences of the positive and negative training set, and scoring a sequence window-by-window. Threads run concurrently to the main program and allow for in-time and more precise control of their execution. The operator consists of an interface, IOPERATOR, and the implementing class, OPERATOR. In order to initialize and start the threads, a valid communicator must have been provided previously, containing all necessary parameters.

The chain of command starts with the user, which tells the program to perform a certain task (see

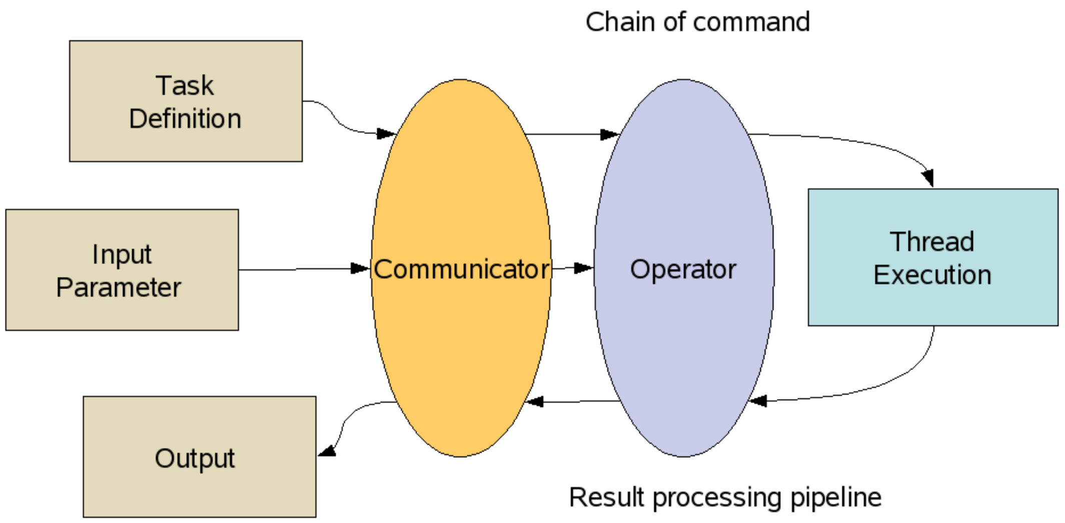


Figure 3.6: Chain of command and result processing pipeline in the *jPREdictor* program. The user defines the task and provides the necessary parameters to the communicator, which orders the operator to start a thread. Results are preprocessed in the operator, then sent to and stored in the communicator. After creating the final output format the user interface receives the results.

Figure 3.6). The user interfaces instruct the communicator, which orders the operator to start the corresponding thread. Thread control is done by the communicator, in response to or after consulting the user. At the moment the thread is started it produces results. Results are passed in reverse order along the chain of command. They are first evaluated and preprocessed by the operator in accordance to the started task. Afterwards, they are sent to the communicator, which stores them and may process them further to a format the user interface expects.

3.6 User interfaces

Interfaces reflect the tasks a program is capable of performing to the user. The basic task the *jPREdictor* is capable of is the search of motifs on sequences. Additionally, as an extension to the search task, the *jPREdictor* assigns weights to motifs by recalculating the results from searching the sequences of positive and negative training set. Scoring sequences, the third major task, combines search results from weighted motifs to score a sequence window.

Tasks have parameter settings, which may change them to a degree that they seem to be different tasks. All these parameters need representations in the user interfaces.

Note that both interfaces, graphical and command-line, are not equally powerful. The motif maker and the cut-off calculator can only be found in the graphical interface. Nevertheless, their functionality can be mimicked in the command-line interface, which, by giving more options and parameters, makes the task more adjustable and configurable.

On the other hand, many specialized tasks can only be started via command line, and have no counterpart in the graphical interface, i.e. the calculation of PSSM occurrence probabilities.

3.6.1 Graphical user interfaces

The graphical user interface was implemented to give an easy access point to the user, and to make working with the *jPREdictor* more comfortable. All settings and parameters can be found here, the tasks are easily started by just pressing the corresponding action buttons.

The graphical part of the *jPREdictor* is started either as a stand alone program or within a website, as a Java Applet or as a Java WebStart application. Both Applet and WebStart application are implemented into the *jPREdictor* website:

```
http://bibiserv.techfak.uni-bielefeld.de/jpredictor
```

To start the program with the graphical user interface simply type

```
java -jar jPREdictor.jar
```

into the terminal. Prior to version 1.2 this was the only way to get the GUI started, and no parameters were allowed at start-up. Now, with version 1.2 and newer, parameters were introduced to force the *jPREdictor* into certain actions. One of these actions is to start the GUI regardless of other settings. It is done with:

```
java -jar jPREdictor.jar [parameters] --forceGUI [parameters]
```

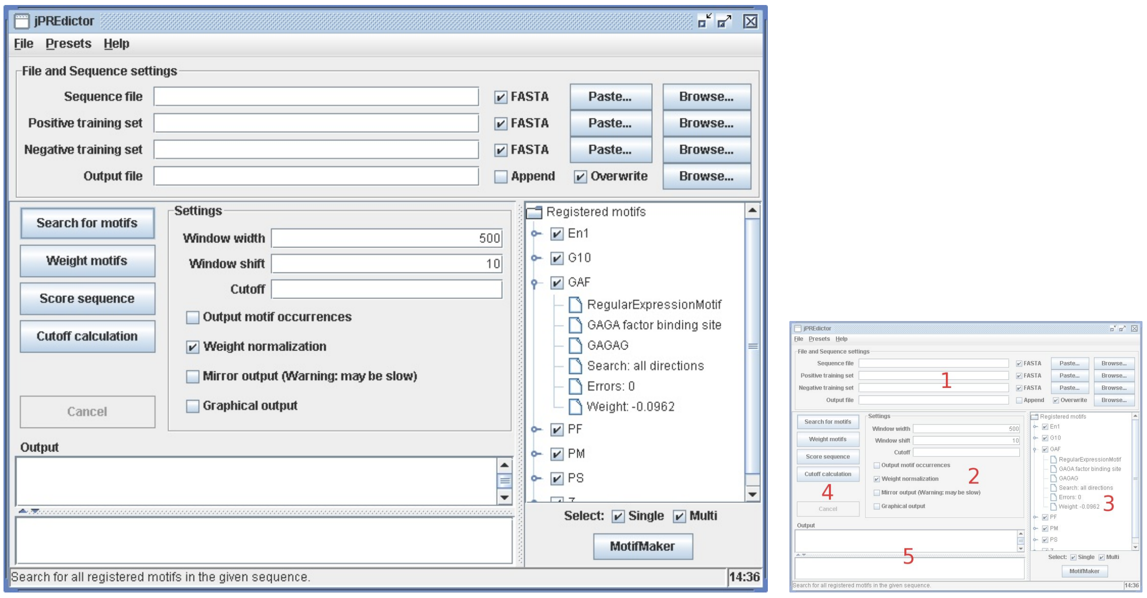
The *jPREdictor* program will first evaluate all other parameters given and then start the graphical user interface. Note that all other force parameters are ignored.

3.6.1.1 Main window

The main window of the *jPREdictor* is organized into sections, each corresponding to different components related to each other (Figure 3.7). In Section 1 all components are related to file input and output, namely the sequence files to search or score, the positive and negative training sets, and the output file. The files may be browsed by pressing the corresponding buttons or by directly typing them in. As an alternative, sequence data can be given directly by pressing the “Paste...” button.

Section 2 contains additional parameters and settings (Figure 3.7). Both window parameters are necessary for scoring a sequence, since the score will be calculated within a window that slides over the sequence. The width is defined in number of nucleotides, e.g. 500. The shift value denotes the number of nucleotides lying between the starts of two consecutive windows, e.g. the first window starts at position 700 on the sequence, the second at 710, which would mean a shift value of 10. The cut-off value is a parameter which changes the output of the score sequence task. If it is omitted, the scoring prints the score for every possible sequence window. If it is specified, only windows with scores higher than the cut-off are output after an additional step, which combines overlapping windows to so-called bands.

Four checkboxes can be found in Section 2 (Figure 3.7). The first checkbox, “Output motif occurrences”, is available for the “Search for motifs” task. Normally, searching motifs does not return the



Window width 500
Window shift 10
[✓ FASTA] [✓ Overwrite] [✓ Weight normalization] [✓ Single] [✓ Multi]

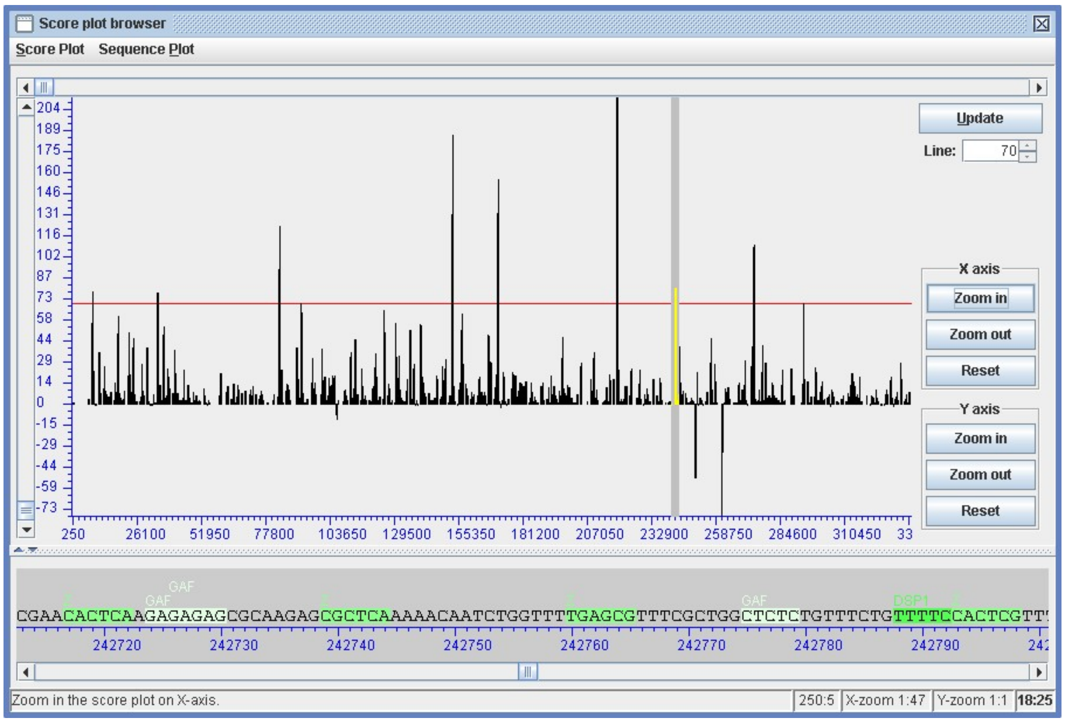
Search for all registered motifs in the given sequence. 14:36

Figure 3.7: Main window of the *jPREdictor* program. Section 1 and 2 contain certain settings and parameters, Section 3 lists all usable motifs in a tree structure, Section 4 contains the action buttons for starting tasks and Section 5 has two output fields, the upper for “normal” output and the lower for “error” messages.

single hits but the number of found motifs per sequence. With this checkbox activated, every and all single hits are printed. The second checkbox, “Weight normalization” enables or disables sequence length normalization while calculating the weights (see Chapter 3.3.2 for an explanation). This checkbox is related to the “Weight motifs” task. The last two checkboxes exist to redirect the output. The first one, “Mirror output”, prints all the text, which is normally sent to the output file given in Section 1, to the output Section 5. If much output is produced this may rapidly slow down the *jPREdictor*, since the text fields supported by Java are not meant for very large amounts of text. The graphical output available by checking the last box in Section 2 will only affect the “Score sequence” task. After starting the task a window will pop up, the *ScorePlotBrowser*, showing a graph over the scores (see Figure 3.8).

In Section 3 (Figure 3.7) the list of motifs is shown to be used to search and score with. Motifs can be checked and unchecked to include or exclude them from being used in the tasks. This list can be filled and edited either by loading an option file via menu, **File→Load...**, by selecting a predefined set of motifs from the menu **Presets**, or by comfortably designing motifs using the available *MotifMaker*. Together with most parameters and settings, the list of motifs can be saved to an option file via the **File→Save...** menu.

In Section 4 (Figure 3.7) the buttons to start certain tasks are located. By pressing the first button, “Search for motifs”, the *jPREdictor* searches the checked motifs in the given sequence file or in the pasted sequence, and outputs either for every motif and every given sequence the number of motifs found, or, if the box “Output motif occurrences” is checked, returns a exhaustive list of all motif occurrences. Pressing the second button, “Weight motifs”, performs a search in both positive and negative set for all checked motifs. As a result, the motifs are assigned weights as explained in Chapter 3.3.2. If they were already weighted, these weights are discarded. By pressing the third



CGAACACTCAAGAGAGAGCGCAAGAGCGCTCAAAAACAATCTGGTTTTGAGCGTTTCGCTGGCTCTCTGTTTCTGTTTTCCACTCGTT

Zoom in the score plot on X-axis.

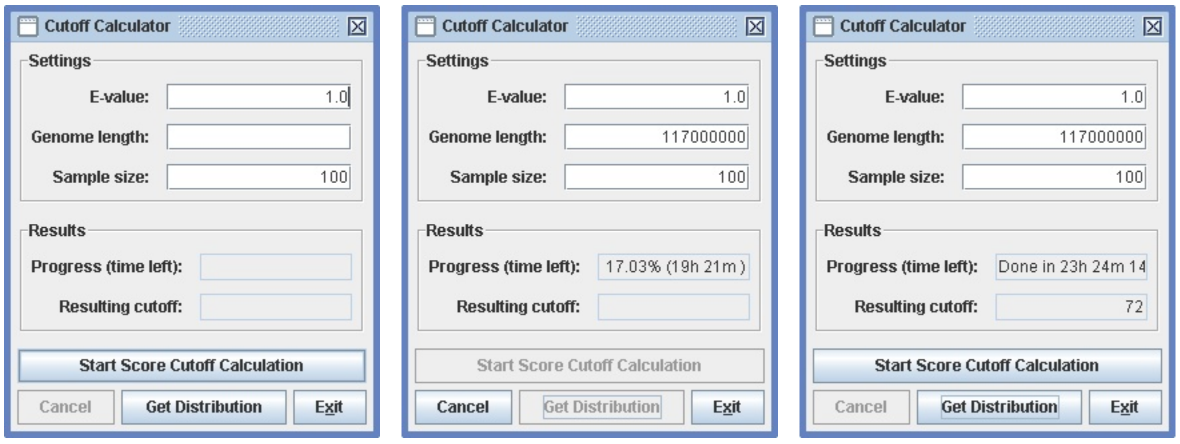
Figure 3.8: Example score plot over the bithorax complex of *D. melanogaster*. The grey vertical bar marks the position of the sequence cutout, which can be seen in the lower section of the window, and which shows single motifs on the sequence highlighted in green colors.

button, “Score sequence”, the task to score a sequence window-by-window is started. Note that all checked motifs must have been weighted previously, otherwise an error message will pop up. These three tasks can be canceled at any time by pressing the “Cancel” button, which is only available if a task really was started. Another commonality between these three tasks is that they send their status and error messages to the output section, numbered as 5 in Figure 3.7.

3.6.1.2 Cut-off calculator

In Section 4, a fourth button exists, the one to start a cut-off calculation via the CutoffCalculator (see Figure 3.9). The cut-off calculator works by randomly generating a large amount of sequences. A zeroth order Markov chain model is used by the underlying generator. The vector of nucleotide distribution is either the program-wide one or it is calculated by counting the number of nucleotides in the sequence file given in the main window (Section 1 in Figure 3.7). This counting is started by pressing the “Get Distribution”-button. The number of sequences to be generated is given in the text field “Sample Size” (Figure 3.9). The length of each generated sequence can be given in the text field “Genome length”.

In order to get a valid cut-off all sequences are scored and the number of windows with scores above each possible cut-off is counted. The motifs needed for the scoring are the ones set in the main window of the *jPREdictor*. As a result a window pops up with the optimal cut-off for every E-value. When scoring the real genome, the E-value is the number of sequence windows above the corresponding cut-off to be expected merely by chance.


Cutoff Calculator

Settings
E-value: 1.0
Genome length:
Sample size: 100

Results
Progress (time left):
Resulting cutoff:

Start Score Cutoff Calculation
Cancel
Get Distribution
Exit

Cutoff Calculator

Settings
E-value: 1.0
Genome length: 117000000
Sample size: 100

Results
Progress (time left): 17.03% (19h 21m)
Resulting cutoff:

Start Score Cutoff Calculation
Cancel
Get Distribution
Exit

Cutoff Calculator

Settings
E-value: 1.0
Genome length: 117000000
Sample size: 100

Results
Progress (time left): Done in 23h 24m 14
Resulting cutoff: 72

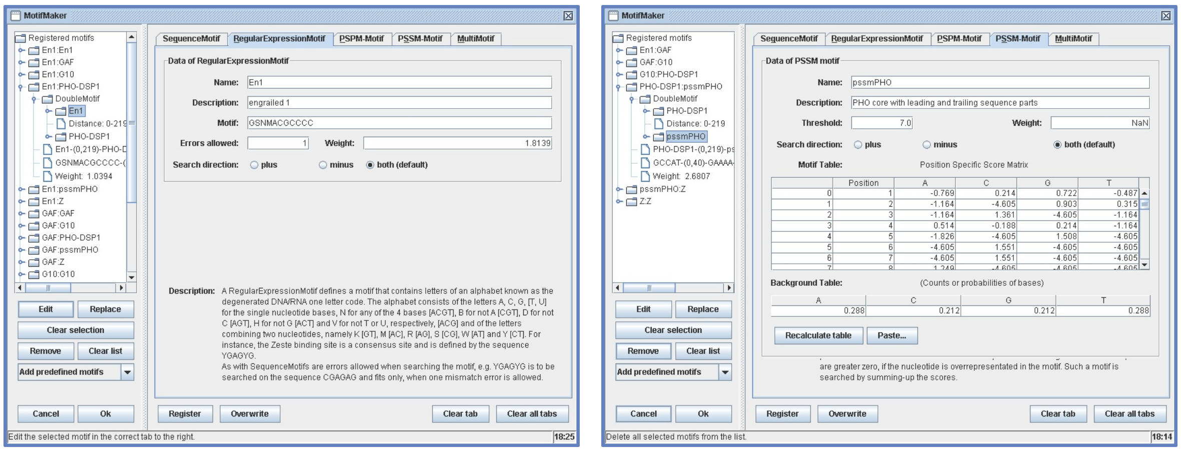
Figure 3.9: The CutoffCalculator built into the *jPREdictor*. Performs a cut-off calculation by scoring randomly generated sequences, using the motifs from the main window of the *jPREdictor*. After start-up (left), while performing a calculation (middle), and after finishing the calculation (right).

3.6.1.3 Motif maker

The motif maker is started via pressing the button “MotifMaker” in Section 3 of the *jPREdictor* main window (Figure 3.7). It allows for easy creation and editing of motifs and for re-arranging the list of used motifs. In Figure 3.10 two windows with different activated tabs are shown, on the left for editing a regular expression motif, and on the right for editing a matrix motif. All attributes and characteristics of a motif are accessible for change, e.g. identifier (name), description, weight, and search direction. In order to get both *MotifMaker* windows, select **Presets→New (2006) PRE/TRE prediction on D. melanogaster** in the *jPREdictor*’s main window, followed by clicking the “MotifMaker”-button. Within the *MotifMaker*, expand the tree to the left until an En1 or a pssmPHO, respectively, is selectable. Press “Edit” to automatically switch to the tab necessary for displaying all attributes of the motif. For the pssmPHO motif, it is possible to see all sequences the matrix was created from by clicking the “Paste...”-button.

After finishing all changes the newly created motif can either be a replacement for selected motifs from the tree (button “Overwrite”), or it can simply be registered as a new motif (button “Register”). Note that the name acts as unique identifier, and no duplicate names are allowed.

In Figure 3.11 the tab to create and edit motif patterns, also called MultiMotifs, is shown. Since MultiMotifs might become very complicated, the usual procedure starts with creating simple, flat MultiMotifs, which are afterwards combined to higher-order MultiMotifs of any complexity. The intermediate motifs are all stored in the list of motifs to the left, and have to be deleted after the assembly process. To insert distance constraints between the motifs, the “Add”-button has to be pressed after the minimal and maximal distance was typed in. Note that in order to append a distance constraint or a motif at the end, the insert policy requires that nothing is selected. If a selection is made, the constraint or motif is placed before it.



Name: En1
Description: engrailed 1
Motif: GSNMACGCCCC
Errors allowed: 1 Weight: 1.8139
Search direction: plus minus [✓ both (default)]

Description: A RegularExpressionMotif defines a motif that contains letters of an alphabet known as the degenerated DNA/RNA one letter code. The alphabet consists of the letters A, C, G, [T, U] for the single nucleotide bases, N for any of the 4 bases [ACGT], B for not A [CGT], D for not C [AGT], H for not G [ACT] and V for not T or U, respectively, [ACG] and of the letters combining two nucleotides, namely K [GT], M [AC], R [AG], S [CG], W [AT] and Y [CT]. For instance, the Zeste binding site is a consensus site and is defined by the sequence YGAGYG.
As with SequenceMotifs are errors allowed when searching the motif, e.g. YGAGYG is to be searched on the sequence CGAGAG and fits only, when one mismatch error is allowed.

Edit the selected motif in the correct tab to the right.

Name: pssmPHO
Description: PHO core with leading and trailing sequence parts
Threshold: 7.0 Weight: NaN
Search direction: plus minus [✓ both (default)]
Motif Table: Position Specific Score Matrix

	Position	A	C	G	T
0	1	-0.769	0.214	0.722	-0.487
1	2	-1.164	-4.605	0.903	0.315
2	3	-1.164	1.361	-4.605	-1.164
3	4	0.514	-0.188	0.214	-1.164
4	5	-1.826	-4.605	1.508	-4.605
5	6	-4.605	1.551	-4.605	-4.605
6	7	-4.605	1.551	-4.605	-4.605
7	8	1.249	-4.605	-4.605	-4.605

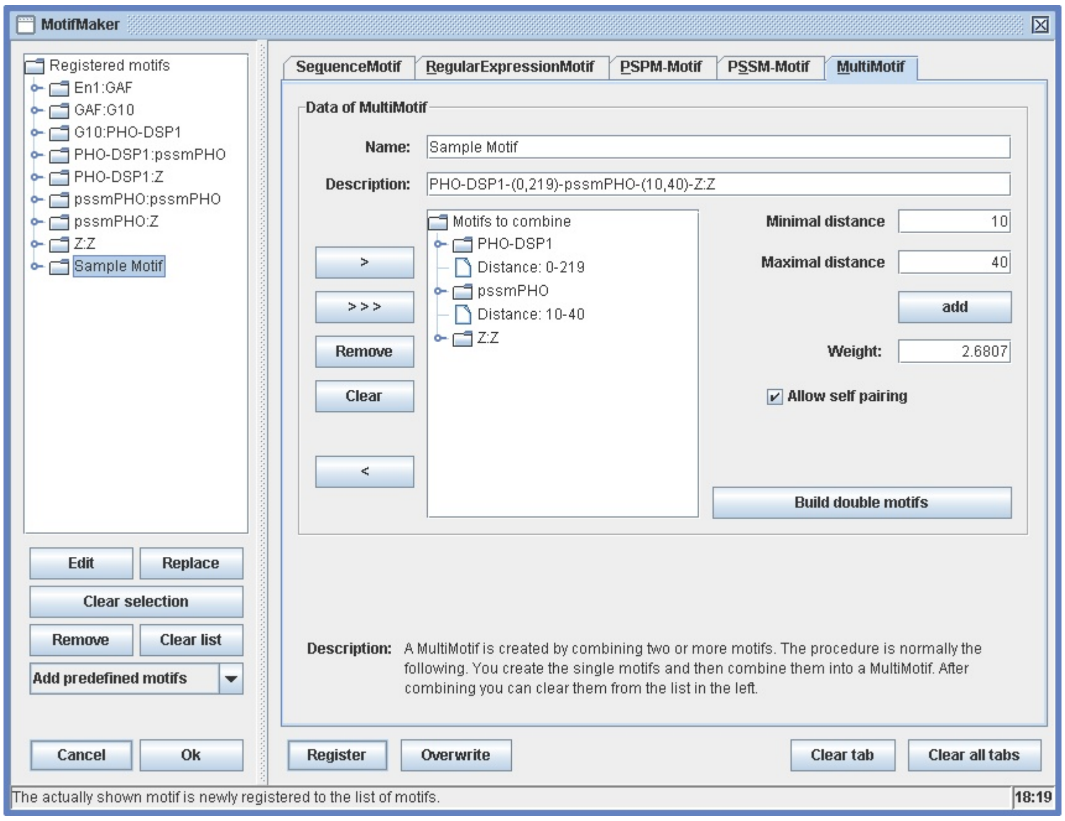
Background Table: (Counts or probabilities of bases)

A	C	G	T
0.288	0.212	0.212	0.288

are greater zero, if the nucleotide is overrepresentated in the motif. Such a motif is searched by summing-up the scores.

Delete all selected motifs from the list.

Figure 3.10: The MotifMaker with two out of five tabs for editing a RegularExpressionMotif and a PSSM motif. In the left picture, the regular expression motif Engrailed, and in the right picture, the position specific score matrix for the Pho motif is shown.



Name: Sample Motif
Description: PHO-DSP1-(0,219)-pssmPHO-(10,40)-Z:Z
Minimal distance: 10
Maximal distance: 40
Weight: 2.6807
[✓ Allow self pairing]

Description: A MultiMotif is created by combining two or more motifs. The procedure is normally the following. You create the single motifs and then combine them into a MultiMotif. After combining you can clear them from the list in the left.

The actually shown motif is newly registered to the list of motifs.

Figure 3.11: The MotifMaker with the selected tab for creating and editing motif patterns, so called MultiMotifs. A non-flat sample motif was created containing one MultiMotif, PHO-DSP1, followed by a matrix motif, pssmPHO, and terminated by another MultiMotif, Z:Z.

3.6.2 Command-line interface

For the command-line interface to be available, the user has to start the *jPREdictor* program within a terminal. Many parameters and options are recognized by the program, the typical call looks like this:

```
java -jar jPREdictor.jar [parameters]
```

Simple parameters start with a single dash (minus sign) followed by a letter. Many parameters that contribute to an extended functionality start with a double-dash, followed by a word. Some parameters need arguments to be specified as well. Omitting these arguments, or using non-recognized parameters will result in meaningful error messages. The *jPREdictor* provides several help pages for using the command-line interface and certain parameters. The typical call for help uses one of these parameters:

```
-?
-h[o|s]
```

Parameter "–?" and "–h" behave the same and print the help explaining how to use the *jPREdictor* via command-line interface. Also, most simple parameters and options are listed and explained. Parameter "–hs" prints help on specialized tasks the *jPREdictor* is capable of performing, while parameter "–ho" prints a help on how to write an option file. Another helpful parameter is "–v" which enables meaningful verbose messages to be printed along the normal output.

The following example shows a standard call. The *jPREdictor* is instructed to score a sequence file. The motifs used are the built-in ones. They are weighted using two other sequence files (given in FASTA format), model (positive training set) and background (negative training set).

```
java -jar jPREdictor.jar -m model,FASTA -b background -f sequ_file
```

Three parameters are available to provide sequences to the *jPREdictor*. The arguments for the first parameter, "–f", are files holding sequences to be searched for motifs or to be scored. This parameter can be given arbitrary many arguments (filenames). Additionally, it is possible to provide a dash (minus), which indicates standard input. Thus, the *jPREdictor* is able to directly process data coming from the preceding process in the pipe, which may be a sequence-generating program. An example can be seen here, which produces 1 as an output:

```
echo "ACGTAAAACC" | java -jar jPREdictor.jar -d AAAA -f - -p single
```

The other two parameters for providing sequences are "–m" and "–b". Both have to be given together or have to be left out together, because their arguments name the files to be used as positive (model) or negative (background) training set. Note that if only one file is given to both, the *jPREdictor* interprets its content as a table of motif occurrences (unless the filenames are appended a sequence-format identifier). This table can be produced with the *jPREdictor* by searching for motifs using default settings.

The *jPREdictor* normally guesses the tasks to perform from the given sequence parameters. If only "–f" is specified, without "–m" and "–b", the search-for-motifs task is started. Omitting "–f" but giving "–m" and "–b" will start the weighting of motifs. By specifying all three sequence parameters, first, the weighting is carried out, followed by scoring a sequence. Nevertheless, there may be situations, where the guessing should be switched off, for instance, if all motifs given, i.e. via option file, are already weighted. In this case a weighting would be unnecessary, and only the scoring of a sequence should be performed. For these situation, parameters to force the *jPREdictor* into certain actions were included in version 1.2, i.e. "—forceGUI" (already discussed in Chapter 3.6.1), "—forceSearchMotifs", "—forceWeightMotifs", and "—forceScoreSequence". They do exactly as their names suggest. Note that forcing a task will disable all task-guessing.

Via command line, only regular expression motifs and rudimentary motif patterns can be specified. Regular expression motifs are defined using the parameter "–d" with an option. For every motif to be defined, another parameter "–d" must be provided. Options for this parameter are the motif representation as a string, the name of the motif, the number of errors allowed for a match, and the search direction. For example, defining the engrailed motif with one error allowed and to be matched only reversed-complementary would look like this:

```
-d GSNMACGCCCC,engrailed,1,minus
```

Creating MultiMotifs is done automatically by the *jPREdictor*. Without the help of an option file or the graphical user interface, creation is restricted to double motifs. The option "–p" specifies the distance between each two motifs. The default option for "–p" is "BELOW_220", which means that double motifs are created from the given basic motifs with a minimal distance of zero and a maximal distance of 219. It is very important to note that the automatic creation of double motifs is done in every case, where no MultiMotif is specified. And in some other cases, too. To switch it off, use parameter "–p" with option "single". In this example two motifs are defined, *AAAA* and $TTTT$, either as singles or as double motifs.

```
-d AAAA,polyA -d TTTT,polyT -p single
-d AAAA,polyA -d TTTT,polyT -p BELOW_220
```

The first line yields two motifs, which are used for searching and scoring, namely *polyA* and *polyT*. The second example yields 3 double motifs, $polyA-(0,219)-polyA$, $polyA-(0,219)-polyT$, and $polyT-(0,219)-polyT$.

Most other parameters are available for fine-tuning the tasks of the *jPREdictor*. The task of searching motifs is altered by parameter "–t", which makes sure that individual motif positions are printed instead of numbers on how often the motifs occur. Parameter "–a" and "–A" switch on (default) and off, respectively, the sequence length normalization while weighting motifs. Finally, scoring a sequence is influenced by the arguments given to parameter "–w" and "–s", which state width and shift value of the sliding sequence window. Normally, scoring prints the score for every window on the sequence. Nevertheless, this behavior is altered by parameter "–u", which, if given, would cause the scoring process to output only the window with the highest score, and also by parameter "–c", whose argument states a cut-off, and only bands above that cut-off are output. Note that bands are sequence windows merged due to overlapping.

The last parameter to discuss is "–o", which is used to load an option file while starting the *jPREdictor*. The option file is always evaluated after all other parameters were recognized. This is important to note, because it is often the reason behind error messages and unintended behavior.

3.7 Option file

The capability of recognizing option files was included into the *jPREdictor*, first, for providing a way to store different sets of settings, and second, to make complicated motifs available to the command-line interface. This makes option files very helpful and handy when running the program semi-automatically. Additionally, they can be easily created on the fly due to the intuitive language.

Option files are very powerful tools. They allow for precise and comprehensive creation of motifs and motif patterns. Additionally, storing complete settings into option files makes working easily retraceable and therefore repeatable. The option file is the interlocking part of the *jPREdictor*, bringing the different front ends together. For example, an option file is created in the GUI when the settings are saved via **File→Save...**, and this option file can be read via parameter "–o" in the command line.

An option file is separated into sections; each section starts with a keyword in brackets. Every section may occur more than once, if need be. Recognized keywords are "Sequence", "Motif", "MultiMotif", and "MultiMotifList". Within a section, special parameters can be specified, always followed by an equal sign and the argument. In the following example a motif section is started, wherein the name of the motif, its representation and finally the regular expression sequence itself are specified:

```
[Motif]
name=SP1/KLF
motif=SEQUENCE
RRGGYG
```

Within the section "Sequence", filenames for the sequence to score as well as positive and negative training set can be given. Additionally, settings already known from the command-line interpreter may be specified, namely "windowWidth", "windowShift", "sequenceLengthNormalization", "cutoff", and "global_background". Except the last, the parameters are self-explanatory, or can be looked up in the help text (produced by starting the *jPREdictor* with parameter "–ho"). The global background specifies a nucleotide distribution, which is either used when generating a PSSM motif from a probability matrix, or to generate sequences under a zeroth order Markov chain. The latter is performed by the CutoffCalculator in the graphical front end (see Chapter 3.6.1.2), or when the probabilities for a PSSM to reach certain thresholds are calculated (see Chapter 3.8).

All other sections are dedicated to motif generation. In section "Motif", sequence and matrix motifs can be specified. The parameters "name", "description", "weight", and "searchDirection" are common to both types. The "motif" parameter knows four arguments, "SEQUENCE", "MULTI_SEQUENCE", "TABLE_PROB", and "TABLE_SCORE". As a result from using the first argument a regular expression motif is created. The second and third argument yield a position specific probability matrix (in the first place), and when using the last argument, the complete score-matrix has to be given representing a PSSM. The sequences or the table, respectively, follow right after the line with the “motif” parameter.

The next examples define the Zeste motif, either as an aligned set of short binding sites or directly as a table. While the first example results in a regular expression sequence, namely *YGAGYG*, the other two examples are interchangeable in the sense that they both yield a probability table in a PSPM motif:

```
[Motif]
name=Zeste
motif=SEQUENCE
CGAGCG
CGAGTG
TGAGCG
TGAGTG
```

```
[Motif]
name=Zeste
motif=MULTI_SEQUENCE
CGAGCG
CGAGTG
TGAGCG
TGAGTG
```

```
[Motif]
name=Zeste
motif=TABLE_PROB
 1   0   2   0   2
 2   0   0   4   0
 3   4   0   0   0
 4   0   0   4   0
 5   0   2   0   2
 6   0   0   4   0
```

Note that the table provided when using “TABLE_PROB” need not contain probabilities. Nucleotide counts are also allowed, since the table is recalculated to probabilities. Defining a motif by using “MULTI_SEQUENCE” and “TABLE_PROB” will result in a PSPM in the first place. Absence or presence of the parameter “background” decides, whether the probability table is recalculated to a score table, to be used in a PSSM. The implemented equation behind the recalculation can be found in Chapter 2.2.2. The pseudo-count value is set to $c = 0.01$.

For working with sequence and regular expression motifs the error number allowed for a match has to specified. Consequently, the parameter’s name is “errorNumberAllowedForMatch”. Likewise, matching PSPMs and PSSMs on sequences requires a threshold. Both, error number and threshold, are optional and therefore have a defined default value, which is zero for the number of matching errors, and the highest reachable value for the threshold.

For creating motif patterns, the two sections “MultiMotif” and “MultiMotifList” are available. In the “MultiMotif” section, the pattern is created step-by-step. The underlying parser of the “MultiMotifList” mimics in some detail the creation of MultiMotifs as known from the command-line interface. In both sections motif names are used to identify motifs. It is therefore mandatory that they are unique. A motif pattern is created by giving the first motif after the parameter “first_motif”, followed by one or more subsequent motifs together with the distance information as arguments to the parameter “next_motif”. The next example creates the flat MultiMotif from Figure 3.3 on Page 39:

```
[MultiMotif]
name=MM1
first_motif=Zeste
next_motif=0,220,Pho
next_motif=0,40,DSP1
```

Note that for the above lines to function all three motifs, Zeste, Pho, and DSP1, have to be specified before, i.e. with three “Motif” sections. Only one motif pattern can be specified within a “MultiMotif” section. On the contrary, a “MultiMotifList” section is used to describe rules for automatic creation of motif patterns. Only one parameter is recognized, “distance”, which has minimum and maximum value as arguments and which is used between every two motifs in all motif patterns to be created within this section. The distance is followed by one or more lines with motif-pattern specifications, each consisting of a list of motif names either separated by colon or comma. In the next example two motifs similar to `MM1` from above are created:

```
[MultiMotifList]
distance=0,220
Zeste:Pho:DSP1
Zeste:DSP1:Pho
```

Note that the created motifs are given names which reflect the composition. The two names assigned in the example would be $Zeste-(0,220)-Pho-(0,220)-DSP1$ and $Zeste-(0,220)-DSP1-(0,220)-Pho$. The separator used above is the colon. Replacing the colon by a comma starts the automatic creation of double motifs, separately for each line. Since in the above example both lines list the same motifs, the same double motifs would be created. Additionally, the created motifs would have been assigned the same names. But because this violates the uniqueness-of-names rule, double motif creation for the second line is aborted after the first encountered name conflict. The adjusted example looks like this:

```
[MultiMotifList]
distance=0,220
Zeste,Pho,DSP1
```

Here, six double motifs are created, namely $Zeste-(0,220)-Zeste$, $Zeste-(0,220)-Pho$, $Zeste-(0,220)-DSP1$, $Pho-(0,220)-Pho$, $Pho-(0,220)-DSP1$, $DSP1-(0,220)-DSP1$. Every motif is paired with itself, afterwards with all residual motifs, and finally removed from the list. If the list initially contains n motifs, all-in-all $\binom{n}{2}+n$ double motifs are created.

3.8 Special tasks

In this chapter, parameters are listed, which do not belong to the original functionality of the *jPREdictor*, but which proved useful for the work with motifs and scores. They are necessary for re-processing results or provide motif statistics. Note that these parameters have to be given in the command-line interface, and that they are not part of the graphical front end. Further note that they cannot be used stand-alone, but only in combination with other parameters. This also was a reason for embedding them into the *jPREdictor*.

The *jPREdictor* is able to calculate a complete p-value distribution for any given PSSM (background and details in Chapter 2.2.2). The parameter for this is

```
--pssmProbs
```

The PSSM motifs must be provided to the *jPREdictor* via option file. The program will then output the probabilities for every PSSM to exceed specific thresholds in the range from minimum to maximum sum-score in steps of 0.1. Figure 2.3 on Page 18 was created using the output of such a calculation performed for the Pho PSSM.

For calculating the probability of every threshold, the *jPREdictor* uses the full enumeration approach under a zeroth order Markov chain. It is implemented recursively by adding scores and multiplying nucleotide probabilities for every position starting at the end of the matrix. Look-ahead scoring in both directions (as described in [49]) is used to speed up the calculation. This way, computing the *p*-value for low thresholds (therefore many matches) may break earlier if the threshold can be reached with any partial sequence yet to come. On the other hand, for high thresholds, computation may also break earlier if the threshold at hand cannot be exceeded anymore. The background distribution over the single nucleotides used for the calculation is either the default one, or provided via option file.

One special parameter was implemented into the *jPREdictor* dealing with the task of calculating cut-offs. Other than the "—pssmProbs" parameter, the cut-off calculation has its counterpart in the graphical front end. To start such a calculation via command line use this parameter:

```
--cutoffCalc
```

It will perform a cut-off calculation by directly invoking the scoring of a sequence (for which the parameters must be given, too). The scores for every window on the sequence are therefore not output, as they normally would, but used in the calculation process. Note that overlapping windows are combined to bands if their corresponding scores exceed the defined cut-off. As a result, bands are maximal in width and do not overlap each other. However, this is also the reason, why very small cut-off values yield less bands than mediocre cut-offs. The smallest cut-off will yield only one band, since all sequence windows have higher scores. If more than one sequence is provided for the calculation, it is combined to one big sequence. However, every sequence will start a new band, which results in the number of sequences for the lowest possible cut-off.

Another special parameter, "—compriseToBands", is also capable of invoking a cut-off calculation, albeit it was implemented with another intention in mind. Two differences in the behavior must be considered. The first difference is that it does not invoke a scoring, but that it gets its scores from a file. This file contains the resulting scores from a previous scoring. The second difference is that the calculation is restarted every time the beginning of a sequence is encountered.

Originally, the parameter was designed to comprise sequence-window scores to bands. This is where the name originates:

```
--compriseToBands
```

Comprising scores to bands follows the same rules as mentioned for a cut-off calculation. Bands are maximally in width and do not overlap each other. However, to start the comprising a valid cut-off must be provided to the program via command-line parameter “–c”. In this case, the file with the scores is read and every band with scores above the cut-off is output in the form: “seq#:start_pos-end_pos:max_score;min_score;mean_score”. In the beginning, the sequence number is given, followed by start and end position of the band. The last three values give maximal, minimal and mean score, which were encountered and calculated from the scores before being comprised to a band.

The following special parameter will start a clustering (see Chapter 5.3 for details). The name of the parameter is a composition of different adjustment values used to fine-tune the clustering:

```
--clusteringXxxxYyyy
```

A clustering is performed over all motifs provided via option file. “Xxxx” can be ’Manhattan’, ’Euclidean’, or ’Likelihood’ and specifies the relation measure between motifs. “Yyyy” can be ’Greedy’ or ’Forward’, and specifies the algorithm used to form a cluster. Additionally, a valid term for “Yyyy” is ’Relation’, which prints the relation value between each two motifs in a big table without forming clusters. A threshold can be defined using parameter “–c”, which causes the clustering to stop when the used relation measure exceeds this value. If the number of clusters is known beforehand, use parameter “–C” to set the number of clusters to reach. If both parameters are omitted a full clustering is performed, yielding only one super cluster.

The last special parameter was implemented to perform an evolution on motifs with the goal to maximize the weights. Consequently, positive and negative training set must always be provided (parameter “–m” and “–b”), along with the parameter itself:

```
--motifEvolutionZzzz
```

Again, the parameter’s name adapts with the evolutionary machinery chosen by replacing “Zzzz” with either “ES”, or “RewardedSelection”. The details on the approaches can be found in Chapter 5.2. After performing the evolution the *jPREdictor* prints the 20 highest-weighting motifs. Both approaches are capable of evolving basic motifs as well as double motifs. In either case, parameter ’-c’ denotes the starting temperature in the process of simulated annealing. High temperatures mean many mutations from parent to child generation and, thus, big jumps over the search space. For low temperatures the evolutionary process tends to preserve the high weighting motifs found so far. The temperature is reduced by one every generation and if it reaches zero the process ends. Thus, the starting temperature denotes the number of generations as well. The default value is $1,000$. Parameter ’-C’ alters how the motifs are changed from one generation to the next. It is possible to assign probabilities for different mutation events and to switch recombination on or off. Mutations affect the length of a motif, the distance constraint in a MultiMotif, the number of errors allowed for a match, and the nucleotides in a motif.

4 Prediction and properties of PRE/TREs

This chapter deals with the prediction of Polycomb/Trithorax response elements (PREs) on sequences of the *Drosophila* species. PREs are one kind of *cis*-regulatory elements. Motifs are the fundamental element in the prediction process. Basic motifs are presented in Chapter 2.3. In the first section, the motif sets built from the basic motifs are listed. Then, based on the work of Ringrose et al. [5], the prediction pipeline is presented, followed by summarizing the results from their work. All subsequent chapters address the question, how the prediction process can be improved without losing statistical significance.

Emphasis of this chapter is placed on the analysis of the basics of the predictory process. Adding new motifs, creating different motif sets, using different training sets to weight the motifs, and changing the underlying null model for cut-off creation all lead to a deeper understanding on how the prediction works. Experience shows that there is not the one motif set or the one training set that covers all possible PREs in an organism, but normally different motif sets yield different predicted PREs, albeit often overlapping.

4.1 Motif sets

The basic motifs to be analyzed for their use in the prediction of PREs are listed in Table 2.6 on Page 23. In this table, the log-odd-value (column "Odd") is comparable to the weight of a motif. Thus, it reveals over- and under-representation, but on a genome wide scale. Essentially, no motif is underrepresented, all show a log-odd-value near or above zero. However, some motifs, especially the longer ones, i.e. En1, and G10, occur more often than expected. This hints at their importance for the organism. Hunchback (HB) and Knirps (Kni) normally play a role in enhancer/silencer functionality and are included in the PRE/TRE analysis as a kind of robustness test.

These basic motifs are combined to double motifs and only these double motifs are used in the prediction process. This was done, because single motifs are often too unspecific. Even many of them found in a tight spot cannot ensure the spot to be the element in question. This was first shown in detail by Ringrose et al. [5], and to overcome the limited specificity, Ringrose et al. composed all single motifs into double motifs. In addition, this approach is consistent with reports of single-type basic motifs to act cooperatively. It was shown for the Zeste motif [78] as well as the GAF motif [73]. Furthermore, this co-occurrence approach was also used for predicting transcription factor binding sites in *Escherichia coli* [108]. Combining the basic motifs to double motifs shifts the emphasis from simple motif enrichment to clustering of motifs in a dense spot.

Not all motifs were used in all predictions. To reveal influences of certain motifs on the prediction results, nine different motif sets were created. Table 4.1 lists these motif sets and their corresponding basic motifs which will be used for further analysis. Note that before a prediction is made all basic motifs in the different motif sets are combined with each other to form double motifs, where the

Table 4.1: Motif sets used for the prediction of PRE/TREs and in robustness tests. The column “Doubles” specifies the number of double motifs resulting from combining the single motifs.

Name	Single Motifs	Doubles
Original	En1, GA, G10, Z, PH, PM, PS	28
DSP1	En1, GAF, G10, Pho-DSP1, Pho, Z	21
SP1	like DSP1, but +SP1	28
Grh	like SP1, but +Grh	36
HB+	like SP1, but +HB	36
Kni+	like SP1, but +Kni	36
HBKni	like SP1, but +HB and +Kni	45
HB-	like DSP1, but +HB	28
Kni-	like DSP1, but +Kni	28

distance between two motifs may not exceed 219 nucleotides. The row “Doubles” in Table 4.1 gives the number of double motifs resulting from such combinations. The term “double motif” might be a bit misleading, because any motif paired with Pho-DSP1 will be a non-flat triple motif, and self-pairing Pho-DSP1 results in a non-flat quadruple motif.

4.2 Prediction pipeline

Performing an *in silico* prediction of PREs follows a set of instructions, depicted in Figure 4.1. To this extend, the pipeline was first presented and used by Ringrose et al. [5]. A score graph, which is laid over the sequence, is the final outcome of this process, with all high scores above a certain threshold being the sought-after elements, and all low scores considered to be background and noise. An example of such a score plot can be seen in Figure 3.8 on Page 49.

The prediction starts with a set of motifs. These motifs must have certain properties. First, they should be specific enough to not occur too often in the genome. Second, they should be relevant for the element in question and therefore, third, either be enriched or avoided in the element. As such, finding them in a genomic sequence gives strong evidence that either the element was found or that this sequence position cannot house an element. The latter ensures that the element in question is predicted against the noise of the residual genome and, moreover, against other similar *cis*-regulatory elements.

In the next step the motifs are weighted (for the equation see Chapter 3.3.2). For this, positive and negative training set have to be provided. The sequences of the positive training set are the model sequences, they act as templates for the sought-after elements in the genome-wide prediction. On the other hand, the sequences of the negative training set can be seen as the background, representing noise or elements, which should be avoided. Consequently, the calculated weights should reflect this relationship. Motifs with high weights are important for the element, since they are strongly over-represented in the sequences of the positive training set in comparison to the sequences of the negative training set. Motifs with negative weights act vice versa. Motifs with small weights around zero do not contribute to the discrimination effect.

In the second-last step, a cut-off score is obtained. In the last step, this cut-off is used in the actual prediction. Both steps make use of weighted motifs in order to score sequences. The difference lies

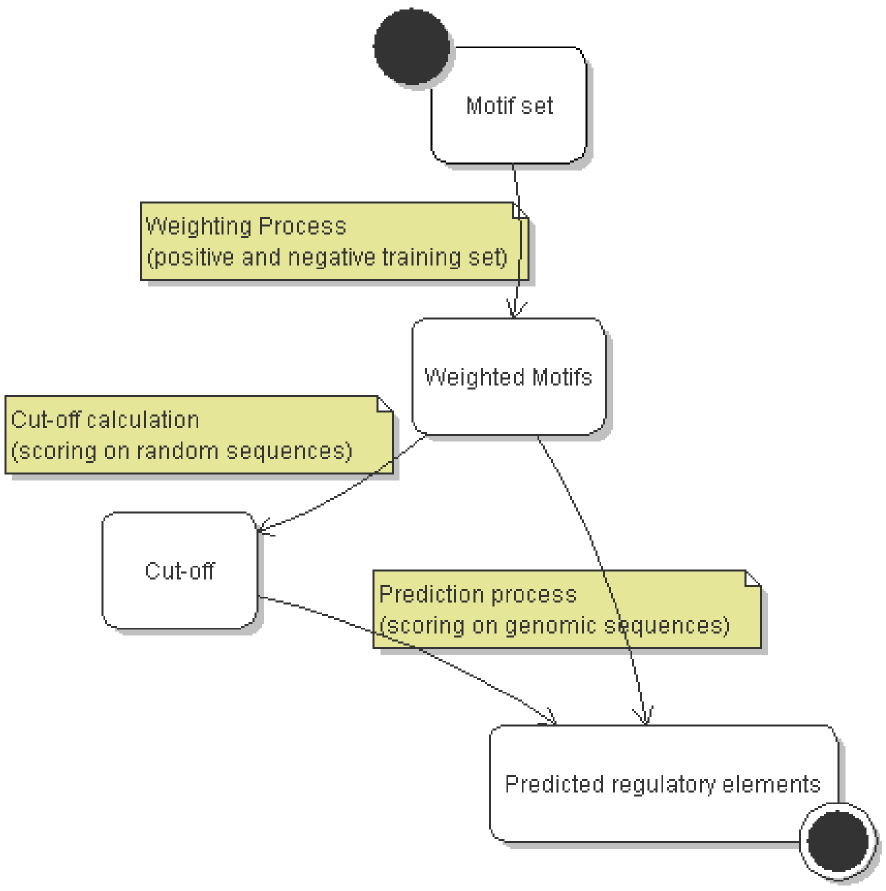

Motif set
Weighting Process
(positive and negative training set)
Weighted Motifs
Cut-off calculation
(scoring on random sequences)
Cut-off
Prediction process
(scoring on genomic sequences)
Predicted regulatory elements

Figure 4.1: Pipeline for the prediction of regulatory elements. In the first step, the initial set of motifs is weighted. Using the weighted motifs random and genomic sequences are scored. Scoring applied to random sequences yields a cut-off, which in turn is used in the prediction process. Every sequence fragment exceeding the cut-off is reported to be a regulatory element.

in the kind of sequences to score. In the process of scoring, a window is shifted over the sequence and every window gets a score. In order to obtain the cut-off for the genome-wide scoring, random sequences, many times the size of the genome and generated under a null model, are scored. This calculation yields the cut-off corresponding to an expected number of elements to occur merely by chance.

Finally, in the last step, the genomic sequence is scored. Every sequence window above the cut-off is treated as the candidate element, all other parts of the genome are considered noise.

Running a prediction yields a number of short sequences considered to be candidate PREs. The cut-off was chosen so that only few elements would occur by chance while scoring the whole genome. This *E*-value represents the false positive (FP) rate. All other predicted elements are considered true positives (TP). Therefore, the higher the number of predicted elements the higher the positive predicted value (PPV, also called precision rate, for the equation see Chapter 2.5). Note that this is only true *in silico*, since in this case the number of false positives rate can be well-controlled. *In vivo* or *in vitro* the precision rate might drop drastically.

For a genome-wide prediction the number of true as well as false negatives is not known. Based on the assumption that the sought-after elements rarely occur in the genome, a guess for the number of true negatives would be the number of adjacent sequence windows. In this case, specificity is very high. Sensitivity, on the other hand, cannot be specified, since the real number of positives is not known, at least not genome-wide.

While the presented prediction pipeline is not altered, the single steps are subject to change in order to increase the number of predicted PREs. A higher number of predicted elements increases the PPV. While the specificity stays the same, because it is determined by the *E*-value, the sensitivity will increase together with the PPV (even though it cannot be specified). The latter is the big goal in all analysis, increasing sensitivity while at the same time maintaining a very high specificity.

4.3 Predictions in 2003

All predictions presented here are based on the ideas and data of Ringrose et al. [5]. The authors performed a genome-wide prediction of PRE/TREs and used the motifs GA, G10, Z, PF, PM, PS, and the *engrailed* (En1) binding site (Table 2.6 on Page 23). These seven motifs were combined pairwise to 21 double motifs, with distance zero to 219. After weighting them, random sequences generated under a zeroth order Markov model were scored. This led to a cut-off score of 157, which corresponds to an *E*-value of 1, i.e. one predicted element is expected to be a false positive.

On a genome wide scale in *Drosophila melanogaster* 167 PRE/TREs were predicted [5]. Only one is expected to be a false positive, which gives a positive predictive value of 99.4%. The genome length of *D. melanogaster* is around 117 Mbases. Since prediction was performed with a window width of 500, around $234,000$ of such windows exist non-overlapping. The majority of the genome does not house a PRE/TRE, and even if this number drops by a few thousand, the specificity stays higher than $99,9\%$.

After running the *in silico* prediction, Ringrose et al. [5] evaluated the results biologically. They performed chromatin immunoprecipitation (ChIP) to detect enrichment for Polycomb binding in *Drosophila* Schneider cells. Twenty-nine of their 43 tested PRE/TRE fragments showed a very high enrichment. The residual 14 fragments might not be PRE/TREs. Alternatively, they could still be

PRE/TREs, but without enrichment in this special cell type. Assuming the prediction results to be correct, the sensitivity of the biological test is 67%. Assuming the biological test to be correct, the positive predictive value of the prediction drops to 67% as well. Therefore, at least two out of three predicted PRE/TREs are correct, which is a great result for a computational prediction.

The number of predicted PRE/TREs can be increased by simple means. In the original prediction by Ringrose et al. [5] the sequence window moving over the genome was of width 500 and was shifted in steps of 100. Reducing the shift value to 10 means capturing all PRE/TREs previously missed because their motif-enriched sequence fragments were not covered precisely enough by the shifting window. The result was an increase in the number of predicted PRE/TREs to 201 (unpublished data). The best shift value would be one, because it ensures that every two motifs at the beginning and at the end of the window are captured correctly. Nevertheless, considering the number of predicted PRE/TREs, virtually no difference exists to a shift value of 10. However, computation is faster for a higher shift value, therefore, in all future predictions a shift value of 10 will be taken.

The first application done with the new *jPREdictor* was to redo the PRE/TRE prediction in *D. melanogaster*. The motifs used were GA, G10, Z, and En1 (Table 2.6 on Page 23). Additionally, instead of the three Pho derivatives the Pho PSSM was used (Table 2.5 on Page 14). A new motif, DSP1, was coupled with the Pho core motif to $GCCAT-(0,40)-GAAAA$. These 6 motifs were combined pairwise to 21 motif patterns. Scoring random sequences yielded a cut-off score of 70. As a result, 344 PRE/TREs on a genome wide scale were predicted [104]. This number is slightly lower than the published one due to a methodic error in evaluation.

4.4 Motif weights

Motif weights depict over- and under-representation in one sequence set with respect to another sequence set. This alone makes them very meaningful, since different sequence sets can be compared for their motif composition via the weights of motifs.

Equation 3.4 for calculating weights can be found in Chapter 3.3.2. If both sets, positive and negative, contain only one random sequence generated under the same model, and if $|S| \rightarrow \infty$, the motif weights become $w(M) = 0$, since for each set $\frac{fr(M|S)}{|S|}$ resolves to the probability of approaching a motif as discussed in Chapter 2.2.

Due to both normalization steps, by sequence length and by number of sequences within a set, the equation is very well balanced and motif weights obtained from applying different sequence sets can be compared without further conversions. Nevertheless, one side effect must be mentioned when working with the formula. It results from adding the pseudo-count c before applying the second normalization. While this is necessary to avoid dividing by zero, the pseudo-count is normalized, too. Therefore, if the motif is not found in both sequence sets, the number of sequences in either set decides on the weight. This can be seen in Figure 4.2. If both sequence sets contain an equal number of sequences, a weight of zero is obtained for any non-occurring motif. The weight becomes negative, if the positive training set contains more sequences, and negative otherwise. Nevertheless, this side effect normally causes no trouble, since sequences and motifs were chosen beforehand to be in a way compatible, i.e. the sequences contain the motifs.

In the PRE/TRE prediction from 2003, Ringrose et al. [5] used confirmed PRE/TREs as the positive training set, and heatshock (HS) gene promoters as a negative training set. In Figure 4.3 the weights of

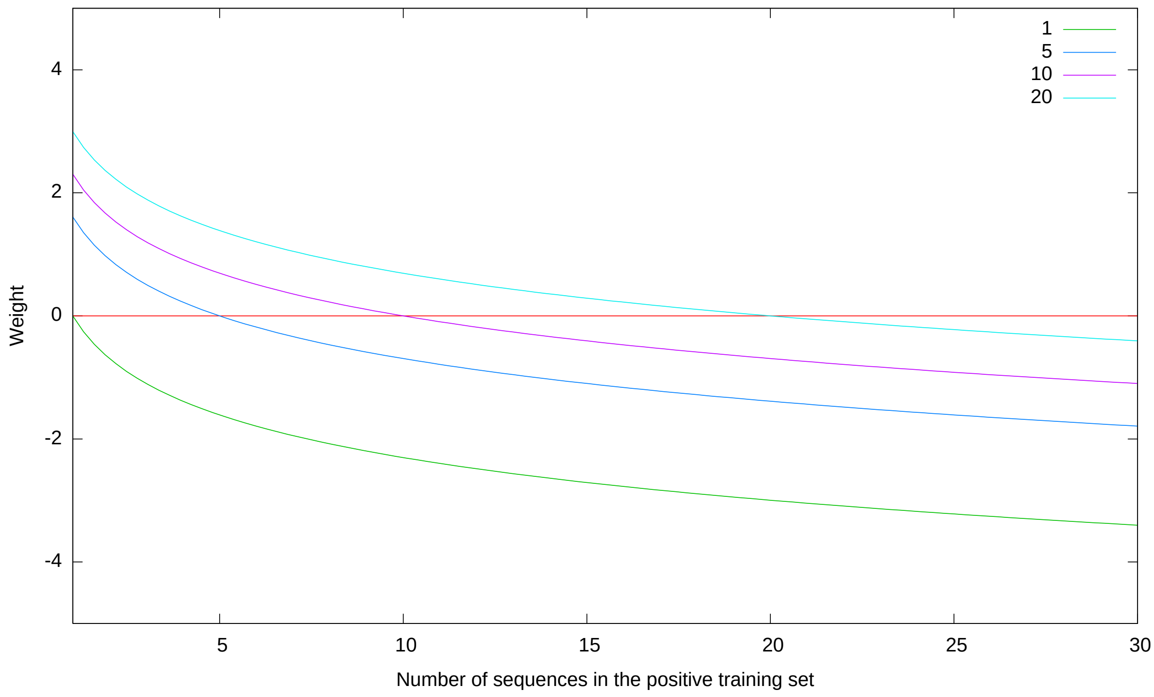


Figure 4.2: Weights for a motif, which is not found in either sequence set, dependent on the numbers of sequences in the positive (x-axis) and in the negative training set (1, 5, 10, and 20), These weights result from the normalization of the pseudo count value.

all double motifs of the Grh motif set can be seen for this training set, as well as for two other training sets. For the first two sets, PREs vs. HS promoters, high negative weights are obtained for some GAF and G10 double motifs due to frequent (GA) repeats in HS promoters [5]. However, GAF motifs must not be completely avoided in PRE sequences, since double motifs containing only one GAF as part of their pattern obtain mediocre or high positive weights. Especially, any motif pattern containing En1 and Pho gets high weights. On the other hand, the Grh double motif has a weight around zero, and is therefore neither over- nor underrepresented in the PREs with respect to HS promoters. Therefore, at least it cannot be used to distinguish between PREs and HS promoter sequences.

The negative training set “promoters” (Figure 4.3) contains 28 promoter sequences, 16 of them are the aforementioned HS promoters. The residual 12 promoters are from cell cycle genes and chosen to not play any role as PRE/TRE or HS promoter (see Table A.4 in the Appendix). Therefore, they represent random sequences in some way. Adding them to the negative training set, shifts the weight for most double motifs toward higher positive values. The reason for this is that the used motifs occur very rarely in the new promoters. This is not true for some patterns of Zeste, Grh, and SP1, their weights drop with the new negative training set. This is because these three motifs are very frequent in at least two of the new background sequences, namely cycA and dp.

For the third training set in Figure 4.3, the model consists of 15 sequences predicted by Negre to be PRE/TREs [23] and the background consisted of the same promoters as in the second set. Especially double motifs containing the two GAF variants and Grh have low weights. Thus, Negre sequences seem to lack GAF binding sites. This was the reason why Negre et al. [23] suggested that GAF is not necessary for PRE/TRE functionality, despite the fact that all previously described PRE/TRE

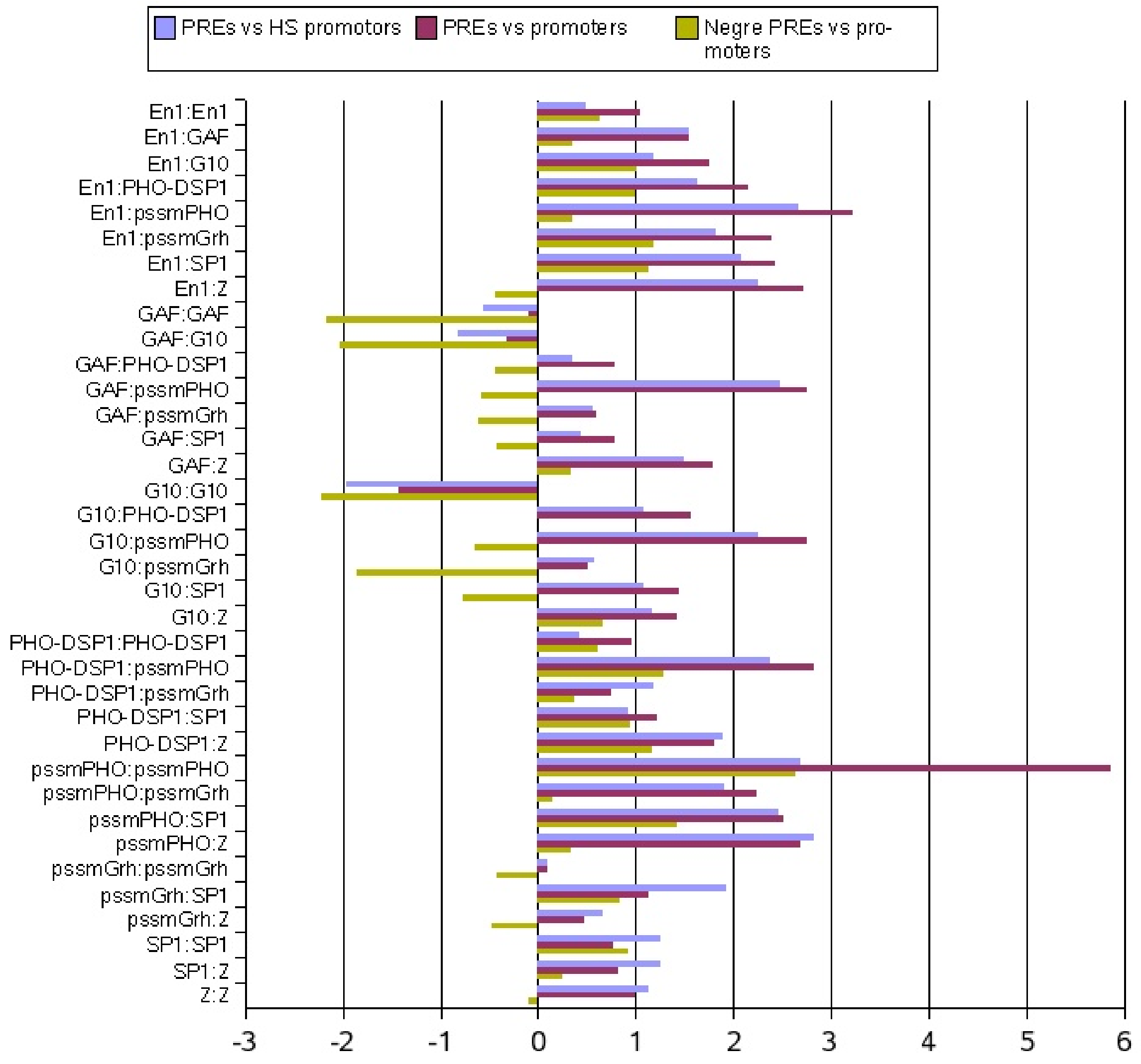


Figure 4.3: Weights of double motifs calculated using different training sets. The positive training set "PREs" contains 12 confirmed PRE/TRE sequences, while the negative training set "HS promoters" contains 16 promoter sequences of heatshock proteins. These two sets are the same as used in [5]. The set "promoters" consists of the 16 HS promoter sequences and additional 12 promoter sequences from cell cycle genes. The 15 sequences in the "Negre PREs" set are predicted in [23].

sequences contain these binding sites.

In general, the sequences which are part of one of both training sets should be chosen with care, since they determine the motifs weights. This is especially true, if the weights are not only used for comparative reasons, but are used in further analysis, e.g., in the process of scoring sequences. They define which motifs are avoided and which motifs are preferred in the model sequences with respect to the background sequences. Examples for over-representation are the Pho variants, while the GAF variants are under-represented. Additionally, they define which motifs are actually unimportant for the distinguishing process, i.e. the Grh motif.

In order to use weighted motifs in the process of scoring sequences, which results in the prediction of regulatory sequences, the positive training set should consist of *cis*-regulatory sequences similar to the ones that are to be predicted. The sequences of the negative training set should consist of sequences whose motif composition should be avoided. This is necessary, because during the scoring process motifs are not simple counted but their weights are added. Therefore, one motif can add greatly to the score of a sequence window while another with a score around zero might occur a lot but adds nothing to the score. Negative weights may be assigned to motifs that are to be avoided in the predicted sequences.

Another intricacy is the use of motif patterns. Again, some patterns might have low weights, but some others might have high weights. In this sense, avoidance of basic motifs which later form the patterns cannot mean complete absence. One basic motif can be part of several motif patterns, and if the basic motif is absent on a sequence, all patterns with this basic motif are not found.

Additionally, motif weights are essential for distinguishing different types of sequences. For a genome-wide analysis, high scoring sequences are distinguished against noise or background sequences, which are assigned lower sum-scores. In accordance to the motif composition of the analyzed training sequences different sum-scores are yielded. High weights for found motifs add up fast to a high sum-score. In comparison, low weighting motifs contribute to the sum-score to a lesser extend, more of them are needed to compensate for one high-weighting motif. On the other hand, motifs with high negative weights countervail the motifs with high positive weights. Therefore, since the sum-score is strived for being maximal, this motif is seen to be avoided in the high-scoring sequences.

On a smaller scale other than genome-wide, the distinguishing process works as well. Sequence by sequence is scored and classified after the yielded sum-score, which was calculated in accordance to its motif composition. In this process, sequences with similar motif frequencies, but different motif composition can be distinguished, i.e. PREs vs. heatshock promoters [5].

4.5 Scoring null models and resulting cut-offs

In order to predict PREs *in silico*, genomic sequences are scored in accordance to the found motifs. Every found motif adds its weight to the score of a sequence fragment, the latter of length 500 in PRE/TRE prediction. Sequence fragments with high sum-scores hint at the desired functionality, low scoring parts of the genome are meant to be noise or background. In this sense, a score is needed to distinguish PREs from the noise level. In order to obtain such a cut-off, background sequences are scored, too. This leads to a score, the distinguishing cut-off, for which only one regulatory element is expected to be found in the genomic sequence. Therefore, this cut-off score corresponds to an E-value of one, which is at the same time the number of expected false positives in the prediction. Every other

Table 4.2: Comparison of null models derived from the *D. melanogaster* genome. Values are calculated as sum over absolute log-odds of normalized occurrences from all possible DNA motifs of length 5 (upper right part of the matrix) and 6 (blue shaded area). Two null models differ more from each other in terms of motif occurrences the higher the numbers.

		Shuffled		Markov chain			
	D. mel.	Out	In	MC2	MC1	MC0	Probability
D. mel.	0	56	-	116	178	306	306
Shuffled-Out	301	0	-	88	134	260	260
Shuffled-In	431	141	0	-	-	-	-
MC2	627	445	442	0	142	285	285
MC1	878	639	571	650	0	251	250
MC0	1440	1194	1079	1291	1117	0	2
Probability	1440	1194	1079	1291	1117	15	0

sequence fragment exceeding the cut-off counts as identified PRE.

4.5.1 Null model analysis

The background sequences are the null model for the prediction. They are random sequences, somehow derived from the original genomic sequence. And since there exist multiple ways to derive a background sequence the cut-off for the prediction of regulatory elements is not fixed. It is rather far from it.

The best null model would be the original genome where all sought elements are eliminated. But since the sought elements are unknown other methods to build the null model must be used. The task is to build a background sequence closest to the *D. melanogaster* genome. In Table 4.2 different background sequences are analyzed for their motif occurrence counts and compared to each other. The table was built by searching all possible motifs of length five and six in all sequences.

The shuffled-out sequence was created by concatenating fragments of length 10 that were randomly chosen from the *D. melanogaster* genome. This is the same as drawing with replacement. The disadvantage of this method is that there might be fragments of the source genome which are not represented in the background and others which are represented twice or more. Of course, such probabilities are very small if the shuffled sequence is for instance 100 times the size of the source genome. Nevertheless, for comparative reasons, another way to create a shuffled background sequence was applied, shuffled-in. It works by systematically exchanging two randomly chosen fragments of length 10. This leads to a background sequence which consists completely of the source genome. Therefore, nucleotide composition is exactly the same and no fragment occurs twice like in a shuffled-out genome.

To generate the Markov chain sequences labeled MC2, MC1, and MC0, the *D. melanogaster* genome was analyzed for its nucleotide composition and for its transition probabilities between nucleotides. The number gives the order of the Markov chain, e.g. for MC2 the transition probabilities used are $P(x_{i+2} = \sigma_2 | x_{i+1} = \sigma_1 \& x_i = \sigma_0)$, where x_i gives the nucleotide at sequence position i and every σ denotes a nucleotide from the alphabet. In the appendix, the tables of transition probabilities corresponding to a Markov chain of order one and two can be found (Table A.2 and Table A.3).

As a result of all derivation methods from the *D. melanogaster* genome, higher order structures in

Table 4.3: Score cut-offs for different motif sets, different training sets and different null models. Cut-offs corresponding to an *E*-value of 1 are shown. Training set “orig” refers to the PRE sequences vs. HS promoters. Training set “ext” refers to the PREs vs. promoters.

Motif set	Training set	MC0	MC1	MC2	Shuffled
Original	orig	159	185	215	216
	ext	162	195	223	231
DSP1	orig	73	87	105	112
	ext	86	106	126	141
SP1	orig	99	118	131	141
	ext	108	136	158	171
Grh	orig	109	129	144	154
	ext	120	145	164	179
HB+	orig	132	216	263	287
	ext	124	160	186	203
Kni+	orig	98	120	133	142
	ext	117	139	159	173
HBKni	orig	131	215	263	287
	ext	124	162	191	204
HB-	orig	96	175	223	240
	ext	94	129	157	169
Kni-	orig	72	88	107	113
	ext	87	106	132	142

the genome, like genes, promoters, and repetitive elements occur merely by chance. Table 4.2 shows that both shuffled sequences are the closest representatives of the *D. melanogaster* genome in terms of motif preservation. Nevertheless, the shuffled-out approach is superior to the shuffled-in approach. The reason for that is that the repeated exchange of sequence fragments disrupts the fragments of formerly length 10. Therefore, in all further analyses, the shuffled-in approach is omitted.

4.5.2 Cut-off scores

The cut-offs used in further predictions are not fixed, but depend on the null models scored. Of course, they depend on the used motif sets and the motif’s weights, too. In Table 4.3 score cut-offs for different motif sets weighted using different training sets are shown. They correspond to an E-value of 1 and result from scoring random sequences 100 times the size of the *D. melanogaster* genome, around $11,7$ Gb. In Figure 4.4 the curves for different cut-off calculations are shown for the “Original” motif set.

The first trend to be seen is that the closer a null model is to the original genome the higher is the cut-off. This is valid for each motif set and each training set. Second, the cut-offs for the extended training set are higher in comparison to the original training set, except for every motif set containing HB. The reason is that the weights are shifted to higher positive values when the extended training set is applied. Double motifs containing HB are an exception to this observation. HB occurs not only frequently in heatshock promoters but also in cell cycle promoters. This leads to higher weights for double motifs containing HB for the extended training set in comparison to the original training set.

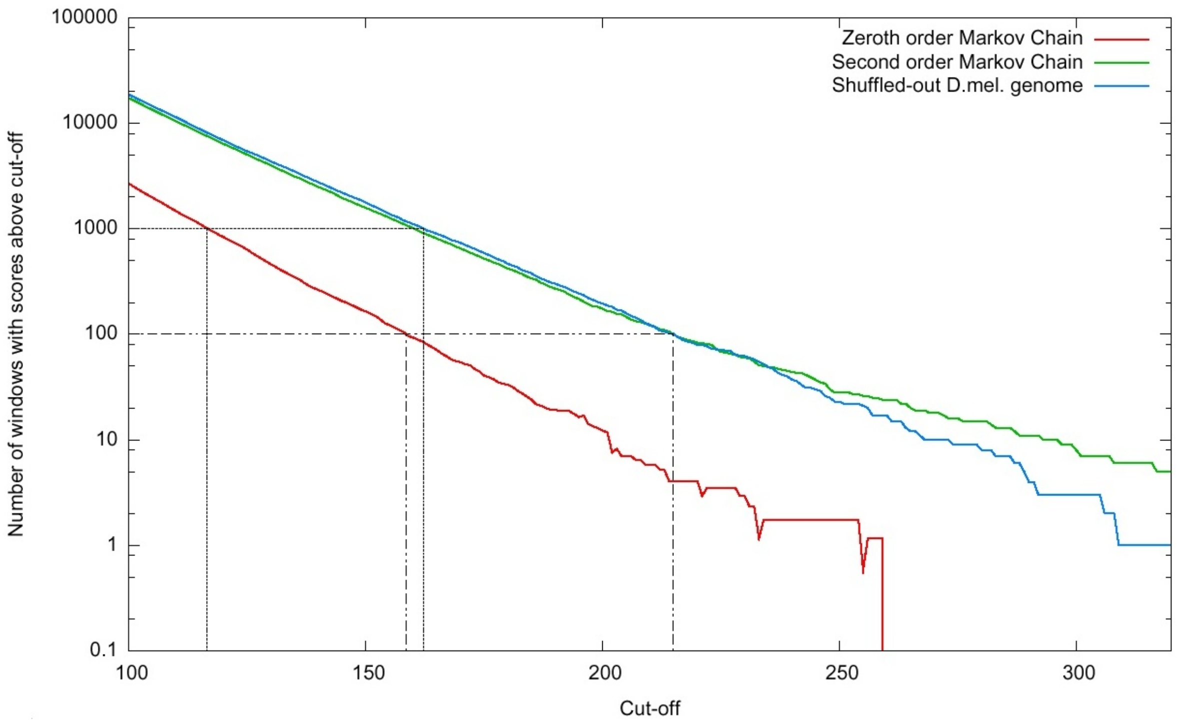


Figure 4.4: Results from a cut-off calculation: number of windows in randomly generated sequences exceeding the cut-off. The sequences are scored using the “Original” motif set, weighted with the PREs vs. HS promoters training set. The random sequences are derived from the *D. melanogaster* genome and are 100 times the size of the genome. A window number of 100 marks an *E*-value of 1 (dashed-dotted line), a window number of 1000 marks an *E*-value of 10 (dashed line).

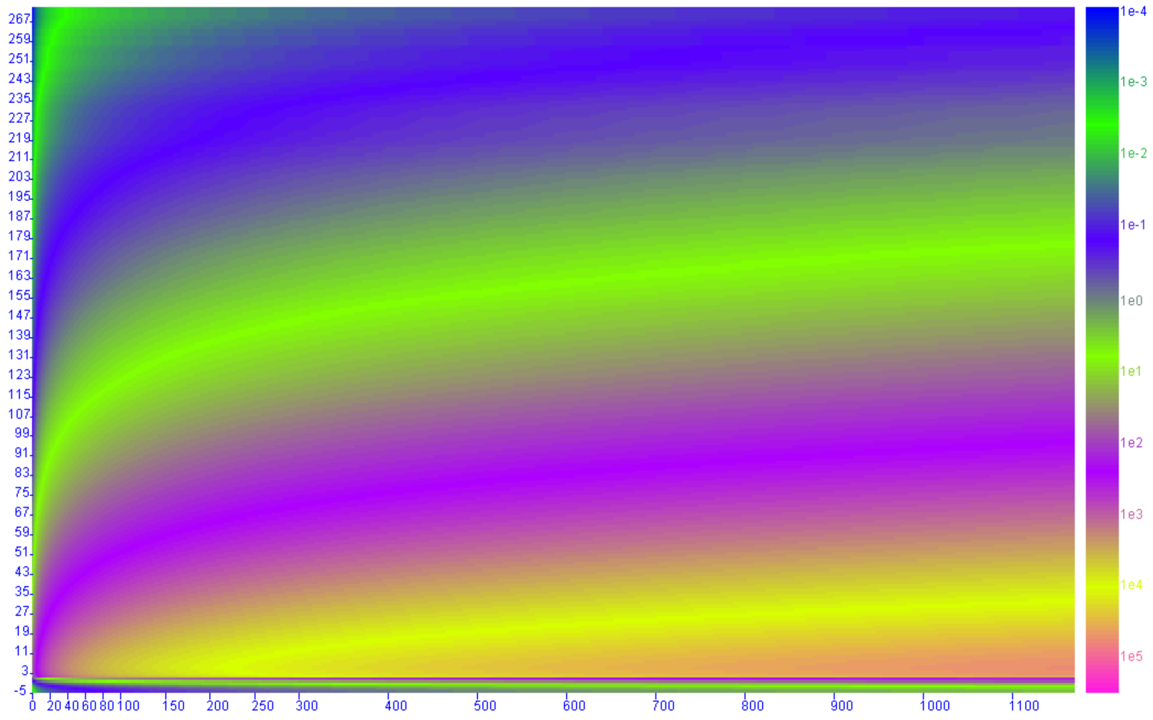


Figure 4.5: Cut-offs for different sequence sizes (in 10^5 nucleotides), color-coded corresponding to an *E*-value. This picture was drawn from a cut-off calculation using the following settings: “Original” motif set, PREs vs. promoters training set, shuffled *D. melanogaster* genome. A cut-off of 216 corresponds to an *E*-value of 1 in the *D. melanogaster* genome of size 117 million nucleotides.

From the “Original” to the “DSP1” motif set, the cut-offs drop drastically, although the sets differ only in their Pho representation and in the new motif DSP1. But that explains the difference very well. A Pho binding site found in the genome counts multiple times in the “Original” motif set, since this set contains 18 out of 28 double motifs, which have a Pho motif as part of their pattern. In the contrary, in the “DSP1” motif set, only 11 out of 21 double motifs have Pho as part of their pattern. Additionally, the Pho representatives from the “DSP1” set are more restrictive in matching.

Adding Kni to the “SP1” motif set does not change the score cut-off. Adding Grh rises the cut-offs by up to 10%. The greatest impact has HB, which, after adding it to the “SP1” motif set, rises the cut-off by up to 100%. This can only be explained by the probability with which the three motifs match on a random sequence: HB is ten times more frequent than Kni and 2.5 times more than Grh. Therefore, finding HB near the other motifs is much more probable than finding Kni or Grh close by.

Cut-off calculation can be applied to other *E*-values as well. Figure 4.5 gives an overview, how the cut-offs are distributed in accordance to different sequence lengths and different *E*-values. This picture was drawn by a short self-written Java program called “CutoffrecalcDraw”, which recycles the cut-off calculation results obtained from running the *jPREdictor*. This recycling is necessary, since a cut-off calculation is done with a specific *E*-value in mind, in this case 1. Therefore, subsequent recalculation of the results for other *E*-values, especially for smaller ones, have to be taken with care. The reason is that smaller *E*-values require the null model to be larger in order to minimize the variance. While

Table 4.4: Numbers of predicted PRE/TREs in the *D. melanogaster* genome and other sequences using several motif sets. All predictions use a window of width 500, shifted in steps of 10. The training set is the extended one, 12 confirmed PREs vs. 28 promoters from heatshock and cell cycle genes. Cut-offs were calculated using a zeroth order Markov chain. The 131 and 30 domains provided by Tolhuis et al. [22] and Negre et al. [23], respectively, are sequences from the *D. melanogaster* genome enriched for PcG proteins. In parenthesis are the predicted number of sequences using the lower cut-off.

Motif set	Cut-off	PRE/TREs	Training set		Cut-off	Tolhuis	Negre
	117 Mb		pos	neg	3.7 Mb	(131)	(30)
Original	162	301	3	0	101	7 (26)	0 (1)
DSP1	86	481	4	0	50	12 (43)	0 (3)
SP1	108	708	5	0	73	17 (42)	0 (2)
Grh	120	602	4	0	77	14 (45)	0 (2)
HB+	124	934	6	0	80	20 (53)	2 (5)
Kni+	117	566	5	0	74	15 (44)	0 (2)
HBKni	124	947	6	0	81	22 (49)	2 (5)
HB-	94	840	4	0	58	24 (51)	2 (5)
Kni-	87	470	4	0	53	12 (43)	0 (4)

for an *E*-value of one a null model 100 times the size of the genome is sufficient, an *E*-value of 0.1 already requires the null model to be $1,000$ times larger. The influence of variance can be seen in Figure 4.4, where the linear relationship is disturbed for high cut-offs.

Nevertheless, Figure 4.5 gives an impression how the cut-off changes with the size of the source sequence and with different *E*-values. The smaller the size of the genome the smaller the cut-off becomes, since it gets more and more improbable to obtain such a high score merely by chance. Nevertheless, this decrease is not linear, a root function more accurately describes the shape of the curve. On the other hand, the cut-off becomes higher and higher, the smaller the *E*-value becomes. This relationship again is not linear, but seems to be close to exponential. Changing the *E*-value from 10 to 1 rises the cut-off almost by the same amount as changing the *E*-value from 1 to 0.1.

The conclusions from the null model analysis is that a shuffled genome is closest to the original genome in terms of motif occurrences. This results in very high cut-offs for each motif set in comparison to other null models. As a consequence, the number of predicted PREs will be reduced. However, the chance for a predicted PRE to be a false positive is reduced, too. This higher reliability of the prediction is the reason, why the shuffled-out null model is used as a further reference model.

4.6 Prediction results

In this chapter different predictions of PRE/TREs are presented and analyzed. In Table 4.4 the influence of different motif sets on the prediction can be seen. The high cut-offs ensure that the sequences of the negative training set are not predicted to contain PREs. The specificity is therefore 100%. On the other hand, the high cut-offs also prevent in part the prediction of PREs in the true PRE/TREs of the positive training set. However, the sensitivity changes with the motif sets. Applying the "Original" motif set, three out of 12 sequences from the positive training set are predicted to contain a

Table 4.5: Numbers of overlapping PRE/TREs previously predicted in the *D. melanogaster* genome using several motif sets. In parenthesis are percentage values of overlapping sequences.

Motif set Predicted PREs	Original 301	DSP1 481	SP1 708	Grh 602	HB+ 934	Kni+ 566	HBKni 947
Original	-	158 (33)	183 (26)	165 (27)	184 (20)	170 (30)	184 (19)
DSP1	158 (52)	-	396 (56)	366 (61)	402 (43)	351 (62)	400 (42)
SP1	183 (61)	396 (82)	-	575 (96)	624 (67)	565 (99)	626 (66)
Grh	165 (55)	366 (76)	575 (81)	-	562 (60)	530 (94)	563 (59)
HB+	184 (61)	402 (84)	624 (88)	562 (93)	-	548 (97)	929 (98)
Kni+	170 (56)	351 (73)	565 (80)	530 (88)	548 (59)	-	551 (58)
HBKni	184 (61)	400 (83)	626 (88)	563 (94)	929 (99)	551 (97)	-

PRE (namely *iab7*, *iab2*, and *bxd*, see [5]). This leads to a sensitivity of $\frac{3}{3+9} = \frac{1}{4} = 25\%$. Using the “DSP1” motif set correctly identifies four sequences from the positive training set (namely *iab7*, *iab2*, *bxd*, and *engrailed*). Despite the great changes in the motifs, the three model sequences confirmed with the “Original” motif set to contain a PRE are included. Applying the “SP1” motif set adds another PRE sequence to the set of correctly predicted ones (namely *polyhomeotic PRE distal*), increasing the sensitivity to 42%.

The number of predicted PREs shows great variety. Ringrose et al. [5] predicted 167 PREs on a genome-wide scale, the number increased to 201 by changing the shift-value for the sliding window from 100 to 10 (results not shown). Fiedler et al. [104] changed the motif set to “DSP1”, but preserved the training sets, and predicted 344 PREs in the *D. melanogaster* genome. On the contrary, all predictions in Table 4.4 are made using the PREs vs. promoters training set. With it, the number of predicted PREs was increased from 201 to 301 for the “Original” motif set, and from 344 to 481 for the “DSP1” motif set. All further predictions presented here are based on the “DSP1” motif set, which is extended motif-by-motif to show the impact of every motif on the predictions (Table 4.1).

Adding the SP1 motif to the “DSP1” motif set increases the number of predicted PREs by roughly 47% to 708. Both predictions share 396 fragments (see Table 4.5). Therefore, 85 sequences are uniquely predicted with the “DSP1” set due to the lower cut-off. The novel 312 sequences, almost 45% of the predicted 708 ones, are caused by the SP1 motif. Thus, the SP1 motif frequently occurs near the other motifs, such that motif patterns containing SP1 are often found. Adding Grh to the “SP1” motif set does not add to the number of predicted PREs. On the contrary, the higher cut-off causes the number of predicted PREs to drop significantly. Additionally, only 27 novel PREs are predicted (Table 4.5). Therefore, Grh occurs rarely near the other motifs and its impact on PRE/TRE functionality seems to be limited, at least in comparison to adding SP1.

Adding Hunchback (HB) and Knirps (Kni) to the “SP1” as well as the “DSP1” motif set was meant to be a robustness test for the prediction. It was assumed that HB and Kni occur within functional PRE/TREs merely by chance, which would have caused the number of predicted sequences to stay the same or to increase only slightly.

However, the basic HB motif occurs very frequently in all sequences of the positive training set, often much more frequent than even the core Pho motif. Nevertheless, in few sequences of the negative training set, in HS as well as cell cycle promoters, HB also occurs very frequently. This leads to a negative weight of -0.13 for the HB-HB double motif. However, combined with other motifs, HB yields positive weights, e.g. 1.5 if combined with En1 as well as with the Pho PSSM motif. The reason for the positive weights is that Hunchback binding sites occur enriched near the other motifs. The positive weights for double motifs containing HB as well as the high occurrence probability of HB in the *Drosophila melanogaster* genome result in a huge number of predicted PREs for the "HB+" motif set. Additionally, due to a cut-off only slightly higher than the one yielded with the "SP1" motif set, the prediction results for "HB+" overlap strongly with the ones for "SP1".

Contrary to including HB, adding Knirps to the "SP1" motif set changes almost nothing in the prediction results. The higher cut-off of 117 decreases the number of predicted elements (566, Table 4.4) much more than adding a new motif could rise them. This also can be seen in Table 4.5, only one result predicted with the "Kni+" motif set is novel, the other 565 are already predicted using the "SP1" motif set.

All motif sets were also applied to the prediction of PcG protein enriched sequence fragments. Tolhuis et al. [22] provided 131 fragments with an overall length of 3.7 Mb. Negre et al. [23] provided 30 sequences each of length $4,000$ nucleotides. Using the genome-wide cut-off, a PRE was predicted in at most 18% of the Tolhuis sequences. Reducing the cut-off in order to adjust it to the smaller sequence length increases the number of Tolhuis fragments with at least one PRE to 34% (Table 4.4, numbers in parenthesis). This contrast might reveal a drawback in the biological experiment: association of PcG proteins to the DNA might not hint at a PRE at this position, because PcG proteins are discussed to play a role in many cell-regulatory processes. On the other hand, assuming all reported sequence fragments to be real PREs, the reason might be that functional PREs are content with a smaller density of binding sites than required for producing a signal in the *in silico* prediction. Applying the prediction to the sequences provided by Negre et al. [23] gives zero hits for almost all motif sets (Table 4.4). Exceptions are the three motif sets containing the Hunchback motif. Applying these sets results in always the same two sequences predicted to contain a PRE. Using the smaller cut-off results in 10% of the sequence fragments predicted to contain a PRE (Table 4.4, numbers in parenthesis, corrected for the HB predicted sequences). The same already discussed for the Tolhuis sequences applies to the Negre sequences, too.

From the previous analyses, especially from the genome-wide prediction, the most promising (stand-alone) motif set for the prediction of PREs seems to be the "SP1" motif set. It produces a high number of predicted PREs while statistically being the most sensitive. In addition, it contains all motifs contributing the most to the number of PREs. Consequently, it is used to demonstrate the influence of different sequence windows on the number of predicted PREs (Figure 4.6). The higher the width of the sequence window the higher the possible sum-scores. This leads to higher cut-offs, which, in order to predict more PREs, have to be countervailed by the higher sum-scores.

At a first glance, the number of predicted PREs rises the larger the widths of the sequence windows are (Figure 4.6). Nevertheless, at a second glance, this is only true to its full extend for the MC0 background model, where the number of predicted PREs increased by around 300% from a window width of 50 to a window width of $1,500$. The prediction process applied by Ringrose et al. [5] used the very same background and a window width of 500. On the other hand, Berman et al. [77] used a window width of 700 for their enhancer prediction. For the MC0 background, Figure 4.6 shows that from a width of 100 up to a width of 550 the number of predicted PREs increases stronger in comparison to the widths of 600 and higher. This forms a kind of "elbow" and choosing a width from

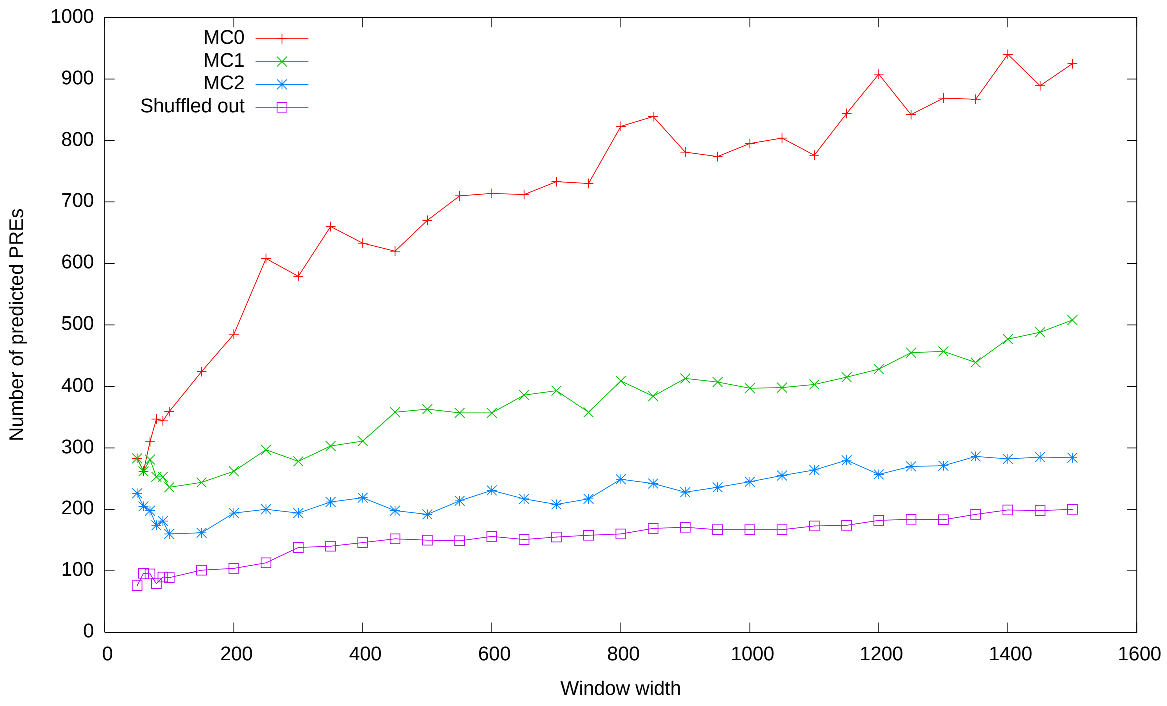


Figure 4.6: Number of PREs predicted in the *D. melanogaster* genome using the “SP1” motif set, dependent on the window width and certain background models. The cut-offs for the prediction correspond to an *E*-value of 1.

this “elbow” is reliable in terms of a maximizing the sensitivity. Applying higher ordered Markov chains and the shuffling of the genome as the background model clears out the “elbow” and also shows that the number of predicted PREs only slowly increases with the sequence window width.

This analysis was also applied to the sequences of both training sets. In Figure 4.7 the influence of the sequence window width on the number of sequences containing at least one predictable PRE can be seen. The used motif set is again “SP1”. The cut-offs chosen are the same as for a genome-wide prediction and each cut-off corresponds to an *E*-value of one. As background model, the shuffled-out *D. melanogaster* genome was taken. The negative training set contains 28 sequences, and for almost all sequence windows, none of them contains a predictable PRE. Only for smaller sequence window widths, the cut-off is low enough to have one or two sequences to contain such a PRE. In this case, specificity dropped from 100% to 96% or 93%, respectively.

The positive training set contains 12 confirmed PREs. Only one is found for high window widths larger than 920 (Figure 4.7). This number slowly rises with the sequence window width becoming smaller and smaller. For a width of 500, two sequences are found to contain a PRE. This number is contrasted by the 5 sequences, which are found if the background model is a zeroth order Markov chain (Table 4.4). At a sequence window width of 180, for which the specificity is still 100%, 5 out of 12 sequences contain at least one predictable PRE. This corresponds to a sensitivity of 42%.

For the genome-wide prediction of PREs, higher window widths yield more predicted PREs. While this is preferable in terms of a higher sensitivity (specificity remains the same), it is contrasted by the finding that a prediction in confirmed PREs prefers small widths in order to increase the sensitivity. This contradiction can be solved repeating the predictions with a very small, a mediocre, and a large

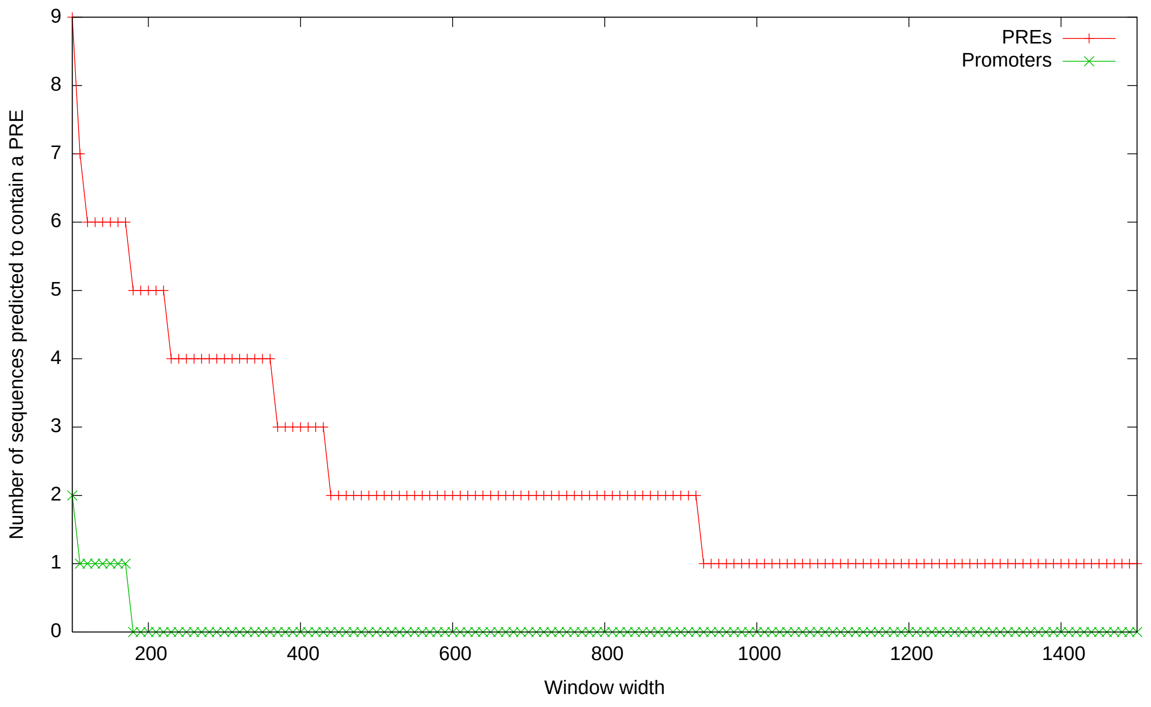


Figure 4.7: Number of sequences of the positive and negative training set predicted to contain at least one PRE, dependent on the window width. The motif set used is “SP1”. The cut-offs for the prediction correspond to an *E*-value of 1 and are calculated using the shuffled-out *D. melanogaster* genome as background.

sequence window width. After analyzing the overlap for such a prediction, the predicted PREs can be classified as short, mediocre, and long.

4.7 Smoothing sum-scores

In the prediction process each sequence window is assigned a sum-score resulting from adding up the weights of found motifs. This window is shifted along the sequence in small steps. Therefore, adjacent windows overlap with large areas. As a result, sum-scores for adjacent sequence windows often differ by only small amounts, if they differ at all.

Moreover, adjacent sequence windows with sum-scores exceeding the cut-off are combined to bands. This ensures that predicted regulatory elements (which are bands after all) are maximal in width and do not overlap each other. As a consequence, a band fully covers a regulatory element, and adding more windows from left and right to the band will not improve the mean sum-score.

Sum-scores are smoothed before bands are created. Smoothing means calculating the mean value over K adjacent sequence windows, one half before the intial window, the other half following that window. It is applied to both background and genomic data. In general, smoothing lowers high peaks and flattens high-scoring areas. This results in lower cut-offs. However, the hope is that high-scoring areas in genomic data either span over more adjacent windows or are higher in comparison to background data. This would enable them to exceed the cut-off even after the smoothing. In this sense, the lower cut-off will countervail the smoothing process, eventually resulting in more predicted regulatory elements.

Together with the smoothing process the width of the sequence window is changed. Smaller sequence windows have two advantages. First, they can cover small regulatory regions. Functional PRE/TREs are reported to be as small as 138 (MCP138, [109]) or 219 (Ab-*Fab*, [110]) nucleotides long. Second, they can cover larger regulatory regions with many adjacent sequence windows having high scores. This makes the smoothing of sum-scores valuable. However, small sequence windows mean overall smaller sum-scores and cut-offs, since fewer motifs fit into such windows. This is expected to be a disadvantage, since high-scoring windows will not easily rise above the noise level.

In Figure 4.8, cut-offs for different widths of the sequence window are shown. As expected, for the same window width, smoothing in general lowers the cut-off. The cut-offs are calculated for two fixed-size K's, $K = 11$ and $K = 17$, and three variable-sized K's. The varying K's are calculated from the width of the sequence window:

$$K = \left(\left\lfloor \alpha \cdot \frac{width}{shift} + 1 \right\rfloor \& \sim 1 \right) + 1 \,. \tag{4.1}$$

The term $(\ldots \& \sim 1) + 1$ makes sure that each K is odd (by switching the last bit off, and adding one to the then even number). The factor α is 0.5 for the variable sized K's named "One half", 1 for "Double" and 2 for "Triple". The names correspond to the overall width of the smoothing window. For instance, a window width of 500 corresponds to a K of 27, 49, and 101, respectively. And calculating the mean over 101 adjacent sequence windows, each shifted by 10, gives an overall window width of $101 \cdot 10 + 500 \approx 1500$, which triples the original width. Note that for "Double", the 1 is not added before rounding down, but subtracted (Equation 4.1). This change leads to appropriate K's for very small window sizes.

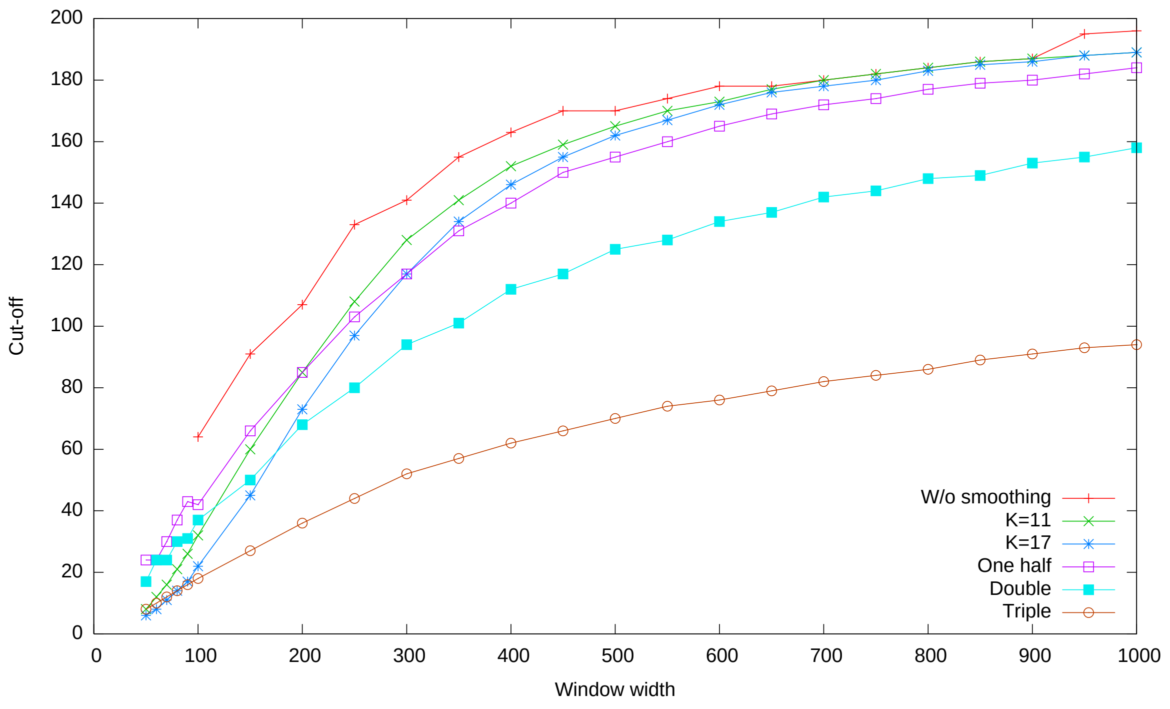


Figure 4.8: Cut-offs for different sequence window widths and different smoothing strategies applied to sequence window scores. All cut-offs correspond to an *E*-value of 1 and were calculated under the following terms: “SP1” motif set, PREs vs. promoters training set, sequence window shifted by 10, 11.7 Gbases of shuffled-out *D. melanogaster* genome. Smoothing means calculating the mean over the scores of *K* adjacent sequence windows. “One half”, “Double” and “Triple” have *K*’s which increase with higher window width (see text).

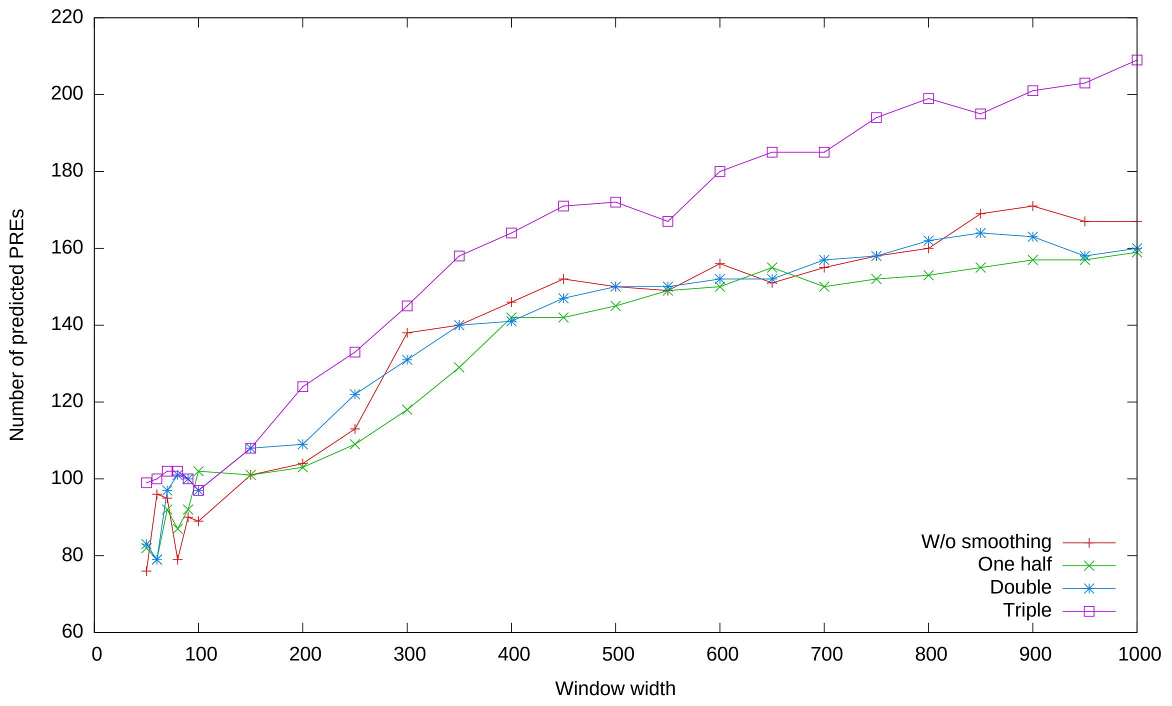


Figure 4.9: Number of PREs predicted in the *D. melanogaster* genome dependent on the window width and certain smoothing strategies. The cut-offs for the prediction correspond to an *E*-value of 1 and were calculated under the following terms: "SP1" motif set, PREs vs. promoters training set, sequence window shifted by 10, 11.7 Gb of shuffled-out *D. melanogaster* genome.

For large sequence windows a smoothing with fixed-sized K does not result in a reduced cut-off. The reason is that the relative impact of 10 exchanged nucleotides (a loss of 10 nucleotides at the beginning, and an additional 10 nucleotides at the end) is smaller the larger the windows are. In addition, small high-density areas of motifs are covered by more adjacent windows if they are larger, resulting in such areas to survive the smoothing, if K is relatively small. An example might make this clear. A small high-density area of size 250 is fully covered by 25 adjacent sequence windows if the window width is 500, in comparison to only 5 if the width is 300. With $K = 17$ and a width of 300 the area will not survive the smoothing, whereas for a width of 500 and higher the area is not smoothed out.

All three variable-K strategies led to lower cut-offs in comparison to without smoothing (Figure 4.8). The lowest cut-offs are obtained with the most extensive smoothing. But the question is, how does such smoothing affect the prediction of PREs?

In Figure 4.9, the prediction of PREs in the *D. melanogaster* genome is shown, dependent on window width and three varied-K smoothing strategies. In general the number of predicted PREs increases with increasing the window width. Even for small window widths up to 150 candidate PREs are successfully predicted. Nevertheless, there is no difference in the applied smoothing strategy for such small widths. Thus, these regulatory elements must be much longer.

In comparison to omitting a smoothing, the "One half" as well as the "Double" smoothing strategy has lower cut-offs (Figure 4.8). Nevertheless, in the prediction process, the lower cut-offs do not result in more predicted PREs (Figure 4.9). Only in applying the “Triple” strategy, the very low cut-offs pay off, resulting in up to 30% more predicted PREs.

4.8 Discussion

In this chapter the complex prediction pipeline was analyzed with the task in mind to increase the number of predicted PREs, while maintaining a high specificity. Several statistical considerations make the task harder, since the question needs to be answered whether a prediction can be trusted or not.

The first task is always to choose motifs that are significant for the elements to be predicted. In general, the more significant the used motifs are the higher the number of predicted regulatory elements. The significance of a motif can be judged in comparison to a background. Weighting motifs is a first indicator of a motif’s significance, because enrichment or avoidance of a motif can be recognized. Enrichment in the positive training set in comparison to the negative training set leads to high positive weights, avoidance leads to negative weight. Combining the single motifs to double motifs (a motif pattern containing two single motifs with a distance constraint in-between them) and weighting the double motifs gives insights not only into whether the motifs are enriched/avoided but also into whether the motifs occur closely together. Such double motifs are only found if both comprised single motifs occur close to each other. Judging double motifs by the weight, the most important motif for PRE prediction is the Pho motif. It is assigned very high weights not only as a single motif [5] but also as a double motif in combination with all other motifs. In the latter sense, other important motifs are En1 and SP1. Contrary to this high weighting motif, both the long (G10) and short (GA) GAF motif yield negative weights. The reason is that motif pairs composed of these two motifs occur rarely in the positive training set, but frequently in the negative training set. Nevertheless, combined with other motifs than themselves both GAF varieties yield high weights showing that co-operation and teamwork of different binding factors are very important for the functionality of a PRE.

The background model in order to obtain a cut-off is important for statistical evaluation. The more the background model resembles the target genome the higher the cut-offs for the same E-value will become. In this thesis the target genome is from *Drosophila melanogaster*. The consequence of a higher cut-off is a smaller number of predicted elements. But what are characteristics of the genome a randomly generated sequence lacks? From the genetic point of view, a random sequence lacks genes, regulatory elements, repeats, and low-complexity regions. Such elements resemble the semantic of the genome and are built upon shorter units that have to occur in the correct order and with a certain arrangement. Such short units are motifs, exons, nucleotide boxes, start sites, and so on. From the point of view of the prediction of PREs, a random sequence should conserve the motifs as much as possible while disrupting the structure of a regulatory element. This would lead to the correct number of motifs in the background model, while clusters and assemblies of motifs occur merely by chance. Markov chains of low order cannot preserve the number of motifs, because the order of the chain is normally shorter than the length of the motif, and therefore the motifs occur by chance. However, with the hexanucleotide 5th order Markov chain suggested by Thijs et al. [88], the number of motifs of length 6 and less is preserved, if the model was trained on the target genome. Such a model has already $4,096$ parameters. The advantage of shuffling the source genome is that it can preserve the number of motifs very well. This was shown by calculating the sum over the log-odd-scores for all motifs of length 5 and 6 in comparison to the target genome. For the generation of random sequences it is therefore strongly recommended to replace Markov chains by shuffling the genome. This produces sequences most similar to the target genome in terms of motif occurrences.

In the prediction process, several motif sets (a list of single motifs combined to double motifs) were used for the prediction of PRE/TREs. Such motif sets differ in only few motifs. In general, the larger the motif set was the higher became the cut-off (corresponding to an *E*-value of one), first, due to more double motifs contributing to the score of a sequence window, and second, due to the higher chance of single motifs occurring close together. When using the same motif set, a higher cut-off would reduce the number of predicted elements, because fewer sequence windows exceed the cut-off with their scores. Adding a motif to an already existing motif set increases the cut-off as well. Two cases are now possible. If the number of predicted elements increases despite the higher cut-off, the motif seems to occur frequently near the other motifs. In this case, the double motifs comprising the new motif add to the score of a sequence window in a way that more sequence windows exceed the cut-off. The addition to the scores of sequence windows is higher than the addition to the cut-off. This leads to the conclusion that the motif is important for the predicted elements and, moreover, to their functionality. The other case is when adding a motif to a set causes a drop in the number of predicted elements. The new motif seems to not occur near the other motifs from the set. This leads to the conclusion that the new motif has no impact on the functionality of the predicted element.

In the performed PRE prediction on a genome-wide level, adding SP1 to the DSP1 motif set increased the number of predicted elements from 481 to about 700. Therefore, SP1 is very important for PRE functionality. This supports the findings of Brown et al. [9], now not only limited to the analyzed *engrailed* gene but genome-wide. Grainyhead, on the other hand, was proposed to interact with Pleiohomeotic [11]. This cannot be confirmed, because in the prediction process adding Grh to the SP1 motif set leads to a drop in the number of predicted PREs by about 100. It is probable that only the Grh protein is involved in PRE functionality and not the corresponding binding site.

Two robustness tests were performed. For motifs added to the sets in the context of a robustness test, the number of predicted elements should not increase. Consistently, adding Knirps to the SP1 motif set led to a drop in the number of predicted elements. Knirps is unrelated to PRE functionality and the test was passed for this motif. On the other hand, adding HB to the SP1 motif set increased the number of predicted PREs by about 200.

As a consequence, concluding that SP1 is important for PRE/TRE functionality imposes the question whether HB is somehow important, too, and if not, why not. The first possibility is that, by including HB into the SP1 motif set, additional enhancer elements are predicted. This is supported by the fact that Pho was reported to occur in enhancer elements [111]. Nevertheless, the sole presence of Pho does not give the high scores necessary to exceed the cut-off. The other motifs must be present as well. The second possibility is that PREs are predicted that contain many HB binding sites. Mahmoudi et al. [80] showed that binding sites for hunchback occur frequently in the *Ubx* PRE. The authors suggest a potential role for HB, not only during the initiation of *Ubx* repression, but also during the transition from establishment to maintenance. This transition seems to involve dMi-2 recruitment by HB [112]. Other authors show that Hunchback is an early regulator for some PRE/TREs [113]. The results from the genome-wide prediction support the assumption that HB plays an important role in PRE functionality. However, it is noteworthy that Poux et al. [114] were able to establish HB-independent PcG repression of the *Ubx* promoter. But this does not contradict the importance assumption. It merely shows that the role of HB during the transition from establishment to maintenance can be adopted by other factors as well.

All in all, the prediction pipeline knows many variables, and all such parameters influence the prediction results to a certain extend. The most promising prediction uses a window width of 500, the “SP1” motif set, the PREs vs. promoters training set, and a cut-off obtained with the shuffled-out *D. melanogaster* genome as background without smoothing. This results in 150 PREs, which need to be checked for their biological relevance.

5 Motif discovery by evolutionary means and motif clustering

In this chapter a motif discovery approach for the de-novo identification of over-represented motifs is presented. It was developed foremost for the identification of double motifs in the training sets used in the prediction of PREs. The goal was to identify double motifs and to repeat the genome-wide prediction of PRE/TREs in order to find sequences similar to the already confirmed PREs from the positive training set. In addition to the motif discovery, a motif clustering approach is presented in this chapter. It was developed to subsequently reduce the number of de-novo discovered motifs, but can be applied to any number of motifs with any order and complexity. The clustering is based on aligning motifs gap-free and also on calculating the distance between the motifs using this alignment.

Both, the motif discovery and the clustering of motifs were implemented into the *jPREdictor*. The reason was that this allowed for the reuse of all classes and methods dealing with motif definition, sequence reading, and motif weighting. However, plans exist to source out the discovery and clustering from the original *jPREdictor* and to incorporate them into a new program called *jEvolver*. While this is not a difficult task, it is time-consuming, especially the development and implementation of adjusted (graphical) user interfaces. As a consequence, the details on function and implementation of motif discovery and clustering are presented in this chapter, and not in Chapter 3.

5.1 Problem definition

The task of the motif discovery approach presented here is to identify an unknown motif over-represented in the sequences of the model with respect to the sequences of the background. The objective function to maximize is the weight of the motif, defined as the normalized log-ratio between the number of matched binding sites in the model versus the number of sites in the background (see Equation 3.4 on Page 43). This task is addressed with an evolutionary approach.

Some characteristics of the motifs used in this work make the situation for a motif discovery unique. While all presented motif discovery approaches strive to find optimal single motifs based on certain objective functions, the evolution built into the *jPREdictor* is able to evolve not only basic motifs but also motif patterns. Therefore, it is possible to obtain, for instance, double motifs with high weights, which fit into the prediction process presented in Chapter 4. For all motifs, a high degree of degeneration is allowed. In addition, the error number allowed for a match is subject to evolution as well. Such characteristics increase the search space by a great amount, which is the reason, why enumeration approaches cannot be applied. Nevertheless, the main design reason behind the evolution built into the *jPREdictor* was the discovery of high-weighting double motifs.

Some restrictions to motifs and the complexity of motifs exist, which makes them more practical to handle. The length of the motif can vary in the limits of 5 to 10 nucleotides. It was chosen in accordance to the length of most motifs presented in this work, i.e. Zeste, the core Pho, DSP1, and the two GAF motifs. Longer motifs are available by using the basic motifs in motif patterns with a small maximal distance. Both evolution strategies are capable of evolving basic motifs as well as double motifs. Basic motifs are regular expression motifs. Double motifs comprise two such motifs. No restriction exists to the number of degenerated positions, nor to the amount of degeneration. All letters of the IUPAC code are available for mutation. The distance between two basic motifs comprised in one double motif is restricted to a maximum of 440. This is twice as much as the distance used in the PRE/TRE prediction.

One run of the motif discovery based on motif evolution already yields 20 motifs with high weights. These motifs are never identical. Invoking the motif evolution a number of times increases the number of high-weighting motifs significantly. Even after removing duplicates, the number stays high. In order to reduce the number of motifs to a manageable degree, a motif clustering was implemented into the *jPREdictor*. Since the number of clusters is unknown beforehand, a hierarchical clustering is performed. The task behind is to calculate the distance between each two gaplessly-aligned motifs, and to combine the motifs with the least distance into one cluster (agglomerative approach). Then, a consensus motif for the cluster is calculated, which acts as representative. What makes the task interesting, and more complicated, too, is that the clustering has to deal with different motif representations as well as different orientations of motifs. In addition, clustering should not only function with basic motifs, but also with motif patterns. This was also the reason, why the clustering was implemented into the *jPREdictor*. The full algorithm is explained in Chapter 5.3.

5.2 Evolution strategy with and without rewarded selection

Two evolution strategies are built into the *jPREdictor*, a basic and an extended one. The basic one is abbreviated “ES” (evolution strategy), the extended one is named “RewardedSelection”. Both can be invoked via command-line interface using one of the following special parameter:

```
--motifEvolutionES
--motifEvolutionRewardedSelection
```

The evolution strategy implemented into the *jPREdictor* is $(n/2+5n)$. This annotation describes the following steps. With the beginning of each cycle, the parental generation contains n motifs. In the evolution step, two of them are randomly selected, recombined, mutated, and afterwards added to the offspring generation. This is repeated $5n = 100$ times. In the evaluation step the offsprings are weighted. Then, in the selection step, parents and children are mixed and the best-weighting n motifs are chosen to make up the next generation. The *jPREdictor* works with $n = 20$. This procedure is repeated a number of times, per default 1000. The number of cycles can be set via command-line parameter:

```
-c 1000
```

Simulated annealing is incorporated into the cycles to allow jumps over the search space in order to cover it as much as possible and to allow for suboptimal motifs to survive into the next generation. The starting temperature of the annealing process is the number of cycles to perform. The end temperature is always zero. In every cycle the temperature is reduced by one. The first task of search space covering is achieved by coupling the number of mutations introduced from parent to offspring with the temperature. The formula, which gives the number of mutations, is $1 + \lfloor log_{10}(temperature + 1) \rfloor$. The minimum number of mutations is one for temperatures below 10. When motifs are strongly mutated they may get assigned a low weight, which would prevent them from surviving the subsequent selection step. But the possibility exists that this motif has just discovered a higher hill to climb, but has landed on the base. Thus, the motif must be allowed to survive the selection step and to be part of the next parental generation. Therefore, before the selection step is performed, the weight of every motif of parental and offspring generation is increased by a random number. The numbers are uniformly distributed in the range zero to a maximum number r, which is calculated by incorporating the temperature:

$$r = \left| \max_M \{w(M)\} - \min_M \{w(M)\} \right| \cdot \frac{\sqrt{temperature}}{5}, \tag{5.1}$$

with $w(M)$ being the weight of the motif. For high temperatures, r is large and for small temperatures below 20, r is small. However, r only amplifies the range of weights already there. The existence of large differences in the motif weights supports the low-weighting motifs, because r becomes large and their chance to overpower the high-weighting ones is increased.

The actions performed in the evolution step depend on the motif type. Double motifs are recombined before they are mutated. Non-pattern motifs are directly mutated. For the recombination, two motifs are randomly chosen from the parental generation, and from each motif, one basic motif is drawn. Together they build the new double motif. The distances between the two basic motifs are also drawn completely at random from the parents. The recombination step is repeated 100 times, and afterwards, all 100 double motifs are mutated. For non-pattern motifs, a motif is repeatedly drawn from the parental generation and mutated.

Mutation events mainly affect the sequence of regular expression motifs. Mutation events are mutually exclusive and can only happen simultaneously to a basic motif, if the number of mutations is greater one. The first mutation event is length alteration, either at the end or at the beginning. The motif is either shortened by one position, or elongated. In case of elongation, a randomly chosen nucleotide is introduced. The second mutation event is applied to the error number allowed for a match. This number is switched between zero and one. The third mutation event affects base alterations. One position in the motif is randomly chosen, and either increased or decreased in its degeneration level, or the nucleotide is replaced. An increase in degeneration level reflects the mutation of one binding site, while other binding sites are not changed. Nevertheless, the change does not cause a complete lose in binding affinity. Decreasing the degeneration level reflects a change in the binding factor, which is more selective and does not bind to all sites anymore. To increase the degeneration level bases are added to the position, e.g., adding an A to a C yields a M, and adding a T to the M yields an H. Reducing the degeneration level acts vice versa.

For double motifs, an additional characteristic under mutation is the distance between the two motifs. Either the minimum distance or the maximum distance is mutated, both in the range $[0, 440]$. In order to change the distance a random number from a normal distribution $N(0, 10)$ is drawn and added to the distance. If the minimum is no longer smaller than the maximum, both distances are exchanged.

The mutation events have certain probabilities to occur. If one mutation event is incurred into a motif, the probability that this event has changed the motif in length is 5%. With 2%, the number of errors allowed for a match is altered. In case of a double motif, another 5% are allotted to distance mutations. As a consequence, for double motifs, nucleotide alterations have an 88% probability. For basic motifs, the 5% probability for distance mutations is divided between the other probabilities on equal terms. The *jPREdictor* allows the user to change the probabilities and even to switch recombination on or off. The command-line parameter is:

```
-C length,distance,error,nucleotides,recombination
```

The values given for length, distance, error, and nucleotides, can have an arbitrary range. They are recalculated to probabilities. Negative values are assumed to be zero. The value for “recombination” is either zero or one to switch it off or on, respectively. The settings can become quite handy, if only single motif characteristics should be under evolution. In order to evolve the distances between already known double motifs, distance mutations should have a probability of 100%, and recombination events have to be switched off:

```
-C 0,100,0,0,0
```

Another application would be the discovery of a basic motif with prior knowledge about its length. In this case, the motif must be given as a pattern to determine the length (e.g. 7), and double motif creation must be switched off. In addition, length alterations have to be disabled:

```
-d NNNNNNN -p single -C 0,0,5,95
```

In the selection process, the 20 highest-weighting motifs are chosen to form the parental generation for the next evolutionary step. As was already discussed, simulated annealing affects the weights to introduce a random element into the selection. The procedure “RewardedSelection” changes the selection process once more in order to balance out the number of binding sites over the model sequences. The normal “ES” strategy does not incorporate the balancing. “ES” should only be used, if great uncertainty exists about the input sequences carrying the binding site. Nevertheless, balancing out does not mean that all model sequences are forced to contain a specific amount of binding sites. Motifs occurring everywhere, or almost everywhere are rewarded and survive with greater probability. Motifs, which already occur frequently, and therefore have high weights, survive regardless.

The rewarding system uses the information entropy to calculate a factor, which is afterwards multiplied to the weight. The formula for the entropy factor is

$$f_M = -\sum_{i=1}^{k} p_{M,i} \cdot ln(p_{M,i}) , \tag{5.2}$$

with k being the number of sequences in the positive training set (model) and $p_{M,i} = \frac{fr_{M,i}}{\sum_{j=0}^{k} fr_{M,j}}$ being the probability mass function of the occurrences of motif M in the model sequences. In Figure 5.1,

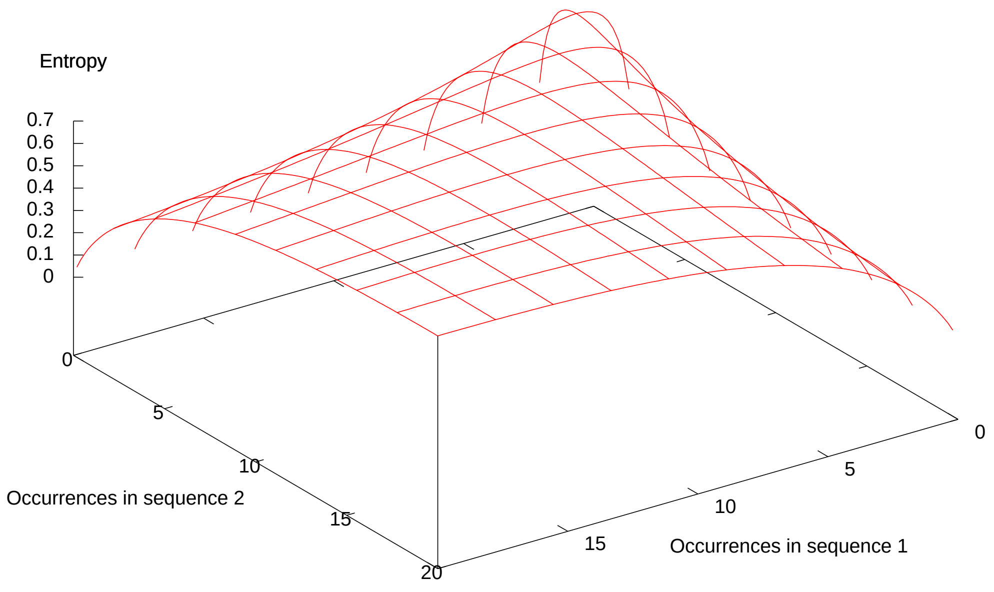


Figure 5.1: Entropy calculation in a system with two random variables. In this example, the variables are the occurrences of a motif in two sequences. The occurrences are recalculated to probabilities via probability mass function. These probabilities are used in calculating the information entropy afterwards.

Table 5.1: Influence of a rewarded selection step in the evolution on the number of occurrences of discovered motifs (M1 to M5) in sequences of the positive training set (model seq 1 to 12). The motifs were previously discovered running the motif evolution in either “ES” or “RewardedSelection” approach. The rewarded selection algorithm alters the weights before the selection step by multiplying them with the information entropy factor. The maximal information entropy is 1.08 (column “Opt”).

	ES					Opt	Rewarded Selection				
	M1	M2	M3	M4	M5		M1	M2	M3	M4	M5
Model seq 1	16	0	3	0	0	1	15	1	5	4	0
Model seq 2	0	2	7	0	0	1	5	3	4	0	1
Model seq 3	0	0	0	2	1	1	0	1	0	4	1
Model seq 4	6	1	0	3	4	1	7	2	0	0	3
Model seq 5	0	2	9	0	0	1	6	0	0	21	4
Model seq 6	12	0	0	0	0	1	0	9	10	1	3
Model seq 7	1	0	0	1	0	1	0	0	0	0	6
Model seq 8	5	11	0	3	0	1	4	12	17	3	1
Model seq 9	0	0	0	1	0	1	0	0	0	2	2
Model seq 10	0	18	0	0	65	1	6	12	15	6	6
Model seq 11	0	1	0	2	0	1	4	3	3	5	0
Model seq 12	0	0	0	0	0	1	2	1	1	9	0
Entropy	0.59	0.54	0.44	0.74	0.13	1.08	0.84	0.78	0.72	0.81	0.87

the information entropy for a model containing only two sequences is shown. This entropy factor maximizes for one motif, if its occurrences in all model sequences are equal. The influence of the rewarding system on the discovery of motifs can be seen in Table 5.1. Most motifs obtained by applying the “ES” strategy yield low entropy factors, because they are present in only few model sequences. In comparison, the motifs obtained by applying the “RewardedSelection” strategy occur more uniformly and equally distributed over the sequences of the model, and therefore have high entropy factors. However, rewarding balanced-out motifs does not guarantee that the motifs occur in all sequences. This is an advantage over other motif discovery strategies, especially probabilistic ones like Gibbs sampling.

As a consequence of both rewarding system and simulated annealing the motif’s weights are changed. The selection step chooses the 20 highest-weighting motifs always after all changes are incorporated into the weights. The final weight $w'(M)$ of a motif M in the “ES” strategy, thus without rewarding factor, is calculated as:

$$w'(M) = w(M) + F(0,r)\,. \tag{5.3}$$

The complete formula including both rewarding factor and random number from the annealing process looks like this:

$$w'(M) = w(M) + F(0,r) + w(M) \cdot f_M = w(M) \cdot (f_M + 1) + F(0,r)\,. \tag{5.4}$$

In both equations, $F(0,r)$ specifies the uniform distribution, from which a random number is drawn. Note that both weight changing processes are applied to the original weight. This is the reason, why the complete formula looks a little bit strange. First, the random number from the simulated annealing is added, and in the second step, the original weight is rated with the rewarding factor and added as well.

One cycle in the evolutionary process consists of mutating, weighting, and selecting. With every cycle, the temperature is reduced by one. If it reaches zero, the evolution ends and the 20 highest-weighting motifs are printed out by the *jPREdictor*. The results produced by the evolution also depend on the initial generation. Two ways exist to set the initial generation, either by letting it be created randomly, and by providing it to the program. If no motif is given to the *jPREdictor* (use command-line parameter "–G" to prevent an accidental start of the evolution with the built-in motifs), double motifs are generated randomly. For the evolution of basic motifs, one or more template motifs are always necessary, which either consist of all *N* nucleotides or which are randomly generated outside the *jPREdictor* program.

5.3 Motif clustering

An agglomerative (bottom-up) and hierarchical clustering is performed over all motifs provided via option file. An example for a dendrogram formed after a complete clustering can be seen in Figure 5.5. Note that the clustering performed is bottom-up, even if the term in this example is unsuited, since the unclustered motifs are not arrayed at the bottom, but at the top. The command-line parameter to invoke a clustering is:

```
--clusteringXxxxYyyy
```

Xxxx specifies the distance metric, and Yyyy identifies the clustering approach. If the number of clusters is known beforehand, use parameter '-C' to provide it to the program. Parameter '-c' defines a threshold to stop the clustering, when the optimal distance exceeds this value. If both parameters are omitted a full clustering is performed, yielding only one super cluster.

Each basic motif to be clustered is represented internally as a PSPM. Consequently, motif patterns contain two or more such PSPM motifs as well as minimal and maximal distance in-between each two. Accessing a single motif out of a pattern and the values in its two-dimensional matrix is done in the form $m_{k,l,n}$, where k represents the motif order, l a position and n the value within the vector. If single motifs are clustered, k is constant with $k = 0$. Indices can be omitted to refer to the motif alone (only k is given), or a special position within the motif (k and l are given). Counting for all three indices starts with zero.

5.3.1 Distance measure

In order to cluster motifs correctly a relation measure between two motifs must be defined. The *jPREdictor* provides several relation measures. Using distances is one possibility. Given an alignment (explained below) for two double motifs, the distance can easily be calculated as sum of sum over each two positional vectors. The shift value represents an alignment without gaps and defines the number of nucleotides the first single motif is shifted against the second one. Since double motifs consist of two single motifs, two shift values must be defined for an alignment between two double motifs. A

$$
\boxed{
\begin{array}{ccccccccccccccc}
\multicolumn{5}{c}{shiftvalue = -2} & \multicolumn{4}{c}{shiftvalue = 0} & \multicolumn{5}{c}{shiftvalue = 1} \\
A & C & G & T & n & A & C & G & T & n & A & C & G & T \\
| & | & | & | & | & | & | & | & | & | & | & | & | & | \\
n & n & G & T & A & G & T & A & n & G & T & A & n & n
\end{array}
}
$$

Figure 5.2: Gapless alignment of two nucleotide sequences (*ACGT* and *GTA*) using the shift value, which defines the number of nucleotides the first motif is moved against the second. The *n* letter defines a position, where all nucleotides are possible to occur.

shift value of zero means, both single motifs start at the same position. For values smaller than zero the first motif starts earlier than the second (see Figure 5.2).

The distance value between two single motifs m_x and m_y is calculated with the following equation:

$$
dist_{shift}(m_x, m_y) = \sum_{i=min(0,shift)}^{max(|m_y|,shift+|m_x|)} dist_{nuc}\left(\begin{cases} m_{x,i-shift} & i \geq shift \wedge i < shift + |m_x| \\ N & otherwise \end{cases}, \begin{cases} m_{y,i} & i \geq 0 \wedge i < |m_y| \\ N & otherwise \end{cases} \right) \tag{5.5}
$$

where N denotes a vector of background probabilities for all nucleotides of the alphabet. Indices to denote the positional vectors start with zero, and thus are always less than the length of the motif. The $dist_{nuc}$ function is defined for two positional vectors the size of the alphabet Σ:

$$
dist_{nuc}(p, q) = \sum_{i=0}^{|\Sigma|-1} |p_i - q_i|^a . \tag{5.6}
$$

In case of $a = 1$, the simple distances over every pair of values is summed up. This is called the Manhattan distance. For the Euclidean distance to be calculated, $a = 2$. In both cases, the distance measure has a range of $[0, \infty]$ and if comparing two distances, the minimal value is the optimum.

5.3.2 Likelihood measure

Another way to express the relationship between two motifs would be using a likelihood measure. The value expresses, how well one matrix explains the other one, and vice versa. It is calculated similar to the distance measure, but multiplies over probabilities:

$$
L_{shift}(m_x, m_y) = \prod_{i=min(0,shift)}^{max(|m_y|,shift+|m_x|)} conf_{nuc}\left(\begin{cases} m_{x,i-shift} & i \geq shift \wedge i < shift + |m_x| \\ N & otherwise \end{cases}, \begin{cases} m_{y,i} & i \geq 0 \wedge i < |m_y| \\ N & otherwise \end{cases} \right) \tag{5.7}
$$

The used function $conf_{nuc}$ is defined similar to the $dist_{nuc}$ function, but adds products and not distances:

$$conf_{nuc}(p,q) = \sum_{i=0}^{|\Sigma|-1} p_i \cdot q_i \,. \tag{5.8}$$

It is possible that $conf_{nuc}$ yields a zero value if one or both factors are zero. In this case, the pseudo count value 0.01 is taken instead, which prevents L_{shift} from becoming zero overall. The likelihood measure has a range of $[0,1]$ and is often used as a log-likelihood with the new range $[-\infty,0]$. In both cases, the optimal value when comparing two likelihoods is the maximum.

The likelihood of one motif to express another one is given by the above equation for L_{shift}. Assuming both matrices to be of the same length, and to be aligned without shift gives a much simpler equation:

$$L(m_x,m_y) = \prod_{i=0}^{|m_x|-1} \sum_{j=0}^{|\Sigma|-1} m_{x,i,j} \cdot m_{y,i,j} \,. \tag{5.9}$$

where $|\Sigma|$ is the size of alphabet, 4 in the case of DNA bases. This approach has a complexity of $|m_x| \cdot |\Sigma|$. A much more straight-forward approach is working with sequences. This is much more intuitive, because a motif is the representation of binding sites. Binding sites are simple sequences and not as complex as motifs. Therefore, the best would be to generate all possible binding sites for one motif, and afterwards add the probabilities that they can be generated with the second motif. This should give the likelihood for the first motif under the second. More formally, generate all possible sequences using matrix m_x, $S \leftarrow m_x$, and then add their probabilities under the matrix m_y, normalized with the occurrence probability of their creation. The equation for this is

$$\begin{aligned} p(m_x|m_y) &= \sum_{S \leftarrow m_x} p(S|m_y) \cdot p(S|m_x) \\ &= \sum_{S \leftarrow m_x} \prod_{i=0}^{|S|-1} m_{y,i,S_i} \cdot m_{x,i,S_i} \,. \end{aligned} \tag{5.10}$$

The term m_{y,i,S_i} refers to motif matrix m_y, positional vector i, and nucleotide S_i within i, for m_{x,i,S_i} accordingly. This approach has a complexity of $|\Sigma|^{|m_x|} \cdot |m_x|$. The complexity is much higher than calculating L. Now, it should be proven that this sequence generating approach is interchangeable with the likelihood calculation, and that they give the same result: $L(m_x,m_y) = p(m_x|m_y)$. The proof can be found in Figure 5.3. It starts with $p(m_x|m_y)$ and transforms this into $L(m_x,m_y)$. First, a generalization is made. The sum then runs over all possible sequences of length $|m_x|$, not only the ones that can be created from matrix m_x. This presents no problem, because the probability for a sequence that is not creatable under m_x is zero at one or more positions and thus the overall product turns zero as well. At line three sequences are expressed as single nucleotides, as references into the alphabet. Line four and five show how different sums can be treated as constants under the former sums, and thus can be moved to almost every position in the sum chain. This reflects the independence of positional order. At the last three lines, this independence is utilized to obtain a product of every positional sum, which eventually leads to the likelihood function, thus giving $L(m_x,m_y) = p(m_x|m_y)$.

$$
\begin{aligned}
p(m_x|m_y) &= \sum_{S \leftarrow m_x} \prod_{i=0}^{|S|-1} m_{y,i,S_i} \cdot m_{x,i,S_i} \\
&= \sum_{S} \prod_{i=0}^{|S|-1} m_{y,i,S_i} \cdot m_{x,i,S_i} \\
&= \sum_{n_0=0}^{|\Sigma|-1} \sum_{n_1=0}^{|\Sigma|-1} \cdots \sum_{n_{|S|-1}=0}^{|\Sigma|-1} \prod_{i=0}^{|S|-1} m_{y,i,n_i} \cdot m_{x,i,n_i} \\
&= \sum_{n_0=0}^{|\Sigma|-1} \sum_{n_1=0}^{|\Sigma|-1} \cdots \sum_{n_{|S|-1}=0}^{|\Sigma|-1} \underbrace{m_{y,0,n_0} \cdot m_{x,0,n_0} \cdot m_{y,1,n_1} \cdot m_{x,1,n_1} \cdot \ldots}_{\text{constant under the last sum over } n_{|S|-1}} \cdot \\
&\qquad m_{y,|S|-1,n_{|S|-1}} \cdot m_{x,|S|-1,n_{|S|-1}} \\
&= \sum_{n_0=0}^{|\Sigma|-1} \cdots \sum_{n_{|S|-2}=0}^{|\Sigma|-1} m_{y,0,n_0} \cdot m_{x,0,n_0} \cdot \ldots \cdot m_{y,|S|-2,n_{|S|-2}} \cdot m_{x,|S|-2,n_{|S|-2}} \cdot \\
&\qquad \underbrace{\left(\sum_{n_{|S|-1}=0}^{|\Sigma|-1} m_{y,|S|-1,n_{|S|-1}} \cdot m_{x,|S|-1,n_{|S|-1}} \right)}_{\text{constant under every sum before}} \\
&= \left(\sum_{n_0=0}^{|\Sigma|-1} m_{y,0,n_0} \cdot m_{x,0,n_0} \right) \cdot \left(\sum_{n_1=0}^{|\Sigma|-1} m_{y,1,n_1} \cdot m_{x,1,n_1} \right) \cdot \ldots \cdot \\
&\qquad \left(\sum_{n_{|S|-1}=0}^{|\Sigma|-1} m_{y,|S|-1,n_{|S|-1}} \cdot m_{x,|S|-1,n_{|S|-1}} \right) \\
&= \prod_{i=0}^{|S|-1} \sum_{n_i=0}^{|\Sigma|-1} m_{y,i,n_i} \cdot m_{x,i,n_i} \\
&= \prod_{i=0}^{|m_x|-1} \sum_{j=0}^{|\Sigma|-1} m_{x,i,j} \cdot m_{y,i,j} \qquad = L(m_x, m_y)\,.
\end{aligned}
$$

Figure 5.3: Proof of $p(m_x|m_y) = L(m_x|m_y)$. m_x and m_y are both matrix motifs of type PSPM. The equations calculate the probability of matrix m_x under the given matrix m_y. $p(m_x|m_y)$ enumerates over all possible sequences generated from m_x and $L(m_x|m_y)$ directly multiplies the matrix values. For further explanations see text.

5.3.3 Alignment

Aligning two motifs means the optimization of their relation measure. An alignment of two single motifs is defined by a *shift*-value, which expresses how the first motif is shifted against the second one. This alignment has to be gapless. An example can be seen in Figure 5.2. The pre- and post-matching sites that miss a corresponding partner are paired with either the *N* nucleotide (equal-distributed probability for every nucleotide) or another probability vector. Per default, the *jPREdictor* uses the background probability vector. The equation for distances is the following:

$$shift_{m_x,m_y} = argmin_{shift \geq -|m_x|, shift < |m_y|} \{dist_{shift}(m_x, m_y)\} . \tag{5.11}$$

Since likelihoods have to be maximized, the equation looks like this:

$$shift_{m_x,m_y} = argmax_{shift \geq -|m_x|, shift < |m_y|} \{L_{shift}(m_x, m_y)\} . \tag{5.12}$$

Motif patterns are aligned basic motif by basic motif. Thus, two motif patterns should only be aligned if they comprise the same number of basic motifs. Each pair of basic motifs is aligned on its own yielding a *shift*-value which is only valid between the two basic motifs. Consequently, alignments of motif patterns consist of two or more alignments of basic motifs. In case of distances, the results from these basic motif alignments are added up to obtain the alignment value for the two motif patterns. In case of likelihoods, the results from all basic motif alignments are multiplied.

5.3.4 Consensus motif

In the clustering process two motifs with minimal distance or maximal likelihood must be combined to form a new motif, the consensus motif, which, at the same time, represents the cluster. In order to create the consensus motif the *shift* value for the motif pair has to be known. Given the *shift* value a new single motif m_z is built from the two basic motifs m_x and m_y using this formula:

$$\bigcap_{i \geq min(0,shift), i < max(|m_y|, shift+|m_x|)} m_{z,i-min(0,shift)} \leftarrow mean_{nuc} \left(\begin{cases} m_{x,i-shift} & i \geq shift \wedge i < shift + |m_x| \\ N & otherwise \end{cases} , \begin{cases} m_{y,i} & i \geq 0 \wedge i < |m_y| \\ N & otherwise \end{cases} \right) \tag{5.13}$$

It is actually nearly the same formula as calculating the distance between two motifs. The first difference is that the function $mean_{nuc}$ is called instead of function $dist_{nuc}$. The second difference is that the result of the function call is not used for calculation, but rather it is assigned to build up the new motif. The function $mean_{nuc}$ builds a new positional vector from two given ones by calculating the mean:

$$mean_{nuc}(p,q) = \frac{p+q}{2} . \tag{5.14}$$

When a motif is fitted into an existing cluster or when two clusters are combined, calculations are done with the respective representative of the clusters. In this case, the function $mean_{nuc}$ is changed

Initialization

- add motifs into a list
- calculate distance or likelihood between each two motifs

Loop

- check break condition (number of clusters, cut-off) and break accordingly
- search motif pair with minimal distance/maximal likelihood
- remove both motifs from the list
- generate consensus motif
- add consensus motif to the list
- calculate distance or likelihood between consensus motif and all remaining motifs
- repeat loop until list contains exactly one motif

Afterwork

- print the consensus motifs of all remaining clusters
- generate dendrogram

Figure 5.4: The motif clustering algorithm as implemented into the *jPREdictor*. The distance measures are either Manhattan, Euclidean, or Likelihood. While the first two have to be minimized between motifs in order to form a cluster, the likelihood measure is to be maximized. The break condition is checked before the first clustering step in order to avoid a clustering, if the user-given requirements are not met.

in a way to normalize the two vectors by the cluster size:

$$mean_{nuc}(p,q,c_p,c_q) = \frac{p \cdot c_p + q \cdot c_q}{c_p + c_q}. \tag{5.15}$$

Thus, the new consensus motif is truly the average motif over all motifs of one cluster.

The consensus motif for two motif patterns is obtained accordingly. Equation 5.13 is applied to each aligned pair of basic motif. The new consensus motif obtains the mean distances between the basic motifs.

5.3.5 Clustering algorithm

The clustering algorithm has three steps: initialization, the main loop, and follow-up work at the end. It is depicted in Figure 5.4. A dendrogram generated from an example clustering can be seen in Figure 5.5. Under the assumption that every initial motif is a one-element cluster, every run of the main loop reduces the number of clusters by one.

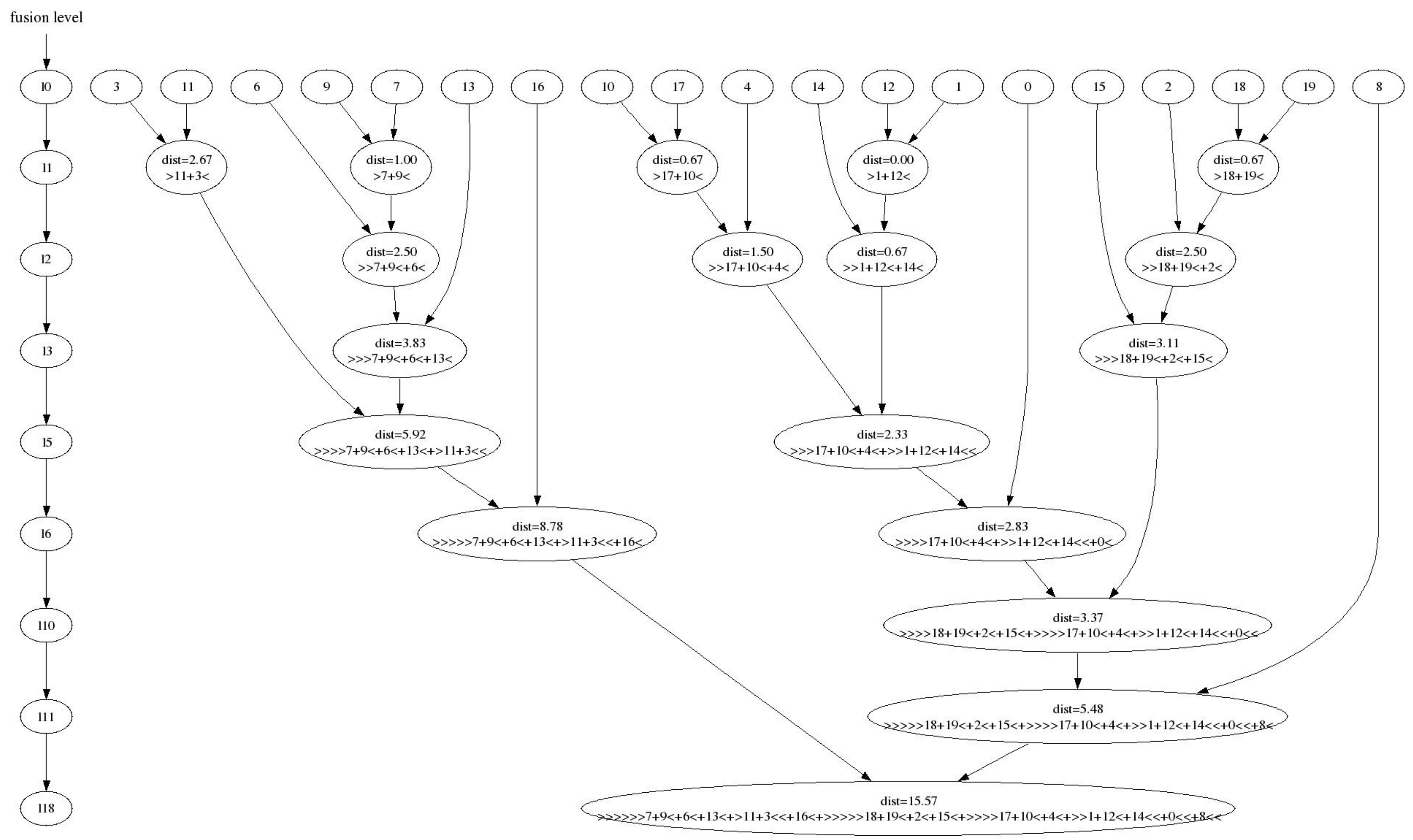

fusion level
l0
l1
l2
l3
l5
l6
l10
l11
l18
3
11
6
9
7
13
16
10
17
4
14
12
1
0
15
2
18
19
8
dist=2.67 >11+3<
dist=1.00 >7+9<
dist=0.67 >17+10<
dist=0.00 >1+12<
dist=0.67 >18+19<
dist=2.50 >>7+9<+6<
dist=1.50 >>17+10<+4<
dist=0.67 >>1+12<+14<
dist=2.50 >>18+19<+2<
dist=3.83 >>>7+9<+6<+13<
dist=3.11 >>>18+19<+2<+15<
dist=5.92 >>>>7+9<+6<+13<+>11+3<<
dist=2.33 >>>17+10<+4<+>>1+12<+14<<
dist=8.78 >>>>>7+9<+6<+13<+>11+3<<+16<
dist=2.83 >>>>17+10<+4<+>>1+12<+14<<+0<
dist=3.37 >>>>18+19<+2<+15<+>>>>17+10<+4<+>>1+12<+14<<+0<<
dist=5.48 >>>>>18+19<+2<+15<+>>>>17+10<+4<+>>1+12<+14<<+0<<+8<
dist=15.57 >>>>>>7+9<+6<+13<+>11+3<<+16<+>>>>>18+19<+2<+15<+>>>>17+10<+4<+>>1+12<+14<<+0<<+8<<

Figure 5.5: Example for an agglomerative, hierarchical clustering of 20 motifs numbered from zero to 19. Every leaf represents a motif, every inner node a cluster. The root node (at the bottom) is the one resulting super cluster. The fusion level gives the number of motifs incorporated into the cluster minus one. Clusters with the same fusion level reside on the same height. The term “dist” gives the Manhattan distance between the two cluster representatives the cluster was formed off. The picture was drawn with the *dot* program of the *graphviz* package (open source, www.graphviz.org).

The motifs to cluster are given via option file. Any number of motifs can be clustered. Note that basic motifs and motif patterns are not mixed in the clustering. This also applies to motif patterns with a different number of comprised basic motifs. Also note that the length of basic motifs does not handicap the clustering. Therefore, if the clustering itself is not restricted to end earlier, it is possible to cluster a sequence of length 100 to a short motif of length 5. Of course, in such a case, the distance would be very high, the likelihood would be low, respectively, which would cause these two motifs to be clustered together very late in the process.

In the clustering process, every cluster has a representative, a consensus motif. At the end, it is printed for every remaining cluster. The type of the final representative motif depends on the original motif. Motif patterns to cluster have motif patterns of the same complexity as representative. Basic motifs are output as regular expressions, PSPMs or PSSMs. Regular expression motifs are only possible, if the initial motif was of this type, and if the motif remains unclustered to the very end. In every other case, regular expression motifs become PSPMs. Note that in the current version of the *jPREdictor* PSSMs are not recalculated to PSPMs or vice versa, and that it is therefore possible, to mix both types of matrices, even if it is not recommended. The reason for the missing recalculation of PSPMs to PSSMs or vice versa is that the calculation method the *jPREdictor* uses might be insufficient to the user. In addition, the *jPREdictor* recalculation method uses knowledge about the background nucleotide distribution, which is normally unknown to the clustering process. And since the user has no means to make his method known to the *jPREdictor*, neither matrix is recalculated. If a PSPM is output as representative of a cluster, its threshold is set to the maximum possible threshold divided by 10. In case of a PSSM, the threshold is set to the maximum score minus two. Both thresholds are very restrictive and are more or less equal to search a sequence motif with one mismatch allowed.

5.3.6 End of clustering

If the clustering is unrestricted, it yields one super cluster comprising all available motifs into one representative. Note that certain kinds of motifs are not mixed together. Therefore, there may exist more than one super cluster at the end. But let’s assume, all motifs have the same kind of complexity. Let N be the number of motifs, the level of this super cluster would be $N - 1$ (also called fusion level, see Figure 5.5).

The question now is, at which clustering level below N the clustering has to stop, on the one hand, to yield a reasonable amount of motifs, and on the other hand, such that all remaining consensus motifs are most different to each other. It is important to note that no limitation is set to the cluster size. Thus, a situation is imaginable, where the clustering stops at level $N - 3$, and yields 2 non-clustered motifs and one super cluster comprising all residual motifs.

If the number of clusters to build is known beforehand, this number is simply provided to the program via command-line parameter ’-C’. If this number is not known, an early stop must be reached by other means.

The rule of thumb is to draw the distance measure against the clustering level and use the “elbow”-criterion: at some point in clustering the distance will leave its near linear increase to rise more strongly. But note that the actual “elbow” might not be identified unambiguously. For this, the first step is a complete clustering with switched on verbose messages (command-line option ’-v’). This will print out, among much other text, the actual clustering level and the corresponding best distance measure. Another very simple rule is to run the clustering until the amount of motifs is reduced to

10% of the original set. But note that this step has to be preceded by an evaluation by-hand on which motifs to keep and which to remove. In case of initial motifs that are obtained by a motif evolution, the motifs yielded by the clustering should be weighted once more and the 10 to 20 highest-weighting ones should be kept. Another way to automate the stopping rule is given now.

The information criterion used to describe the stopping rule (after Akaike, therefore abbreviated AIC, [115]) defines a relationship between the number of clusters present and the likelihood how well the original objects are represented in their clusters. While the clustering level increases by one per step the number of clusters actually is decreased by one, because two clusters are combined in every clustering step. Now, for every cluster a representative, the consensus motif, is always present. This consensus motif "explains" in a way all motifs comprised in this cluster. The degree of explanation is expressed as a likelihood measure: the probability of motif X given motif Y, $p(X|Y)$. When clusters are merged, the newly generated consensus motif most often explains the comprised motifs worse, thus the within-cluster likelihood decreases. For defining the stopping rule at every clustering level the AIC is calculated as:

$$AIC = 2 \cdot k - 2 \cdot \frac{\log L}{k}, \tag{5.16}$$

with k the actual number of clusters, and L the product of likelihoods over all clusters present at the moment: $L = \prod_{i=1}^{k} L_{C_i}$. The likelihood for one cluster $C = C_i$ is defined as the product of likelihoods of the consensus to all motifs comprised within:

$$L_C = \prod_{c \in C} L_{shift}(m_{C_{consensus}}, m_c). \tag{5.17}$$

Since k is to be minimized and $-\log L$ is to be minimized, too, the clustering process stops when the minimum AIC is reached. Dividing $\log L$ by k averages the log-likelihood over the number of clusters. Another possibility would be to average over the number of original objects, by dividing by n, thus stating how well the original motifs are explained by their respective consensus motif. In Figure 5.6, the two formulas are visualized. Using n is closer to the original definition of the AIC, but it also causes the clustering to end very late, which is not preferred, since the final motifs become too degenerated.

Note that the AIC is an incorporation of the "elbow" criterion. The AIC becomes smaller with every clustering level, if $-\frac{\log L}{k}$ increases more slowly than k can fall. In every step k decreases by one, therefore, if $d = \frac{\log L_k}{k} - \frac{\log L_{k-1}}{k-1} < 1$, the AIC will decrease further. If d becomes greater one, the AIC will rise. The *jPREdictor* defines the cut-off for the stopping rule in terms of d. Experience shows that a $d = 0.1$ is often sufficient, when clustering few or very similar motifs.

If the clustering is invoked using the Manhattan or the Euclidean distance measure, the cut-off is given in terms of this distance. The clustering ends, if no two motif pairs exist that have a distance smaller than the cut-off.

In general, the clustering ends if the specified number of clusters is reached, or if the criterion is fulfilled, whichever comes first. In all further evaluations presented in this work, the clustering performed uses the likelihood measure in combination with the "Forward" approach. They belong together either way, and the two main reasons for using them over distance measure and Greedy approach is that the "Forward" approach checks the cluster beforehand, whether the new motif fits into the cluster, and that the automated stopping rule according to the information criterion can be applied.

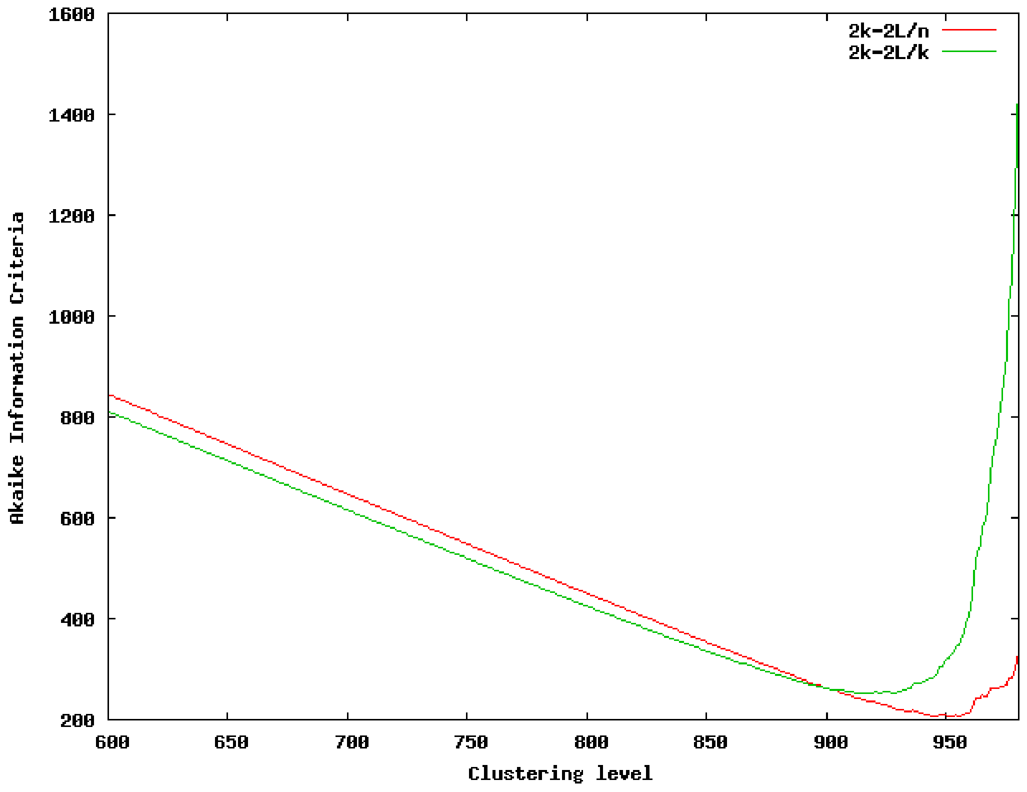


Figure 5.6: The Akaike information criterion dependent on the clustering level. The initial number of motifs was $n = 1,000$. Thus, k is calculated as n minus clustering level. The motifs originate from a motif discovery using several evolutionary runs.

5.4 Single motif discovery: continuing the Tompa assessment

While the *jPREdictor* motif evolution and clustering was foremost developed to discover motif patterns in model versus background sequences, it can be applied to single motifs as well. In this chapter, the discovery of single motifs is presented in order to show the performance of the *jPREdictor* in relation to other motif discovery tools. For this purpose, Tompa et al. [102] created a framework, which is applied here.

5.4.1 Study design and assessed tools

Tompa et al. [102] assessed 13 motif discovery tools for the correctness of their discovery of binding sites. These sites were taken from TRANSFAC and were planted into sequences. Tompa et al. specifically did not work with motifs but with binding sites, and they specifically did not plant sites generated from motifs into sequences. Rather they planted confirmed binding sites from TRANSFAC into the sequences. This is a difference, because sites generated from motifs, i.e. sites matched by motifs due to the degeneration level of the motif, need not necessarily be true binding sites. Eventually, Tompa et al. [102] used binding sites from 26 human transcription factors, and from 6 *Drosophila* factors. The corresponding binding sites for the factors were planted into three different sequence types, first, the binding sites' real promoter sequences ('real'), second, randomly chosen promoter sequences from the same genome ('generic'), and third, sequences generated under a third order Markov model ('markov'). All in all, this procedure leads to 76 human and 18 fly data sets.

The following tools were assessed: AlignACE [87], ANN-Spec [97], GLAM [116], MotifSampler [117, 88], and SeSiMCMC [118] are based on Gibbs sampling, Improbizer [119], and MEME [85] are based on expectation maximization, Consensus [86] tries to maximize the information content of a PSPM, MITRA [120], Oligo/dyad-analysis [121], QuickScore [122], Weeder [123, 124], and YMF [84] are word-based. In the assessment, the creators of the respective motif discovery tools were permitted to vary parameter settings from data set to data set, mask repeats in the input sequences, post-process the output to eliminate low-complexity motifs and generally perform any pre- and post-processing deemed appropriate. Neither consultation of TRANSFAC nor the employment of methods that would not be available in a real application of novel motif discovery were permitted. In order to learn as much as possible from running the *jPREdictor* motif evolution any pre- and post-processing was excluded. Additionally, no parameter adaptation was performed in order to input any prior information about length and nucleotide composition of the motif.

Tompa et al. [102] designed a website (http://bio.cs.washington.edu/assessment/) that allows to compare the motif discovery results of other programs to the tools used in the assessment. On the website, the sequences (called benchmark data sets) can be downloaded and own results can be uploaded. Uploading means reporting predicted sites for each data set. Note that Tompa et al. created a few data sets without any planted site. The creators of the tools could decide on how many sites they report. After the upload, the website will output a table that contains the evaluation results of the comparison of predicted versus planted binding sites for each data set. In addition, an applet can be started that gives the results in a graphical form. Some pictures presented here are screenshots from that applet.

5.4.2 Running the *jPREdictor* motif discovery on the human data set

From motif discovery to binding sites: a framework

The *jPREdictor* motif evolution was first applied to all 76 human data sets. This was done, because the human data set is much greater than the fly one, and thus gives a general overview on how the *jPREdictor* performs. A framework was created around the motif discovery approach. The first step in the framework was the evolution. For each of the 76 human data sets, the motif evolution “RewardedSelection” was run 10 times. The command-line call for every run was:

```
-m model -b background --motifEvolutionRewardedSelection
-d NNNNNNNN -d NNNNNNN -v -p single -c 1000
```

In the assessment, the motif discovery tools were not informed about the sequence type. For comparability, the same condition is assumed here and the evolutions were run with only one background sequence set. This negative training set (background) consisted of 10 randomly chosen promoters from the human genome, each of length 10 kb.

The evolution was repeated 10 times and yielded up to 200 unique motifs for every data set. Afterwards, the motifs were clustered using the “Likelihood” measure and the “Forward” algorithm. The cut-off was set to 0.1. On average, this reduced the number of motifs to 14 for each data set. The motifs resulting from the clustering were reweighted using model and background sequences, and for each set, the highest-weighting motif was chosen.

Each highest-weighting motif was then matched to the sequences of its corresponding data set (model) and the obtained positions together with the extracted binding sites were reported to the webpage (http://bio.cs.washington.edu/assessment/).

Results of the motif discovery in the human data set

For each human data set a highest-weighting motif existed. No motif was rejected. Thus, for each data set binding sites were reported to the website. In fact, rejecting motifs is very difficult without ample knowledge about real biological binding sites. A possible approach in order to support the rejection decision could be to mark low-weighting motifs. Nevertheless, with this approach, the rejection task is shifted to answering the question what low means in the context of weights. The problem is that the maximal weight for a motif is unknown within one data set as well as from one data set to the next. In addition, even discovering motifs in randomly generated sequences yields motifs with high weights. These two problems are the reason for refraining from a rejection scheme. Along with the *jPREdictor*, Improbizer, SeSiMCMC, MITRA, and MotifSampler also predicted binding sites for each human data set [102].

For the *jPREdictor*, the mean positive predictive value for all 78 human data sets was 6.8% on both nucleotide and site level (Table 5.2). The sensitivity was 2.9% on nucleotide level and 5.3% on binding site level. This means that only very few binding sites were predicted correctly. For the other tools Tompa assessed, the results are listed in Table 5.2, too. These results were obtained from the website. Consensus is missing in the list, because it was not applied to the human dataset. The best tools in terms of the performance coefficient are ANN-Spec and Weeder. They found the most

Table 5.2: Assessment results for the human data sets. Abbreviations used for the statistical parameters: PC performance coefficient, PPV positive predictive value. The abbreviations for the motif discovery tools are: E expectation maximization based, G Gibbs sampling based, W word-based. On the human data set, the *jPREdictor* motif discovery shows a mediocre performance in comparison to the other tools.

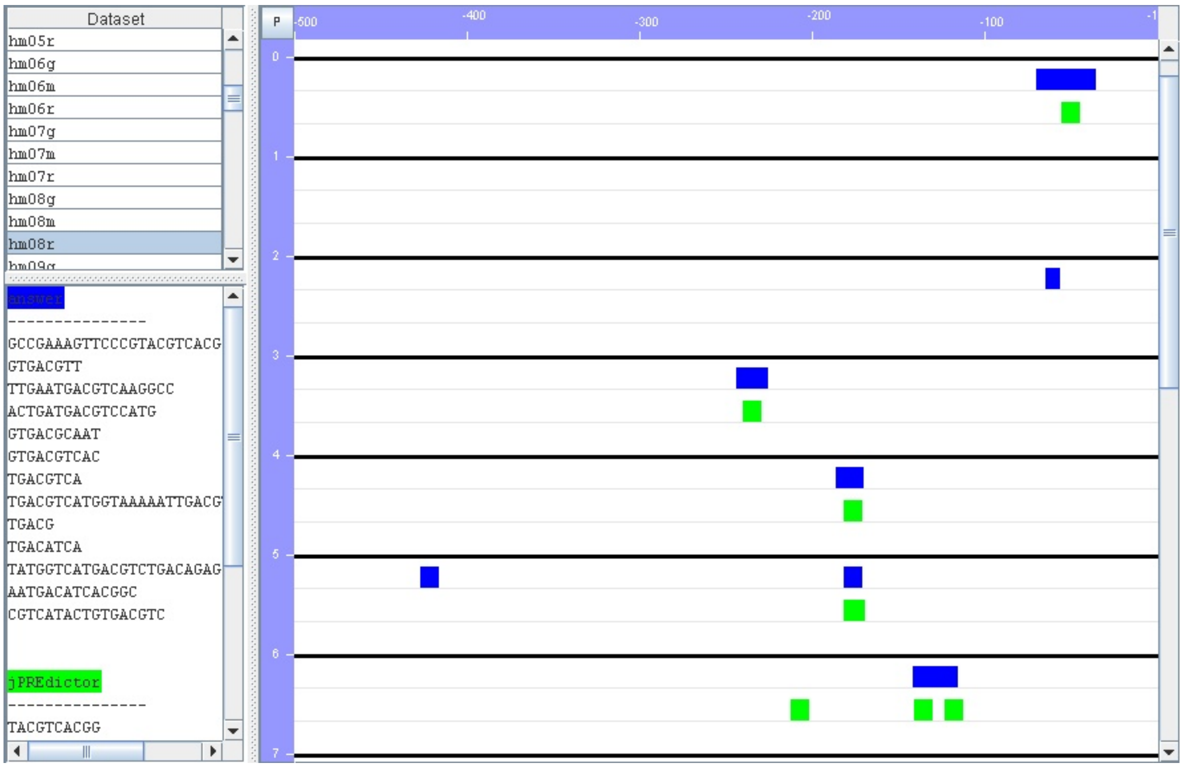
			Nucleotide level			Site level	
		PC	Sensitivity	PPV	Specificity	Sensitivity	PPV
AlignACE	G	2.92	3.93	10.26	99.38	7.38	12.36
ANN-Spec	G	5.06	9.03	10.32	98.59	16.44	9.84
GLAM	G	1.46	2.36	3.68	98.88	4.03	6.00
Improbizer	E	2.27	4.16	4.76	98.50	7.05	4.84
MEME	E	2.39	3.81	6.04	98.93	6.04	8.11
MEME3	E	2.27	4.20	4.71	98.47	6.38	7.88
MITRA	W	1.63	2.44	4.71	99.11	4.03	4.69
MotifSampler	G	1.59	2.50	4.17	98.96	4.70	4.31
oligodyad	W	3.27	3.71	21.40	99.75	6.04	15.00
QuickScore	W	0.34	0.51	0.99	99.09	0.00	0.00
SeSiMCMC	G	1.77	4.59	2.80	97.13	6.71	6.31
Weeder	W	4.75	5.43	27.47	99.74	10.74	25.81
YMF	W	2.97	4.10	9.67	99.31	7.38	8.03
jPREdictor	W	2.06	2.87	6.83	99.29	5.26	6.79

true positives while keeping the number of false negatives and false positives low. AlignACE and ANN-Spec have almost the same PPV (nucleotide level), but differ in their sensitivity. Therefore, AlignACE missed a lot of planted binding sites, even though the ratio of true positives among all positives is high. This means that ANN-Spec predicted many more binding sites than AlignACE. In fact, AlignACE reported no binding sites for 17 out of 26 human binding factors (for each of the three sequence types, thus 51 out of 76 data sets), whereas ANN-Spec for only one [102]. The best tools in terms of true positive ratio (PPV) seem to be Weeder and oligodyad, as their predicted binding sites cover planted sites with a probability higher than 20% (nucleotide level). This is expressed as a high performance coefficient. Both tools are word-based and (almost) exhaustively enumerate sequences.

The *jPREdictor* motif discovery performs with a low sensitivity, and a moderate PPV compared to the other tools. In this analysis, the word-based tools are the best in terms of PPV, their accuracy is the highest. Nevertheless, in terms of sensitivity, all tools perform comparably low, with ANN-Spec being the most sensitive, because its number of predicted sites was very high. The specificity was higher than 95% for every data set, mostly around 99%. This means that the predicted number of binding sites always was low enough to not cover large areas of the sequences.

Two examples

Applying the *jPREdictor* motif discovery to the human data sets produced a mix of very accurately predicted binding sites for some sets and, for other sets, an utter lack of conformance between predicted and planted sites. A very good prediction result is shown in Figure 5.7. In the example, the PPV was 73.5% for nucleotide positions, and 80% for site positions. The latter means that four out of five

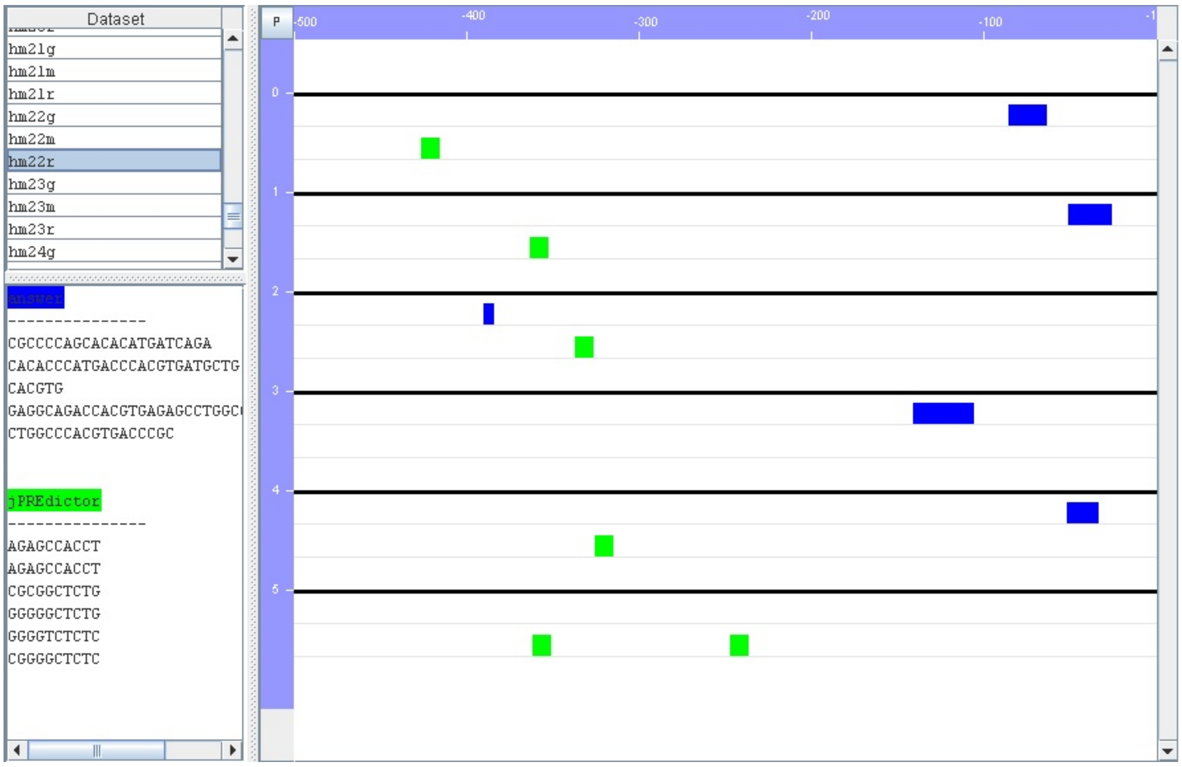


```
Dataset
hm05r
hm06g
hm06m
hm06r
hm07g
hm07m
hm07r
hm08g
hm08m
hm08r
hm09g

answer
---------------
GCCGAAAGTTCCCGTACGTCACG
GTGACGTT
TTGAATGACGTCAAGGCC
ACTGATGACGTCCATG
GTGACGCAAT
GTGACGTCAC
TGACGTCA
TGACGTCATGGTAAAAATTGACG
TGACG
TGACATCA
TATGGTCATGACGTCTGACAGAG
AATGACATCACGGC
CGTCATACTGTGACGTC


jPREdictor
---------------
TACGTCACGG
```

Figure 5.7: The prediction of binding sites in the data set hm08r, consisting of 15 real human promoter sequences, each of length 500. The first seven sequences are shown, depicted as black horizontal lines. The blue bars are the planted transcription factor binding sites, the green bars are the predicted binding sites using the *jPREdictor*'s motif discovery. For this data set, the positive predictive value on nucleotide level was over 73%.



```
hm21g
hm21m
hm21r
hm22g
hm22m
hm22r
hm23g
hm23m
hm23r
hm24g

answer
---------------
CGCCCCAGCACACATGATCAGA
CACACCCATGACCCACGTGATGCTG
CACGTG
GAGGCAGACCACGTGAGAGCCTGGC
CTGGCCCACGTGACCCGC


jPREdictor
---------------
AGAGCCACCT
AGAGCCACCT
CGCGGCTCTG
GGGGGCTCTG
GGGGTCTCTC
CGGGGCTCTC
```

Figure 5.8: The prediction of binding sites in the data set hm22r, consisting of 6 real human promoter sequences, each of length 500. All six sequences are shown, depicted as black horizontal lines. The blue bars are the planted transcription factor binding sites, the green bars are the predicted binding sites using the *jPREdictor*'s motif discovery. For this data set, the positive predictive value on both nucleotide and site level was zero percent.

predicted binding sites correctly shared two third of their nucleotides with the planted binding site, on nucleotide level, this PPV drops a little due to fewer overlap. The sensitivity in the example was 39% for nucleotide coverage, and 61% for site coverage. This means that three out of five planted binding sites were correctly covered by the predicted ones and that almost 40% of the known nucleotides from the planted sites had their counterpart in the predicted sites.

An example for a very bad prediction is shown in Figure 5.8. The positive predictive value as well as the sensitivity was zero on both nucleotide and site level. This means that none of the predicted binding sites covered the planted sites, not even partially. Actually, having a zero PPV and a zero sensitivity for a data set happened very often. On nucleotide level, this was the case for 54 out of 76 human data sets. On site level, no overlap existed in 59 cases (Table A.1 in the appendix). Performing with zero PPV and sensitivity was not limited to the *jPREdictor* motif discovery, but all assessed tools performed this way for the majority of the data sets ([102], supplementary material).

However, performing with zero sensitivity does not mean that no real transcription factor binding sites were predicted. It simply means that the planted binding sites were missed. Some of the sequences the binding sites were planted into are real promoter sequences. As Tompa et al. [102] already pointed out, these sequences may contain other true transcription factor binding sites that were predicted by the motif discovery tools instead of the planted ones.

Discussion

Tompa et al. [102] gave many reasons, why all motif discovery tools performed so badly. One of these reasons is that the planted binding sites for one transcription factor differ in length. The sites are cataloged in TRANSFAC and reflect the resolution of experimental approaches. The true binding site may actually be a shorter subsequence. Tompa et al. [102] used sites up to a length of 71, and 35 binding sites planted were of length longer than 31 nucleotides. In Figure 5.7 the smallest binding site has a length of 5, but the largest was of size 34. Like the other tools, the *jPREdictor* has no means to evolve a motif based on binding sites of different lengths. It is able to evolve motifs of length in-between 5 to 10, but the each motif itself is of fixed size.

Tompa et al. [102] gave another reason for the low performance of the tools. Each tool was allowed to report the sites for only one motif. Tompa et al. [102] argue that the choice of this one motif is subjective and error-prone. However, in the framework, the highest-weighting motif was chosen as the one motif, and this is an objective decision. Nevertheless, it remains error-prone. In practice, a reasonable approach would be to pursue the top several motifs discovered by any given tool. Allowing for more than one motif and therefore allowing for more reported sites would have a drastic positive effect on the sensitivity. This effect is shown in the next chapter, when the *jPREdictor* motif discovery is applied to the fly data set. The assessment designed by Tompa et al. [102] is based on binding sites and not on motifs and therefore supports the report of sites for multiple motifs.

5.4.3 Running the *jPREdictor* motif discovery on the fly data set

Altering the framework

In the next step, the framework was applied to the fly data sets. Different configurations of the framework give insights into how the discovery can be improved. Two different sequence sets were used as background (negative training set), first, the heatshock and cell cycle promoter sequences used in the prediction, and second, 10 sequences of length 10 kb randomly generated under a second order Markov model. The *jPREdictor* motif discovery was applied several times to both the fly data set versus promoters and the fly data set versus MC2 sequences.

For the promoter training set as background model, two changes in the framework are made. Both are introduced to see their impact on the results and maybe to improve accurateness. First, after clustering, the 5 highest-weighting motifs are checked for how often they occur in the model sequences. The motif with the most occurrences was taken, regardless of its occurrence number and how balanced the occurrences were, and also regardless on how often it occurred in the background sequences. This step was made because Tompa et al. [102] planted more than one motif in such small sets and this approach is aiming at maximizing the motifs occurrences in each fly set. The second change is that the orientation of the motifs is either unrestricted (as above) or restricted to the plus strand. The restriction could easily be made by altering the template motifs. This means changing both “-d” parameters in the command-line call:

```
-d NNNNNNNN,temp8,0,plus -d NNNNNNN,temp7,0,plus
```

Contrary to the human data sets, the fly data sets are more challenging, because they are very limited in length and in the number of sequences. The smallest set consisted of only one sequence of length $2,000$. All in all, 6 sets out of 22 contained only one sequence, namely set 2 and 6 for all three sequence types. Both the limited number of sequences per set and the shortness of the sequences led to a third alteration in the framework. For every transcription factor, all three data sets were combined into one set.

Results of the motif discovery in the fly data set

The results of applying the framework to the fly data sets can be seen in Table 5.3. In general, many of the motif discovery tools reported zero correct binding sites. Among the others, the best was SeSiMCMC with a sensitivity of about 10% on both site and nucleotide level. This value was reached, because SeSiMCMC reported overlapping binding sites for three out of six binding factor sets, and because the sensitivity was very high for the first set. The same applies to both MEME tools, they reported overlapping sites for two out of six sets, and the sensitivity was very high for the first set. For all other tools, it is more or less random, whether an overlapping site could be reported or not. As was said, the fly data sets are very challenging, since they are extremely small.

Running the motif discovery framework with the promoter background set yielded few correctly identified binding sites. Thus, sensitivity and PPV are below 1% on nucleotide level (Table 5.3). The higher values on site level in the "Promoter, plus" run result from one correctly predicted binding site for exactly one data set, dm01r. In this set, 7 sites were planted, the *jPREdictor* reported only 3, one was correct. This gives a sensitivity 14% and a PPV of 33% on site level. Again, this tells how small these sets are and that correct predictions are more or less random. In this sense, restricting the search for binding sites to the plus strand has no impact, as well as choosing the most occurring motif among the five best-weighting ones. Consequently, in order to improve the prediction results, other steps have to be taken.

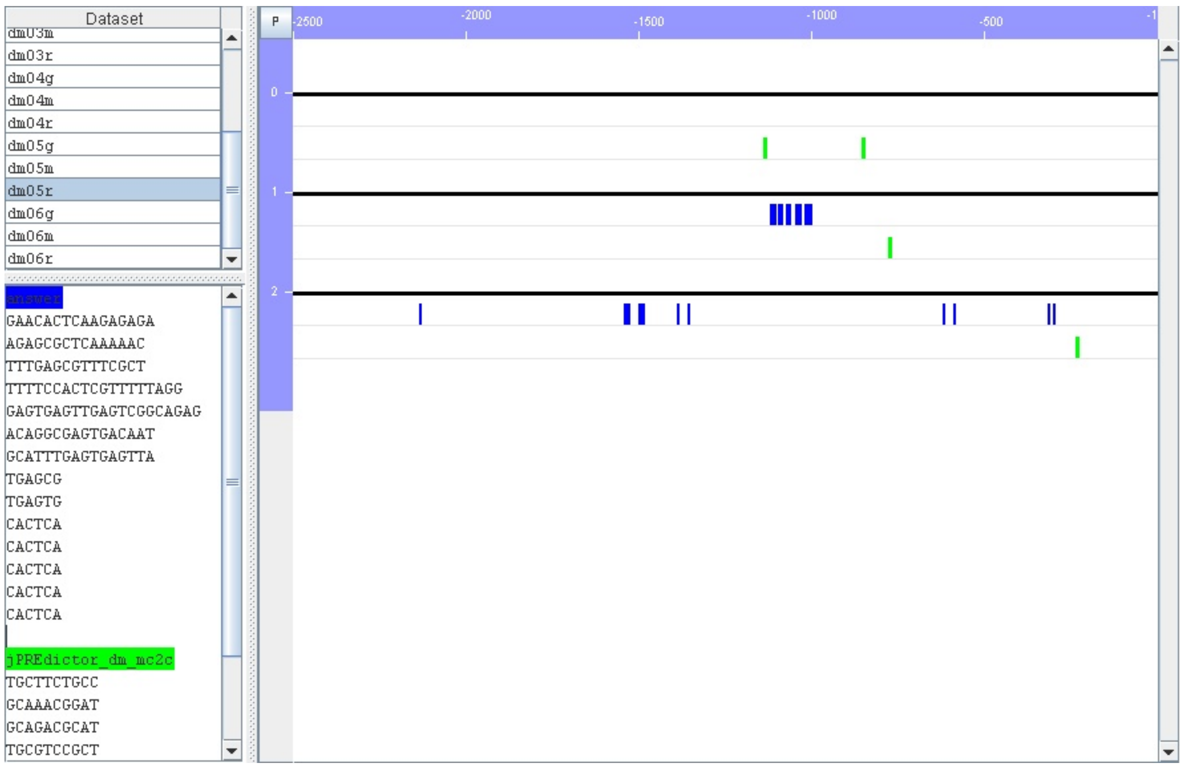
In four runs, the motif discovery was run with randomly generated sequences as a background model. Applying the second order Markov model (MC2) resulted in a significant increase of sensitivity and PPV on both site and nucleotide level in comparison to the runs using the promoter background (Table 5.3). While the latter had a non-zero sensitivity and PPV for only two sets, under the MC2 model it was increased to 4 sets. This shows that the choice of a good background is very important. Either the promoter sequences are not random enough, or they are too biased. The latter might be more probable, since all sequences are promoters from either heatshock or cell cycle genes.

In Table 5.3, both "all" strategies refer to reporting all possible binding sites for all clustered motifs. Normally, one evolution run yields 20 motifs. Repeating this 10 times results in up to 200 unique motifs. Applying the clustering reduced this number to 4 to 21, in the mean around 10. For all these motifs, the corresponding binding sites were reported to the webpage. The results can be seen in Table 5.3 in the row "MC2, both, all". In comparison to row "MC2, both", sensitivity on both nucleotide and site level rises to 15% and 21%, respectively. At the same time, specificity drops significantly to 93%. This shows that many overlapping sites are reported, but at the same time many false positives, too. In fact, very often the correct motif is among the ones obtained after evolution and clustering, even if it is not the highest-weighting one. This is a problem with all motif discovery tools. Since they rely on statistical over-representation, they will report many false positive motifs and the correct one is hidden among them.

For the two last discovery runs, all three sequence sets for one factor, i.e. real, generic, and markov,

Table 5.3: Assessment results for the fly data sets. Abbreviations used for the statistical parameters: PC performance coefficient, PPV positive predictive value. The abbreviations for the motif discovery tools are: E expectation maximization based, G Gibbs sampling based, I information content based, W word-based. The *jPREdictor* motif evolution and clustering was started in 6 different configurations. Either promoters from heatshock and cell cycle genes (=”Promoters”) or sequences generated under a second order Markov model (=“MC2”) served as background set. The other mnemonics mean (refer to the text for further explanations): plus, motif discovery was restricted to the plus strand; both, plus and minus strand allowed; all, binding sites from all resulting motifs (not only from the one best) are reported; combined, the 3 data sets for one binding factor are combined to one bigger set.

			Nucleotide level			Site level	
		PC	Sensitivity	PPV	Specificity	Sensitivity	PPV
AlignACE	G	0	0	0	99.74	0	0
ANN-Spec	G	1.05	2.53	1.75	97.75	1.96	0.94
Consensus	I	0	0	0	99.22	0	0
GLAM	G	0.19	0.30	0.49	99.04	0	0
Improbizer	E	0.81	1.49	1.76	98.68	1.96	2.27
MEME	E	2.14	4.17	4.21	98.50	5.88	5.56
MEME3	E	1.57	3.73	2.64	97.82	5.88	4.48
MITRA	W	0	0	0	99.61	0	0
MotifSampler	G	0.29	0.45	0.82	99.14	0	0
oligodyad	W	0	0	0	98.58	0	0
QuickScore	W	0	0	0	98.44	0	0
SeSiMCMC	G	3.65	10.13	5.39	97.18	9.80	12.50
Weeder	W	0.89	1.19	3.45	99.47	1.96	3.45
YMF	W	0	0	0	98.79	0	0
jPREdictor	W						
Promoter, plus		0.74	0.99	2.82	99.41	1.96	4.23
-, both		0.23	0.35	0.68	99.12	0.65	0.89
MC2, both		1.22	1.94	3.21	98.99	2.61	2.82
-, -, all		2.89	15.25	3.44	92.60	20.92	2.57
-, -, combined		8.29	10.28	30.00	99.49	13.73	23.60
-, -, -, all		12.46	32.54	16.80	96.60	37.91	10.32



dm03m
dm03r
dm04g
dm04m
dm04r
dm05g
dm05m
dm05r
dm06g
dm06m
dm06r

answer
GAACACTCAAGAGAGA
AGAGCGCTCAAAAAC
TTTGAGCGTTTCGCT
TTTTCCACTCGTTTTTAGG
GAGTGAGTTGAGTCGGCAGAG
ACAGGCGAGTGACAAT
GCATTTGAGTGAGTTA
TGAGCG
TGAGTG
CACTCA
CACTCA
CACTCA
CACTCA
CACTCA

jPREdictor_dm_mc2c
TGCTTCTGCC
GCAAACGGAT
GCAGACGCAT
TGCGTCCGCT

Figure 5.9: The prediction of binding sites in the fly data set dm05r, consisting of 3 real *Drosophila* promoter sequences, each of length 2,500. All three sequences are shown, depicted as black horizontal lines. The blue bars are the planted transcription factor binding sites of the factor Zeste, the green bars are the predicted binding sites using the *jPREdictor*'s motif discovery. For this data set, the positive predictive value on both nucleotide and site level was zero percent.

are combined to one model set (Table 5.3). For one of them, the binding sites from the best motif are reported, for the other run, all binding sites from all remaining motifs after clustering are reported to the webpage. The consequences are dramatic. In comparison to the non-combined run sensitivity is increased from around 2% to about 10% while maintaining a very high specificity. This shows that the reported binding sites covered many true binding sites. The PPV rises significantly by a factor of ten. Therefore, many true sites are among the reported ones. More clearly than before, this shows that the data sets in general are too small when uncombined. Combining the three data sets for one factor increases the over-representation level for the binding sites of this factor with respect to the background. This leads to a better separation of motifs over the noise.

Example motif: Zeste

The fly data sets contained binding sites of the motif Zeste. Nevertheless, the *jPREdictor* evolution and subsequent clustering was never able to find this motif. This was true for all assessed tools. Any overlap in reported binding sites was due to the degeneration level of other motifs. The data set containing Zeste binding sites can be seen in Figure 5.9. This data set consists of 3 sequences (all-in-all, 9 sequences). Tompa et al. [102] planted 5 binding sites in the second sequence and 9 in the third.

The Zeste motif is defined as $YGAGYG$. Searching this motif gives 3 occurrences in the first sequence, 7 in the second, and 13 in the third. This applies to the real data set. For the generic and the markov set, the occurrence counts are 3, 10, and 11, respectively, for both sets. This reveals one problem. Zeste is too unspecific. The weight as a measure for over-representation is very small for the Zeste motif, around 0.6 no matter which one of the two background models is applied, and also no matter whether all three sequence sets or only one is applied. Therefore, the Zeste motif was never discovered by the motif evolution and subsequent clustering.

Discussion

The fly data set as a subset of the Tompa assessment [102] depicts the most adverse conditions for running motif discovery tools. The data sets are small and only the binding sites for the best motif are allowed to be reported. Allowing for binding sites of more motifs to be reported drastically increases the sensitivity. This shows that the true motif often is among the top several ones.

As already addressed by Tompa et al. [102], the type of the sequence the binding sites are planted in had almost no effect on the prediction results. This was true for all tools used in the assessment, as well as for *jPREdictor*. In addition, no simple feature such as motif type or used algorithm determined the accuracy of the tools [102]. Therefore, despite considerable efforts in the field of motif discovery, it remains a complex challenge.

5.5 Double motif discovery

In this chapter the de-novo discovery of motif patterns is discussed. The *jPREdictor*'s motif discovery via evolution was specifically designed with the task in mind to be applicable to motifs of arbitrary complexity. Therefore, it is not only possible to discover single motifs as discussed in the last chapter, but the discovery of over-represented double, triple and higher-order motif patterns is also possible. The default setting, however, is the discovery of double motifs. In this chapter, the abililty of the *jPREdictor* to find planted motifs in randomly generated sequences will be discussed.

5.5.1 Study design

At first, the model sequences planted with actual binding sites were created. The task was to mimic the occurrences in confirmed PREs as much as possible. The study of confirmed PREs revealed an overall high number of the three motifs Zeste, GAF, and Pho. However, the motifs do not occur equally distributed over the whole sequence, but they occur in dense spots (clearly visible as peaks in the weight plot, top picture in Figure 5.10). Based on this observed arrangement, a method was developed for introducing binding sites into template sequences. At first, ten blocks of length 500 were created. Randomly, 10 up to 20 binding sites of Zeste, Pho, and GAF were planted into each block. In accordance to the procedure used by Tompa et al. [102], the single binding sites are taken from TRANSFAC, and are not generated from the motifs. Afterwards, one to two randomly chosen blocks are planted into each sequence of length $10,000$. The whole positive training set contains 10 of these sequences and therefore has a length of 100 kb. The planted blocks are visible in the weight plot (bottom picture in Figure 5.10), since they rise above the noise level. A second data set was created,

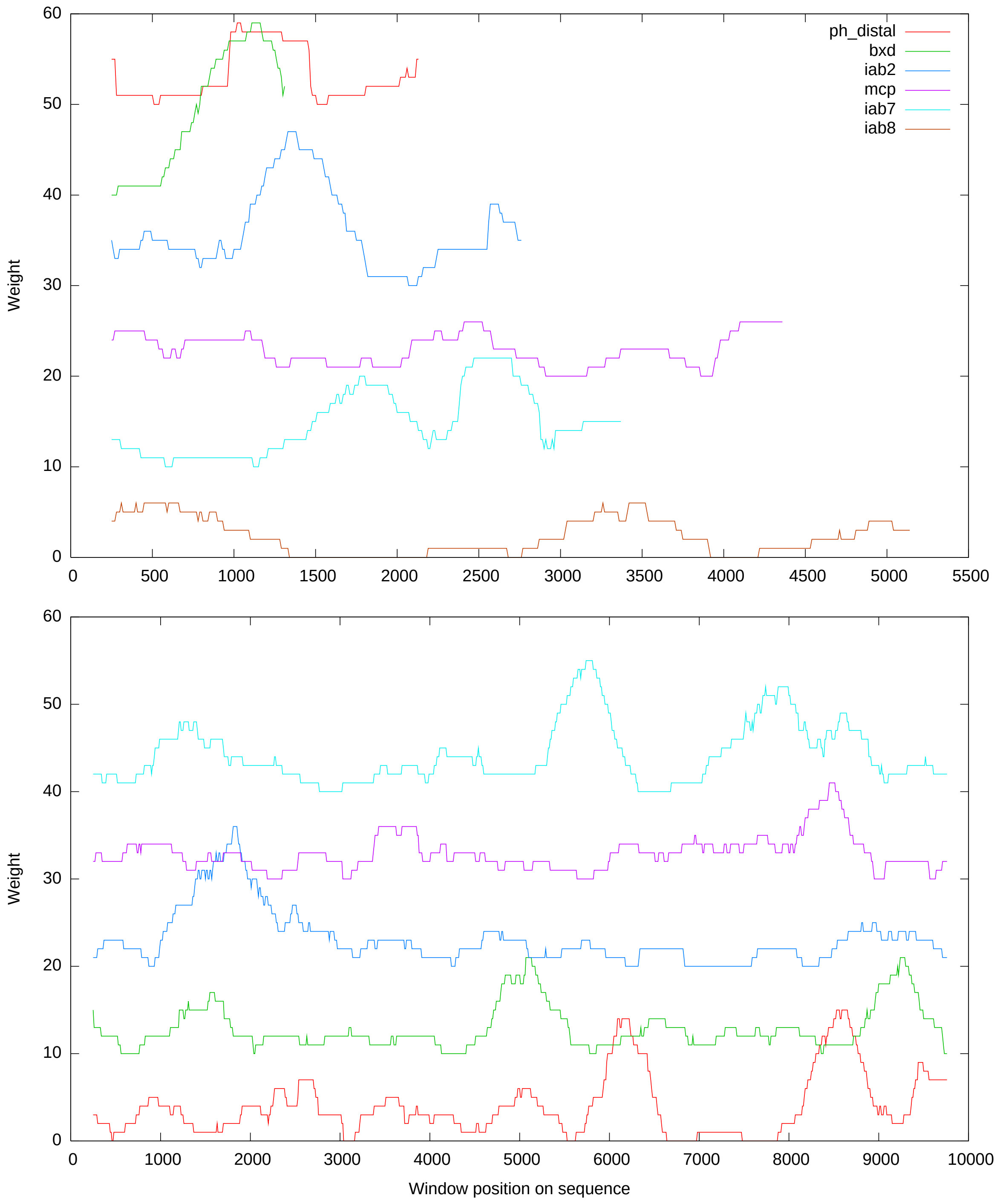


Figure 5.10: Weights of single motifs in confirmed PREs (top) and in random sequences planted with binding sites (bottom). The weights itself are counts of the three motifs, *YGAGYG* (Zeste binding motif), *GCCAT* (Pho core motif), and *GAGAG* (GAF/Psq binding site), per sequence window of width 500. In both plots, the baselines of the sequences are offset in multiples of 10. The planting of binding sites into random sequences was done in order to mimic the occurrences in confirmed PREs, which show not only enrichment but also densification.

for which the single blocks contained 20 to 30 binding sites. This gives even stronger signals. In contrast to the low-planted set, this data set is referred to as “high-planted”.

Note that the planting of binding sites into sequences is in accordance to the Tompa assessment [102]. Nevertheless, one criticism on the Tompa assessment [102] was that the signal-to-noise ratio was very low. In this analysis, the number of planted sites is much higher. Another criticism with the Tompa assessment [102] was that the tools were only allowed to report the sites for best motif. As was shown in the last chapter, the *jPREdictor* can perform much better (in terms of sensitivity), if the binding sites for more than one motif are allowed to be reported. Therefore, in this analysis, the tools are allowed to report the binding sites for several motifs. To counter this, they have to find not only one but three motifs.

All sequences, including the blocks of length 500, are generated under a second order Markov model obtained from *Drosophila melanogaster*. The same applies to the sequences of the negative training set. This latter set also contains 10 sequences of length 10 kb each.

The task of mimicking the appearance of the confirmed PREs in the weight plot (Figure 5.10) was not only fulfilled for single motif occurrences but also for co-occurrences of motifs (Figure 5.11). Counting co-occurrences (expressed as double motifs) amplifies the signal in comparison to counting mere single motifs and gives strong peaks high above the noise level (Figure 5.11). In this light, the discovery of motifs appears much more promising by preferring double motifs over single motifs.

Beside the *jPREdictor* several other motif discovery programs were chosen and ran on the generated data set. First, MEME [85] was used because of its status as famous motif discovery program and because it is a representative of the expectation maximization approaches. MotifSampler [117, 88] as well as BioProspector [89] are representatives of the Gibbs sampling approach. BioProspector was chosen in addition to MotifSampler, because it can be run in two-block mode to identify motif co-occurrences. The fourth motif discovery tool applied to the data set was Weeder [123, 124]. It was chosen, first, as a representative of the word-based approaches and, second, because it was one of the best-performing programs in the Tompa assessment [102].

The results from the motif discovery tools were evaluated in accordance to the Tompa assessment [102]. For each of the two data sets, the number of planted binding sites, their positions, and the coverage over the sequences is known. Applying each of the motif discovery tools to a certain data set yields a second set of predicted binding sites that has to be crosschecked with the planted ones. At nucleotide level, *nTP* gives the number of nucleotides in both planted and predicted sites, *nFN* gives the number of nucleotides in planted but not in the predicted sites, *nFP* is the number of nucleotides in the predicted but not in the planted sites, and *nTN* gives the number of nucleotides in neither planted nor predicted sites. From these values several statistical parameters can be calculated, sensitivity, specificity, PPV, and the performance coefficient (see Chapter 2.5).

On site level, *sTP* gives the number of planted sites overlapped with at least one nucleotide by a predicted sites. Contrary, *sFN* gives the number of planted sites not overlapped by anyone of the predicted sites, and *sFP* specifies the number of predicted sites not overlapping any planted site. On site level, the true negatives, i.e. the number of sites neither planted nor predicted, cannot be given, because such sites are unknown and therefore neither their position nor their length is known. As a consequence, only sensitivity and the PPV can be calculated on site level.

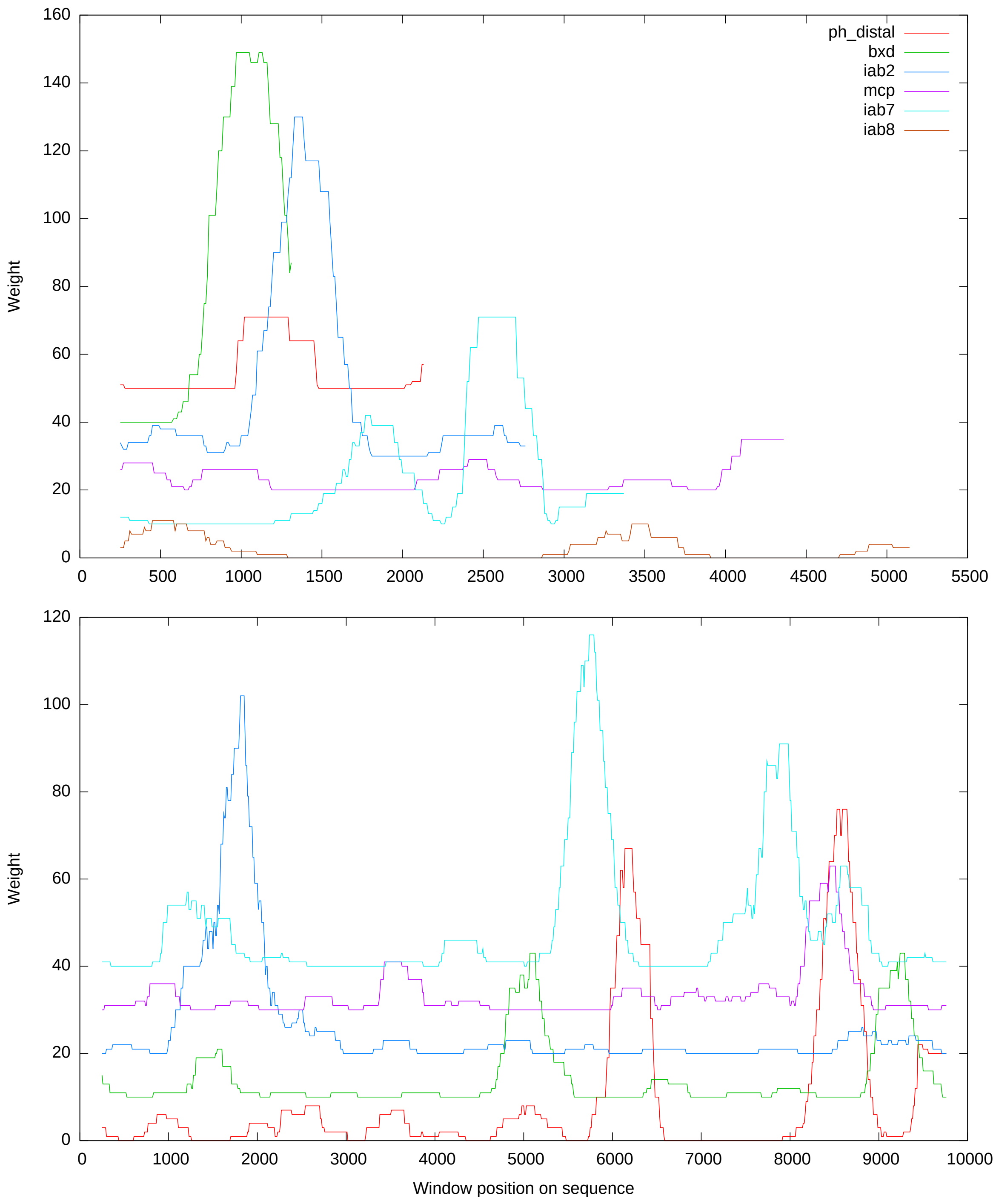


Figure 5.11: Weights of double motifs in confirmed PREs (top) and in random sequences planted with binding sites (bottom). The weights itself are counts of six double motifs per sequence window of width 500. The double motifs are composed of *YGAGYG* (Zeste binding motif), *GCCAT* (Pho core motif), and *GAGAG* (GAF/Psq binding site) with a distance of $(0,219)$ nucleotides. In both plots, the baselines of the single sequences are offset in multiples of 10. The planting of binding sites into random sequences was done in order to mimic the occurrences in confirmed PREs, which show not only enrichment but also densification.

5.5.2 Running the tools and a first analysis

jPREdictor

The *jPREdictor* motif discovery tool was applied to both sequence sets, low- and high-planted, in two ways. First, the discovery of single motifs was applied, and second, the motif discovery using double motifs was tested. This resulted in four runs, and in each run, the *jPREdictor* was started 10 times. The resulting 200 motifs from each run were clustered using the likelihood measure together with the "Forward" approach. The *jPREdictor* clustering was instructed to stop if the threshold of $c = 0.1$ was reached or in case 5 clusters were already created, whichever comes first. In case the clustering yields more than 5 motifs, they were re-weighted and the 5 highest-weighting motifs were taken. In every case, the five best-weighting motifs were used to predict binding sites on the sequences of both data sets.

Applying the single motif discovery to the low-planted sequences yielded 12 motifs after clustering. In Figure 5.12, the five best-weighting motifs are pictured as sequence logos. These logos and all following ones were created using R and the "seqLogo" library. While Pho and Zeste can be recognized in these motifs, GAF cannot be found. Applying the single motif discovery to the high-planted sequences yields exactly 5 motifs. These motifs are depicted, again as sequence logos, in Figure 5.13. Contrary to the motifs from the low-planted sequences, GAF is very dominant here and occurs in 4 out of 5 motifs. The Zeste motif does not occur. Nevertheless, occurrences of T at certain motif positions can be interpreted as Zeste influence lost in the clustering. Pho does appear twice in one (almost) palindromic motif. It is questionable, however, whether such a specific motif can be effectively matched to many binding sites.

Running the discovery of double motifs yields motif patterns consisting of two motifs. All of these runs produced among others certain double motifs that can be used as examples for the weight dilemma (described in Chapter 2.4.5). The first motif in such motif patterns was a motif only occurring in the model sequences and never in the background sequences. The second motif was highly degenerated, up to the case, where the motif consisted of only N nucleotides. In addition, the distance between the two motifs was very high. Such a motif pattern is found extremely often in the model, and never in the background. Thus, the weight is very high. The weight dilemma tells that maximizing the weight means first and foremost the avoidance of occurrences in the background sequences, which is the case here with the first motif. In the evaluation, double motifs consisting of a motif specific to the model and a highly degenerated motif were removed after the clustering step.

Two ways are applied in order to evaluate the double motifs resulting from the motif discovery. The first way is to discard all patterns and to further work with the single motifs previously contained in the double motifs. This would mean that the single motifs need to be re-weighted in order to select the 5 highest-weighting ones. The reason for this evaluation scheme is to increase comparability of double and single motif discovery. Now, both result in 5 single motifs. The second way is to keep the double motifs and to do all evaluation steps with the binding sites reported for the motif patterns. Keeping the motif patterns intact seems most promising in reported binding sites which lie in the dense spots. But note that despite the restriction to the five best-weighting double motifs, the underlying number of single motifs might be only 3, but could also rise up to 10. In the latter case, every double motif would consist of two different single motifs.

Applying the discovery of double motifs to the low-planted sequences reveals the Pho motif in

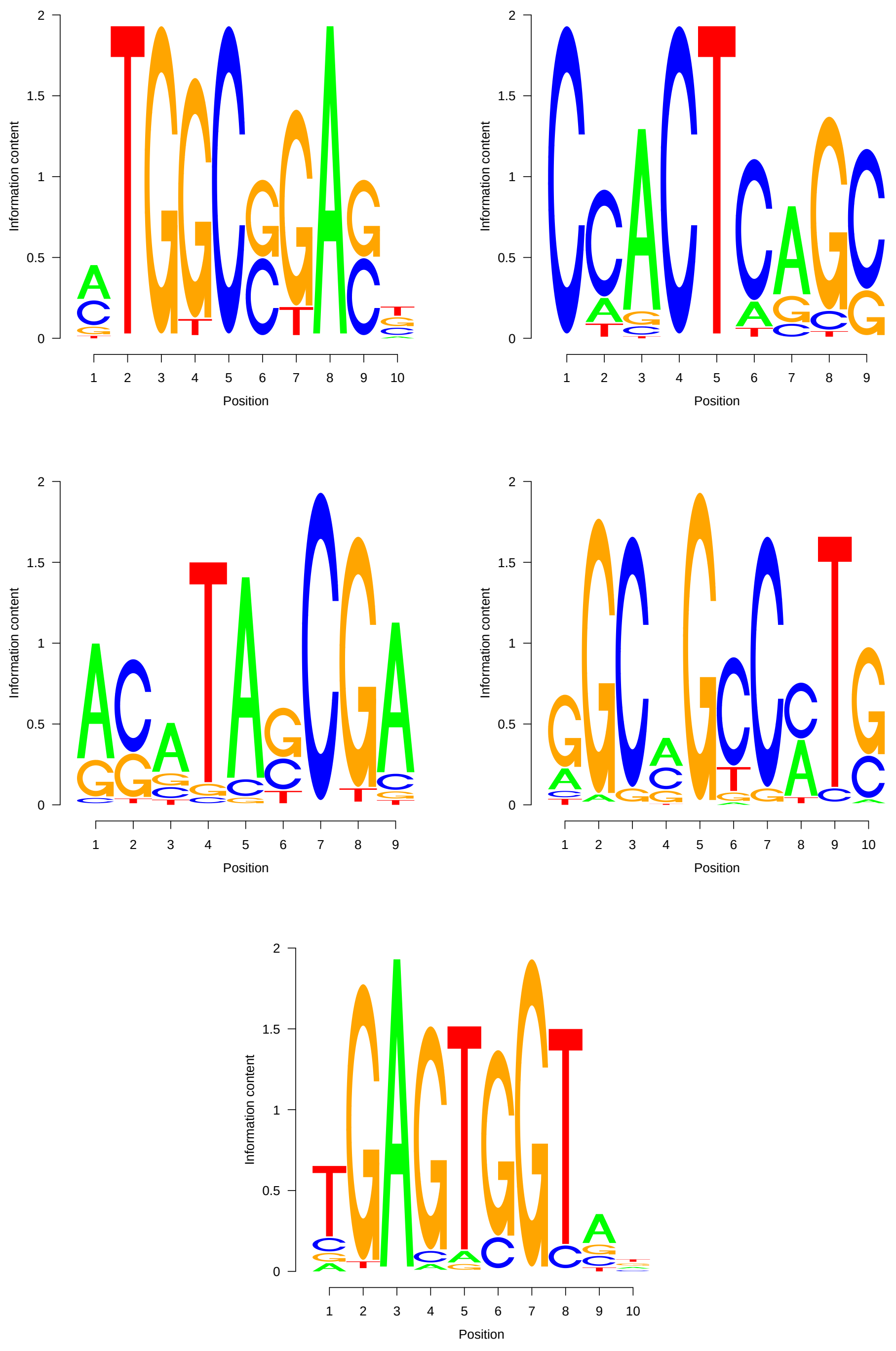


Figure 5.12: Sequence logos of the five best-weighting motifs from the single motif discovery applied to the low-planted sequences. A distorted Pho motif appears in the upper left motif (reversed complemented, positions 5 to 1) and in the middle right motif (positions 5 to 9). The upper right motif (positions 7 to 2) and the bottom motif (positions 1 to 6) resemble the Zeste motif. The GAF motif does not appear. The middle left motif is unknown.

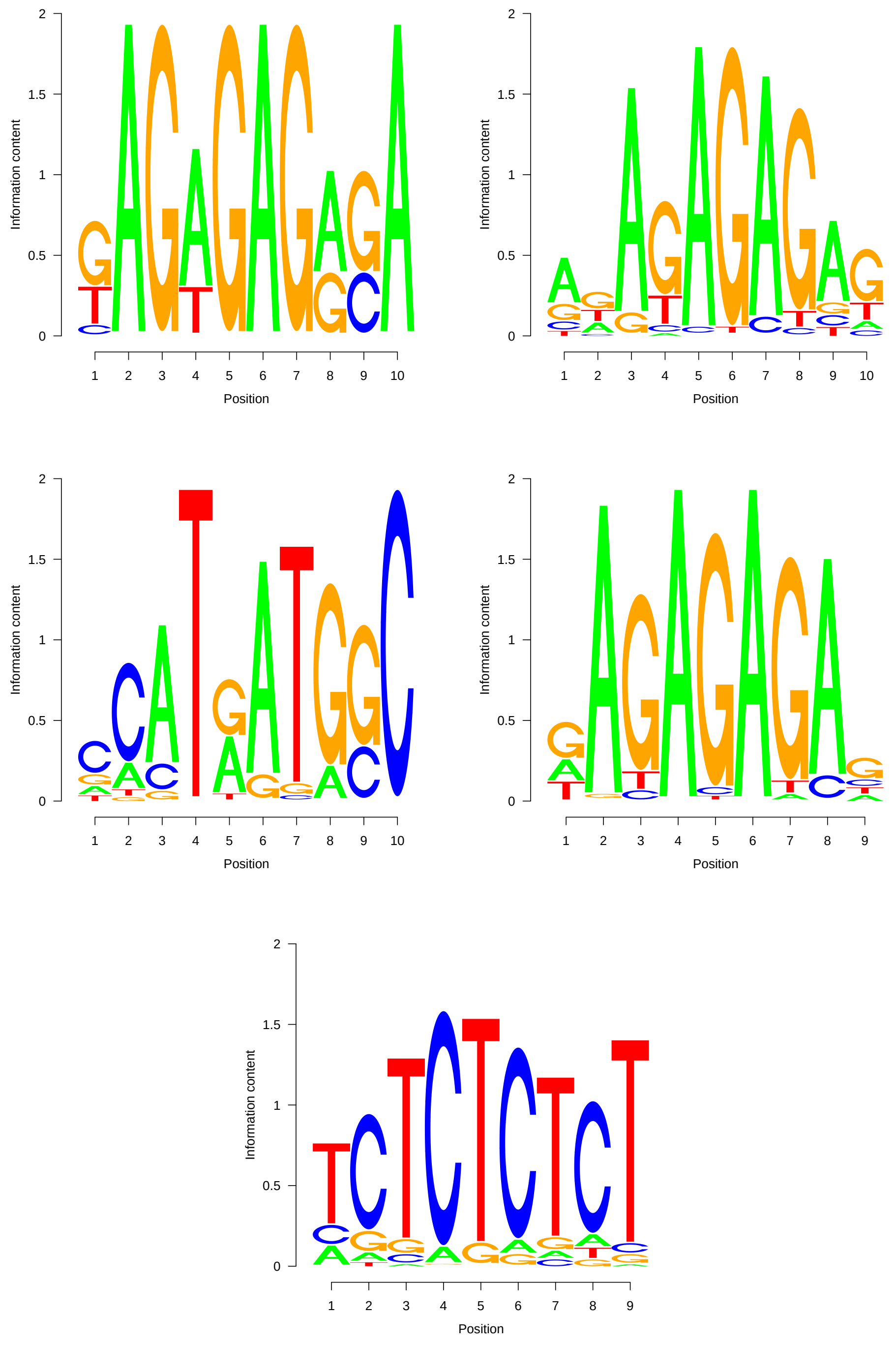


Figure 5.13: Sequence logos of the five best-weighting motifs from the single motif discovery applied to the high-planted sequences. All motifs except the middle left one more or less resemble the GAF motif (reversed complemented in the bottom motif). In the middle left motif Pho appears (as *CCAT* at position 1 to 4 and reversed complemented at position 10 to 6). The Zeste motif does not occur.

two varieties, both embedded in larger motifs. In addition, 3 unknown motifs are discovered, such as *CCA*[*ag*]*GATG*, *GCC*[*ct*][*ac*]*GCGTC*, and *G*[*cg*]*CCACGG*[*ag*]*G*. Zeste and GAF could not be found. Applying the discovery to the high-planted sequence yielded GAF in all varieties. Neither Pho nor Zeste occurred as the five best-weighted single motifs.

MotifSampler

MotifSampler [117, 88] was run with a second order Markov model obtained from *Drosophila melanogaster* as background. It was started ten times and instructed to report the five best motifs for each of the five repeats in every run. This yielded 250 motifs. After sorting for the information content, the five highest ranking, unique motifs were chosen that have non-overlapping binding sites. In the low-planted sequences, only Zeste in several varieties was among these motifs. MotifSampler was unable to combine these varieties to a more degenerated motif. In the high-planted sequences, binding sites for GAF resided at rank one, Zeste was at rank two and Pho was found at rank 5.

MEME

MEME was run without restrictions to the number of motif occurrences per sequence. The program was instructed to return the five best motifs. The minimal motif width was set to 5 and the maximal width set to 10. This reflects the settings possible with the *jPREdictor*. Motifs were allowed on both strands. As background, the second order Markov model specific to *Drosophila melanogaster* was used. MEME was run 10 times, but, afterwards, this turned out to be an unnecessary step, because every run returned the exact same 5 motifs as the best ones. In the low-planted sequences, the GAF motif was never reported. Applying MEME to the high-planted sequence, all three motifs were discovered.

BioProspector

BioProspector [89] was started in two-block mode and instructed to report the five best motif pairs. The motif width was fixed at 6 as a kind of a-priori knowledge. The distances between the two motifs was zero to 50 nucleotides. The program was started several times and the reason for this was that neither Pho nor GAF were among the five best motif pairs in the first run on the low-planted sequences. From the 5 runs, the binding sites for each best motif pair were used. This running strategy can best be described as horizontal gathering. It disagrees with the vertical strategy (start once, gather 5 best motif pairs) used by Jensen and Liu [99] in combination with the BioOptimizer program, but it gives better results at least when binding sites for more than one motif were planted. Nevertheless, the run on the low-planted sequence yielded Zeste as part of the first, second and third motif pair. In the fourth pair Pho showed up as *GCCAT* and the fifth motif pair contained the GAF sequence *GAGAG*. Applying BioProspector to the high-planted sequence, the Pho motif was never reported using the horizontal gathering. All 5 highest-scoring motif pairs contained both Zeste and GAF. Checking each result file for occurrences of Pho uncovered the Pho motif only once. This shows how weak the Pho signal is in comparison to the Zeste and GAF signals. Nevertheless, this one occurrence of Pho among all 25 motif pairs was not reason enough to change the gathering strategy.

Weeder

Weeder [123, 124], on the other hand, was only capable of finding the Pho motif. This motif was always embedded into larger motifs, either fully as *GCCAT* or without the starting *G*. Pho was reported exhaustively for all lengths, 6, 8, and 10. After instructing Weeder to report the 10 best motifs for each length, a variety of the Zeste motif, namely *TGAGCG*, was reported at rank 7 for the low-planted sequence. For the high-planted sequences, neither Zeste nor GAF were reported. One reason for missing the Zeste motif could be that Zeste contains two degenerated positions while Weeder allows for only one such position in motifs of length 6. Instructing Weeder to discover motifs of length 6 with two degenerated positions fails, because it is not supported by the program (Weeder quits with an error message). And although Weeder allows for two degenerated positions for motifs of length 8, the Zeste motif could not be discovered. This may be due to the other two nucleotides being randomly distributed, which would increase the necessary number of degenerated positions to 4. The opposite applies to finding the GAF motif. In its simplest form (length 5) no degenerated positions are necessary. Therefore, discovering motifs of length 6 and using the one degenerated position for the residual nucleotide seems to make GAF an ideal candidate for motif discovery. Nevertheless, it was not reported, neither for length 6 nor for any higher length. In order to improve the performance of Weeder, the developer suggest in the manual to mask out all reported sites and restart Weeder. However, this would have only been beneficial to discovering GAF and Zeste in case Weeder would have reported these two motifs in the list of the 10 best. But this was the case only once. As Weeder presents its results, only the sites for the first two motifs are reported. Masking out these sites would cause Weeder to report the next two sites on top position and so on. For reporting the Zeste motif (which occurred once at rank 7) at the top position, 4 restarts are necessary. In a real world application with genomic sequences, this number of restarts is normally not applicable, because the true motifs are not known beforehand. In such a case, other motif discovery programs will be used for cross checking and the most-conserved motifs will be taken for further analysis.

5.5.3 Results

In accordance to the Tompa assessment [102], several statistical parameters are measured and calculated using the binding sites predicted with the chosen motifs from the different discovery tools. In Table 5.4, these statistics are shown. In the table, both rows "Motif search" were obtained by searching the Pho, Zeste, and GAF consensus motifs in the model sequences. In a way, these rows represent the truth. Nevertheless, the low sensitivity on site-level of around 40% shows two facts. The first is that the consensus motifs do not fully cover each of the planted binding sites from TRANSFAC. The Pho motif, for instance, matches 22 out of 23 binding sites. The Zeste motif matches 21 out of 24 binding sites from Zeste. The lowest coverage shows the GAF consensus motif, as it only matches 10 out of 25 TRANSFAC binding sites for the GAF factor. However, an additional 4 GAF binding sites are covered by the Zeste motif. This means that 15 out of 72 binding sites are not covered by the consensus motifs. The second fact is that the binding sites were arbitrarily planted without keeping them from overlapping. And this could mean that some early planted binding sites were (partially) overwritten by subsequently planted ones.

For the low-planted sequences, MEME outperforms every other tool in terms of positive predictive value. This means that most of the predicted binding sites by MEME are true, both on nucleotide

Table 5.4: Statistics comparing the accuracy of several motif discovery tools. The upper part of the table gives the statistical measures for the low-planted model sequences, the lower part of the high-planted model sequences. Statistics are measured and calculated using predicted binding sites versus planted binding sites, either on site or on nucleotide level. Abbreviations used are: TP true positives, FP false positives, TN true negatives, FN false negatives, PC performance coefficient, Sn sensitivity, PPV positive predictive value, Sp specificity. Both rows "Motif search" were obtained by using the match results of the three core motifs for Pho (*GCCAT*), Zeste (*YGAGYG*), and GAF (*GAGAG*) on the model data set as predicted binding sites. The *jPREdictor* was instructed to either discover single motifs (rows "jPREdictor (single)") or to discover over-represented double motifs. For the sub-sequent matching of the discovered double motifs either the single motifs were used (rows "jPREdictor (double, S)") or the double motifs were used (rows "jPREdictor (double, D)").

Tool	Nucleotide level				Site level			Nucleotide level				Site level	
	TP	FP	TN	FN	TP	FP	FN	PC	Sn	PPV	Sp	Sn	PPV
Low-planted model sequences													
Motif search	401	3349	94074	2176	111	624	151	0.0677	0.1556	0.1069	0.9656	0.4237	0.1510
jPREdictor (single)	210	825	96598	2367	44	80	218	0.0617	0.0815	0.2029	0.9915	0.1679	0.3548
jPREdictor (double, S)	103	662	96761	2474	18	71	244	0.0318	0.0400	0.1346	0.9932	0.0687	0.2022
jPREdictor (double, D)	218	2563	94860	2359	56	461	206	0.0424	0.0846	0.0784	0.9737	0.2137	0.1083
BioProspector	496	3206	94217	2081	115	711	147	0.0858	0.1925	0.1340	0.9671	0.4389	0.1392
MEME	375	945	96478	2202	79	77	183	0.1065	0.1455	0.2841	0.9903	0.3015	0.5064
MotifSampler	64	539	96884	2513	13	91	249	0.0205	0.0248	0.1061	0.9945	0.0496	0.1250
Weeder	226	2504	94919	2351	59	415	203	0.0445	0.0877	0.0828	0.9743	0.2252	0.1245
High-planted model sequences													
Motif search	669	3402	92343	3586	199	635	299	0.0874	0.1572	0.1643	0.9645	0.3996	0.2386
jPREdictor (single)	315	599	95146	3940	73	115	425	0.0649	0.0740	0.3446	0.9937	0.1466	0.3883
jPREdictor (double, S)	305	469	95276	3950	66	155	432	0.0646	0.0717	0.3941	0.9951	0.1325	0.2986
jPREdictor (double, D)	529	1161	94584	3726	127	291	371	0.0977	0.1243	0.3130	0.9879	0.2550	0.3038
BioProspector	657	1453	94292	3598	181	494	317	0.1151	0.1544	0.3114	0.9848	0.3635	0.2681
MEME	610	1060	94685	3645	152	82	346	0.1148	0.1434	0.3653	0.9889	0.3052	0.6496
MotifSampler	158	504	95241	4097	47	96	451	0.0332	0.0371	0.2387	0.9947	0.0944	0.3287
Weeder	342	2694	93051	3913	120	461	378	0.0492	0.0804	0.1126	0.9719	0.2410	0.2065

and on site level. In terms of sensitivity, MEME is outperformed by BioProspector, but only because BioProspector predicts five times more sites. The latter fact is also reflected by the lower specificity of BioProspector in comparison to MEME. The *jPREdictor* shows an average performance. The discovery of double motifs performs on par with Weeder. In the high-planted sequences the *jPREdictor* is among the better performing tools with a high positive predictive value.

For both models the discrepancies in the *jPREdictor* performance between single and double motif discovery can be explained with discarded motifs, being not among the five best-weighted ones. The two ways of evaluating the double motifs obtained from a discovery have an impact on the performance, too. Keeping the 5 best-weighted double motifs means more predicted binding sites, and therefore higher sensitivity, because double motifs normally consist of more than 5 single motifs. However, with the higher number of predicted sites the PPV decreases. All in all, the assessment shows that the discovery of double motifs not only functions but yields reasonably good results.

Future developments should focus on automatically masking out already found sites and restarting the discovery. This would make repeating restarts of the *jPREdictor* program unnecessary. This also would solve the problem that it is probable that every run yields the exact same results. Another way of improving the discovery would be to add another requirement for double motifs in order to become discovered. Not only the double motif itself has to have a high weight, but also the single motifs comprised in the double motif need to occur over-represented.

5.6 Double motif discovery in confirmed PREs and a new prediction

In this chapter the evolution of double motifs for the prediction of PRE/TREs is presented. The reference prediction used the “SP1” motif set, under a null model of the shuffled-out *Drosophila melanogaster* genome. This scoring of shuffled-out sequences yielded a cut-off of 171 corresponding to an *E*-value of one. The genome-wide prediction itself yielded 150 PREs in the *Drosophila* genome.

The evolution of double motifs was tested in three ways. Both evolutionary approaches, “ES” and “RewardedSelection” are applied on the 12 confirmed PRE/TREs (positive training set, model) versus the $16+12=28$ heatshock and cell cycle promoters (negative training set, background). Third, the evolution with rewarded selection was applied to 12 (model) versus 28 (background) sequences of length $3,000$, randomly generated under a zeroth order Markov model.

The evolution was therefore not only applied to the PRE sequences and non-PRE promoters, but also to randomly generated sequences. Doing so provides a robustness test. The *jPREdictor*s evolution will discover over-represented motifs in the random sequences, because it is able to find over-representation in every kind of model and background (as long as both are different). Nevertheless, it is unclear, whether the discovered motifs lead to many nonsense predictions or no predicted fragments at all.

For each of the three settings, the evolution was started 50 times yielding up to 1000 unique motifs. Afterwards, the discovered motifs were selected. Two selection strategies were applied, the first was based on GC-content and the second was based on clustering. In order to perform the first strategy, the discovered motifs were rated according to their GC-content and put into bins for every 5% of GC-content. From each bin the highest weighting motif was chosen. With this approach, 20 bins are possible. Most often, however, the discovered motifs had mediocre or high GC-content. No motif was discovered with a GC-content smaller than 20%. In Figure 5.14, for all resulting double motifs

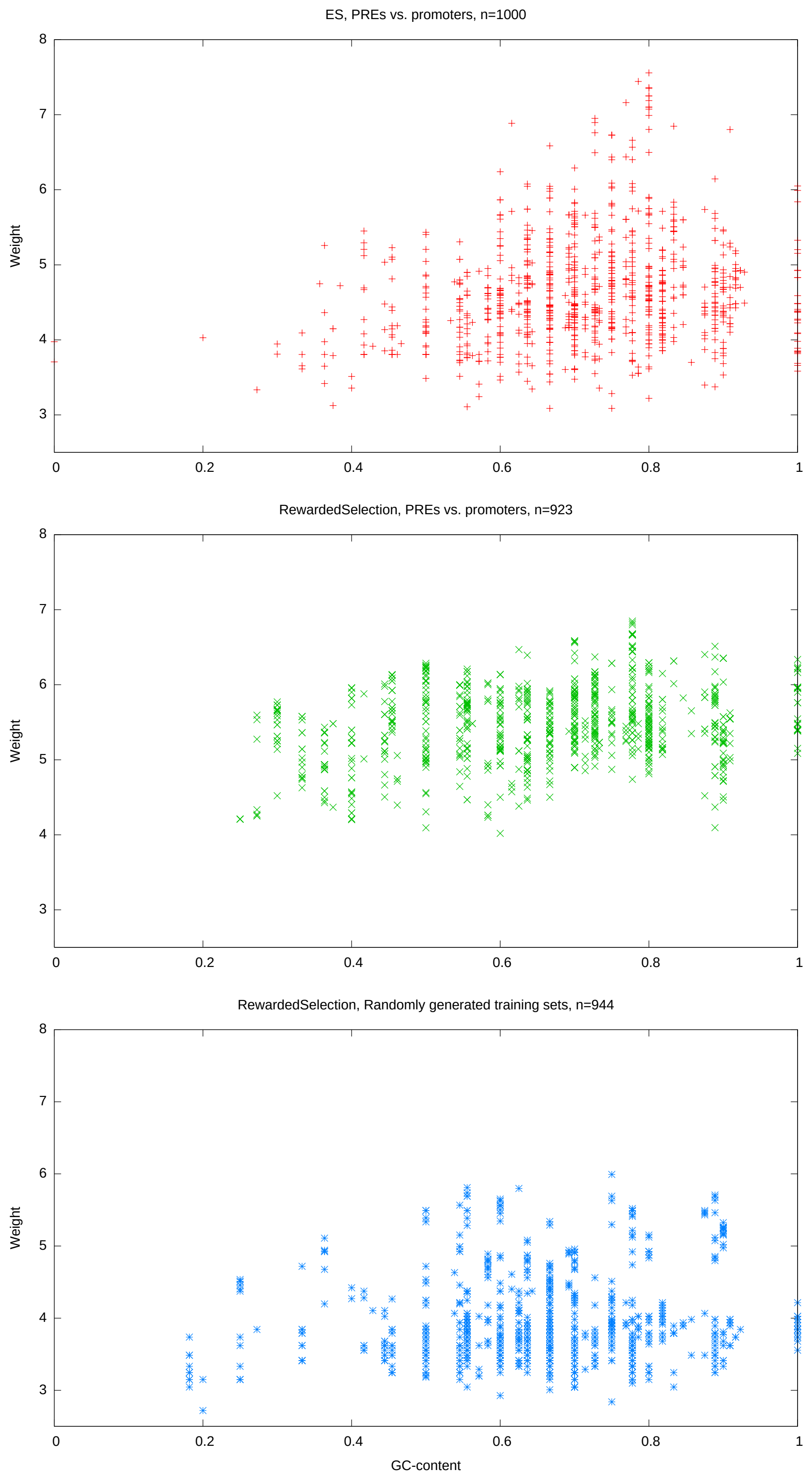


Figure 5.14: Weights and GC-content of discovered double motifs. Two evolution strategies are applied in the discovery process: "ES" and "RewardedSelection". Both are applied to the PREs vs. promoters training set, the latter also on sequences randomly generated under a zeroth order Markov model.

Table 5.5: Double motifs discovered by applying the motif evolution approach "RewardedSelection" to PREs vs. promoter sequences. For every 5% of GC-content, these motifs are the highest-weighting ones. For every double motif, the regular expression sequence together with the number of errors allowed for a match are given for both comprised basic motifs, as well as minimal and maximal distances. Note that no motif was discovered with a GC-content smaller than 25%.

First motif	Distance		Second motif	Weight	GC-content
	Min	Max			
GCGCGT′0	34	351	GGGSCG′0	5.55	0.92
CSGCCSC′1	49	413	CGTCGCT′0	6.32	0.86
GCGCRDG′0	133	405	GCGCRKV′0	6.34	0.82
GCGCRDG′0	123	427	GCSCRT′0	6.51	0.79
CMGCGAC′0	84	316	VYVTCKC′0	6.85	0.70
GTCGCTG′0	87	424	SWCTCY′0	6.59	0.65
RTCGCTG′0	87	424	CWCTCY′0	6.42	0.62
ATKGCTS′0	63	436	MATGGSYG′0	6.26	0.57
ATKGCTS′0	43	436	AABGGCYK′0	6.29	0.54
BAATGGC′0	65	355	KAATGKC′0	6.13	0.48
ARATTAAK′0	73	418	CGWCGS′0	5.96	0.43
CSATSTG′0	86	360	AACAATD′0	5.59	0.38
CGAAKAB′0	73	357	CTTATA′0	5.27	0.32
ATCAAW′0	289	416	AMTCTGT′0	4.33	0.27

of the three different settings, the weights are plotted against the GC-content. The plots show that high-weighting double motifs exist regardless of GC-content. They also show that even randomly generated training sets are capable of housing high-weighting motifs, and that the weights are not that much smaller.

In Table 5.5, the highest-weighting double motif for each 5% of GC-content resulting from the "RewardedSelection" evolution are listed. These motifs have very high weights in comparison to the SP1 motif set that serves as reference. In the SP1 motif set, only the *pssmPho:pssmPho* double motif has a weight higher than six, all other motifs have weights around and lower than three, even double motifs with negative weights occur (see Figure 4.3 on Page 65). In addition, even the double motifs evolved on randomly generated sequences have high weights. On average, the weights for such double motifs are a little lower than the weights of the double motifs obtained using the PREs vs. promoters training set (Figure 5.14, the cloud is a little bit downshifted in comparison). Nevertheless, because the motifs are sorted into bins according to their GC-content, and also because the highest-weighting double motif is taken out of each bin, the slightly "shifted" cloud has no impact on the weights of the chosen double motif used in further analysis. For the bin-selected double motifs, weights are equally high regardless of training set and initial setting.

In order to reduce the number of motifs obtained from evolution, a step other than selection after the GC-content was applied. The discovered motifs were clustered using the likelihood measure together with the "Forward" approach and a threshold of $c = 0.1$. This effectively reduced the number of motifs by a factor of about 20 (Table 5.6). After this step, the motifs from the clustering were reweighted and the 10 to 12 highest weighting ones were chosen for the prediction.

Table 5.6: Prediction of PREs using discovered motifs. The two evolutionary strategies “ES” and “RewardedSelection” are applied to confirmed PREs vs. Promoters sequences, or randomly generated sequences under a zeroth order Markov model. From the evolved motifs, the highest-weighting ones are chosen, either for every 5% of GC-content, or after a “Likelihood”-clustering (in parenthesis: number of motifs after clustering). Using the motifs, the prediction of PREs was performed genome-wide in *D. melanogaster*, in the confirmed PREs, in the promoters, in the 131 domains provided by Tolhuis et al. [22], and in the 30 domains provided by Negre et al. [23]. All predictions use the genome-wide (117 Mb) cut-off. For the Tolhuis and Negre sequences, the lower cut-off was applied, too (numbers in parenthesis).

Selection method	Double motifs	Cut-off 117 Mb	*D. mel.*	PREs	Promoters	Cut-off 3.7 Mb	Tolhuis (131)		Negre (30)	
SP1 motif set	28	171	150	2	0	73	17	(42)	0	(2)
ES on confirmed PREs vs. Promoter sequences ($n = 1000$)										
GC-content	12	311	74	1	0	144	0	(5)	0	(2)
Clustering (39)	10	230	81	1	0	129	1	(9)	1	(1)
Rewarded selection on confirmed PREs vs. Promoter sequences ($n = 923$)										
GC-content	14	221	188	2	0	142	7	(23)	0	(2)
Clustering (40)	10	132	197	2	0	88	8	(27)	0	(2)
Rewarded selection on randomly generated model and background sequences ($n = 844$)										
GC-content	14	2493	1	0	0	1818	1	(1)	0	(0)
Clustering (42)	12	88	43	0	0	60	1	(8)	0	(0)

The motif sets resulting from either of the two selection steps were then used to predict PREs in the *D. melanogaster* genome. In addition, the prediction was also applied to the PcG protein enriched sequence fragments provided by Tolhuis et al. [22] and Negre et al. [23]. For all predictions, the motifs were weighted with their corresponding training set, and the cut-off was obtained under the most restrictive null model, the shuffled-out *D. mel.* genome. The results can be found in Table 5.6.

The reference prediction using the SP1 motif set yielded 150 identified PREs. Applying the set of double motifs obtained under the “ES” evolution to a genome-wide prediction yielded 74 sequences. From these 74 sequences, none overlaps with the previously identified 150 PREs. Applying the evolution containing rewarded selection yielded 188 sequence fragments. Again, none of them overlaps with any fragment predicted with the double motifs from the “ES” strategy, but at least 13 overlap with the reference prediction. The missing overlap between the prediction results from “ES” and “RewardedSelection” shows that the motifs used in these predictions are not even similar. To be more precise, maybe they are similar but with slight alterations of their nucleotide arrangement. Because the motifs were picked from bins in accordance to their GC-content, these similarities are not discovered. The conclusion from this is that sorting and picking after the GC-content is inappropriate for discovering motifs.

Clustering the motifs and drawing the highest-weighting ones afterwards induced a significant reduction in the cut-offs in comparison to picking the motifs in accordance to their GC-content. This led to the assumption that the number of predicted PREs has to increase significantly, too. However, this was not the case: the increase in the number of predicted PREs is rather small. Also expected is an increase in the overlaps of the predicted PREs, because model and background were the same for both evolutionary approaches. However, the clustering did not standardize the motifs, and the subsequently predicted fragments did not overlap any more than using drawn motifs in accordance to their GC-content.

Applying the motifs obtained from an evolution on randomly generated training sets to a PRE prediction yielded exactly one fragment. This fragment is expected to be a false positive, because the cut-off was adjusted to correspond to an E-value of 1. This can be interpreted as a success of the robustness test. While the motif evolution on meaningless training data yields double motifs that have high weights, these double motifs neither occur clustered nor enriched in one spot in the *Drosophila* genome.

Using the genome-wide cut-off, a PRE was predicted in at most 6% of the Tolhuis sequences. Reducing the cut-off in order to adjust it to the smaller sequence length increases the number of Tolhuis fragments with at least one PRE to 21% (Table 5.6, numbers in parenthesis). As already discussed in Chapter 4.6. the reason for the low coverage might be that there exists no PRE in some of the Tolhuis sequences. However, it is far more probable that the motifs specific for the training sequences are non-relevant to the Tolhuis sequences. This assumption is contrasted by the fact that exchanging the positive training set with the Tolhuis sequences yields only slightly reduced weights for the motifs (data not shown). While the weights with the confirmed PREs as positive training set are around five to seven, the Tolhuis sequences as model yields weights around four to six. Therefore, the reason might again be that the density of relevant binding sites is not high enough to exceed the cut-off. Applying the prediction to the sequences provided by Negre et al. [23] gives zero hits for almost all motif sets (Table 5.6) and one to two sequences with at least one PRE in case the lower cut-off was applied. The same already discussed for the Tolhuis sequences applies to the Negre sequences, too.

In this chapter, motif discovery in confirmed PREs was shown with the goal to use the discovered motifs in order to predict similar fragments in the genome. This goal was achieved in the sense that the predicted fragments contained clustered and enriched motifs that also occured clustered and enriched in the confirmed PREs. However, enrichment was measured using the weight and, thus, the discovered motifs are subject to the weight dilemma. This means that the discovered motifs are more likely avoided in the promoter sequences than enriched in the confirmed PREs. This would explain the missing overlap between all predicted fragments, because very many motifs exists that are avoided in the background and all have a chance to be discovered. As a consequence, future work should focus on other strategies of drawing high-weighting motifs. One drawing strategy could be to take the motifs showing high occurrances in the model sequences regardless of their occurrences in the background. Another strategy could be to combine the discovered motifs with the already known ones and to reweight the resulting double motifs. This would lead to a good estimate on whether the discovered motifs lie near the known ones. This strategy, for instance, worked for Dejardin et al. [10], when they discovered the DSP1 motif near the Pho core motif.

6 Discussion

In this thesis, the *jPREdictor* software package was presented. It was developed as a computational tool for representing and working with motifs. The first applications incorporated into the *jPREdictor* is the prediction of Polycomb/Trithorax response elements, PRE/TREs. In addition, the *jPREdictor* can be applied to the de-novo discovery of motifs. For this, an evolutionary algorithm was developed, examining the space of possible motifs by introducing mutations into motifs or mutating specific motif characteristics. The last application is the clustering of motifs, which combines similar motifs to a probability matrix.

Design and implementation details of the *jPREdictor* are shown in Chapter 3. Development of the program started in spring 2005, and it is being updated and improved constantly. The latest version, with number 1.22, is from April 2008. The *jPREdictor* is available as Java Applet or stand-alone program for download from the following webpage:

```
http://bibiserv.techfak.uni-bielefeld.de/jpredictor
```

Since making the *jPREdictor* available on the website and the corresponding publication in 2006, [104], the program was downloaded about 180 times from unique internet sources. Note that in 2006, the motif evolution as well as the motif clustering had not been incorporated into the program. The usefulness of the *jPREdictor* was considerably increased since its launch in 2006, first by including the motif evolution in early 2007, and then by inserting the clustering approach in late 2007.

A first application is the prediction of Polycomb/Trithorax response elements, PRE/TREs. *Cis*-regulatory elements of this type are effectively identified by finding dense motif clusters. Crucial for this task is the knowledge of motifs. Fortunately, binding sites for certain transcription factors are known to occur in PRE/TREs, the most important being Pho (in the sense of a high weight). The prediction of PRE/TREs is based on the work of Ringrose et al. [5]. The authors generated double motifs out of the known motifs, weighted the motifs using model versus background sequences and afterwards performed a genome-wide scoring of the *Drosophila melanogaster* genome. Sequence fragments with a score exceeding a cut-off were assumed to be candidate PRE/TREs. It was demonstrated using ChIP experiments that the assumption was correct for a sample of predicted PRE/TREs.

Obviously, the prediction depends on the motifs used. In the early prediction of PRE/TREs the following motifs were used: En1, two motifs derived from the GAF binding sites (GA and G10), Zeste, and three variants of the Pho binding sites. In a first analysis, this set was changed in the way that the Pho motifs are combined to a position specific score matrix. Additionally, the DSP1 motif recently discovered by Dejardin et al. [10] was included. After the creation of double motifs, the prediction with the new set resulted in 344 PRE/TREs [104], in comparison to 167 predicted with the original set [5]. Both predictions were repeated using the newly introduced negative training set including more promoter sequences from the *D. mel.* genome. The new negative training set led to higher weights for the motifs, which in turn increased the cut-off score. Nevertheless, it was possible, to increase the number of predicted PRE/TREs to 301 and 481, respectively, on the same level of specificity.

Other motifs were reported to play a role in PRE/TRE functionality. Consequently, SP1 was included in the motif set, and increased the number of predicted PRE/TREs to 708. On the other hand, adding the Grainyhead motif to the “SP1” motif set had a negative impact on the number of predicted PRE/TREs: it dropped to 602. In the prediction process, adding new motifs is balanced out with a higher cut-off. This higher cut-off reflects the fact that the new motif might occur near the other motifs of the set merely by chance. In order to exceed this higher cut-off when scoring real genomic sequences, the new motif has to occur in the vicinity of the other motifs more often than expected. For SP1/KLF, this is true, but not for Grainyhead.

The prediction process was also performed with two motifs known to occur in enhancer/silencer elements, Knirps and Hunchback. This was thought as a robustness test for the prediction. The number of predicted PREs should not increase with these two motifs added to the set. For Knirps, the test was passed. On the other hand, adding Hunchback to the “SP1” motif set increased the number of predicted PREs to 934. This shows that many PRE elements comprise binding sites for Hunchback. This finding is in accordance to several reports of Hunchback binding sites to occur in PREs [80, 5] and also to reports of Hunchback playing a role as an early regulator at some PRE/TREs [113].

The choice of the background model has the highest impact on the prediction of PRE/TREs, because it influences the cut-off for the prediction score. Since the prediction is genome-wide, the best null model would be the complete genome free from the elements in question. However, such a null model is impossible, since the sought-after elements are unknown. Various null models were tested for their impact on the number of predicted PRE/TREs. Note that in every case, the cut-off was chosen to correspond to exactly one expected false positive predictions (E-value=1). Further note that every null model was derived from the *D. melanogaster* genome. With the null models representing the genome better and better, the number of predicted PRE/TREs dropped more and more. Under a zeroth order Markov model, the cut-off was 108 for the “SP1” motif set. This led to the mentioned 708 predicted PREs. Under a first and second order Markov model, the cut-off increased to 136 and 156, respectively. The highest impact on the cut-off had the shuffling of the genome. Under this null model, the cut-off was 171 resulting in 150 predicted PRE/TREs. Such a high cut-off means that only the most dense clusters of motifs had a chance to be reported as candidate PREs, which makes it very unlikely that these candidate PREs are false positives. In this sense, higher order background models do not improve prediction in terms of the number of predicted PREs but in terms of reliability.

The *jPREdictor* provides a powerful framework for the prediction of PREs and maybe other cis-regulatory elements. In order to prove, whether the predicted elements truly have regulatory function, biological approaches like ChIP experiments have to be performed. The amount of binding site enrichment and clustering in these very tight spots strongly suggests that the candidate PREs are functional PREs.

The prediction of PREs is based on known motifs. The prediction process itself is based on double motifs. They award clustering and enrichment of motifs in short fragments much more than single motif occurrences would be able to. If motifs are unknown they have to be discovered. The de-novo discovery of motifs is a very challenging task. It is based on measuring over-representation of a motif in model sequences with respect to some background.

Consequently, the second application of the *jPREdictor* is the de-novo identification of over-represented motifs. It was specifically implemented in order to find double motifs. The measure of choice for over-representation is the weight. It incorporates model and background sequences and therefore acts discriminative. Using this measure it is possible to tell whether a motif is enriched or avoided in the model with respect to the background. High positive weights hint at enrichment and over-representation, negative weights hint at avoidance. The model sequences can be promoter sequences from co-regulated genes. In this thesis, the model sequences are confirmed PRE/TRE sequences and the background sequences are promoters from heatshock and cell cycle genes. The background sequences were chosen to be in fact non-PRE/TREs, but to be upstream regions as well.

For motif discovery an evolutionary approach is used. It tries to discover motifs over-represented in the model sequences with respect to the background sequences. Based on an initial set of motifs, the search space is evaluated by mutating the motifs from one generation to the next. Parents and children compete against each other in order to maximize their weights. Simulated annealing affects this competition by introducing a random element. It also affects the mutation strength.

Motif discovery yields many high-weighting but similar motifs. In order to reduce the amount of motifs, the third application incorporated into the *jPREdictor* is the clustering of motifs. The clustering algorithm is agglomerative and hierarchical. Several distance measures are available. Two motifs are aligned in order to optimize their distance and also when they are combined into one cluster. The clustering was specifically designed to be able to cluster motifs of high complexity and of different representation. Sequence motifs can be clustered with matrix motifs, and motif patterns can be clustered together. Above all, the search orientation of the motifs is paid attention to.

By continuing an assessment initiated by Tompa et al. [102], the *jPREdictor* motif discovery approach is compared to other motif discovery tools. Tompa et al. [102] planted transcription factor binding sites from human and fly into sequence sets and the tools had to correctly predict the location of the planted binding sites. Applying the *jPREdictor* motif evolution to the data sets yielded high-weighting motifs for each data set. After clustering the motifs together, they remained high-weighting. For the human data set, the *jPREdictor* was among the better performing tools. Nevertheless, the assessment also showed that the de-novo discovery of motifs remains a very challenging task and that there is much space for improvements. The task presented some difficulties, because the noise level was very high and the data sets were small. In the fly data set, for instance, the Zeste motif was not found, because it not only occurred frequently in the model sequences, but is also occurred very often in the background set. Nevertheless, on the fly data sets, the *jPREdictor* outperformed all other tools after the sets for one binding factor were combined. This reduced the noise level to a manageable amount and it also amplified the over-representation level.

Applying the *jPREdictor* to the discovery of double motifs yielded good results. The data sets in this assessment consisted of randomly generated sequences planted with binding sites of Zeste, Pho, and GAF. Constructing the data sets was done in order to mimic the motif arrangements of the confirmed PREs as much as possible. The *jPREdictor* single motif discovery was able to successfully identify motifs for all three binding factors, while the discovery of double motifs revealed motifs for Pho and GAF. However, this assessment showed as well that the discovery built into the *jPREdictor* needs some fine-tuning in order to support the user at the decision which motifs to keep and which motifs to discard. In addition, user friendliness can be improved by implementing a masking out of already discovered binding sites.

The evolution of motifs means many free parameters, which can be varied independently. These parameters were set to the most reasonable values. For instance, mutation events on nucleotide level are much more frequent than elongating or shortening a binding site. Therefore, mutation of a nucleotide was set to an 88% probability, while changes in the length of a motif were set to occur with a probability of 5%. However, in order to investigate all settings and all combinations of settings, an additional algorithm around the motif evolution is necessary. An approach would be to fix all except one parameter. This one parameter is changed in order to investigate its impact on the motif evolution and then also fixed to its most useful setting. However, this algorithm is a task that remains for the future.

The number of mutations introduced into the motifs is changed with the temperature in the process of simulated annealing. A temperature of $1,000$ allows for 3 mutation events. This enables jumps over the search space. It is questionable, however, whether three mutation events allow for really big jumps and whether they are enough to exhaustively explore the search space. This reasoning leads to another optimal strategy for search space explorations. The strategy would allow certain motifs to survive and proliferate separately from the residual individuals of the generation. After a time, the separate populations are to be combined. This exploration scheme should also be subject to simulated annealing. In early evolution, many separate populations are allowed, while in late evolution combining the patches should be forced. Nevertheless, such a complicated evolutionary process was only incompletely incorporated into the *jPREdictor* and was not yet subject to investigations.

Another improvement of the evolutionary algorithm would be the introduction of a more sophisticated break criterion. By now, the evolution stops if the temperature value drops to zero. Another break criterion would make use of convergence. After a number of generations in which the children are not better than the parents, evolution should stop and be restarted. Obviously, an optimum is reached, either a local or a global one.

All-in-all, the *jPREdictor* software is a very powerful computational tool. It supports numerous functions when working with motifs and sequences, be it the prediction of regulatory elements, or the de-novo discovery of over-represented motifs. In addition, it is able to cluster motifs independently of their complexity. These abilities in combination with its versatility and easy-to-use interfaces make the software a leading capacity in the field of bioinformatics.

A Additional tables

Table A.1: Assessment results for the single human data sets. Abbreviations used: PC performance coefficient, PPV positive predictive value.

Data set	PC	Nucleotide level			Site level	
		Sensitivity	PPV	Specificity	Sensitivity	PPV
hm01g	0	0	0	0.9958	0	0
hm01m	0	0	0	0.9925	0	0
hm01r	0.0055	0.0085	0.0154	0.9964	0.0625	0.0769
hm02g	0	0	0	0.9897	0	0
hm02m	0	0	0	0.9920	0	0
hm02r	0	0	0	0.9953	0	0
hm03g	0.0063	0.0074	0.0429	0.9954	0	0
hm03m	0.0078	0.0098	0.0364	0.9927	0	0
hm03r	0.0079	0.0098	0.0400	0.9934	0	0
hm04g	0.0459	0.0595	0.1667	0.9981	0	0
hm04m	0	0	0	0.9933	0	0
hm04r	0	0	0	0.9961	0	0
hm05g	0	0	0	0.9857	0	0
hm05m	0	0	0	0.9822	0	0
hm05r	0	0	0	0.9822	0	0
hm06g	0.5253	0.6933	0.6842	0.9946	0.6667	0.8571
hm06m	0	0	0	0.9765	0	0
hm06r	0	0	0	0.9842	0	0
hm07g	0	0	0	0.9838	0	0
hm07m	0.0565	0.0787	0.1667	0.9897	0.1667	0.1667
hm07r	0	0	0	0.9934	0	0
hm08g	0.3592	0.3895	0.8222	0.9978	0.6154	0.8889
hm08m	0.0069	0.0105	0.0194	0.9862	0.0769	0.0769
hm08r	0.3456	0.3947	0.7353	0.9963	0.6154	0.8000
hm09g	0	0	0	0.9908	0	0
hm09m	0	0	0	0.9894	0	0
hm09r	0	0	0	0.9910	0	0
hm10g	0	0	0	0.9859	0	0
hm10m	0	0	0	0.9863	0	0
continued...						

Continuing Table A.1

Data set	PC	Nucleotide level			Site level	
		Sensitivity	PPV	Specificity	Sensitivity	PPV
hm10r	0	0	0	0.9828	0	0
hm11g	0	0	0	0.9855	0	0
hm11m	0	0	0	0.9909	0	0
hm11r	0.0257	0.0297	0.1600	0.9946	0	0
hm12g	0.1250	0.1429	0.5000	0.9892	0.2000	0.5000
hm12m	0	0	0	0.9753	0	0
hm12r	0.1340	0.1857	0.3250	0.9710	0.4000	0.5000
hm13g	0	0	0	0.9931	0	0
hm13m	0	0	0	0.9863	0	0
hm13r	0.1231	0.1463	0.4364	0.9947	0.2222	0.4000
hm14g	0	0	0	0.9896	0	0
hm14m	0	0	0	0.9791	0	0
hm14r	0.0980	0.1220	0.3333	0.9896	0.2500	0.3333
hm15g	0	0	0	0.9937	0	0
hm15m	0	0	0	0.9886	0	0
hm15r	0	0	0	0.9949	0	0
hm16g	0	0	0	0.9971	0	0
hm16m	0	0	0	0.9911	0	0
hm16r	0	0	0	0.9966	0	0
hm17g	0.2202	0.2552	0.6167	0.9957	0.4000	0.6667
hm17m	0	0	0	0.9851	0	0
hm17r	0.1598	0.1862	0.5294	0.9955	0.3000	0.5000
hm18g	0	0	0	0.9953	0	0
hm18m	0	0	0	0.9893	0	0
hm18r	0	0	0	0.9966	0	0
hm19g	0.0496	0.0690	0.1500	0.9859	0.2500	0.2500
hm19m	0	0	0	0.9876	0	0
hm19r	0	0	0	0.9834	0	0
hm20g	0	0	0	0.9963	0	0
hm20m	0.0049	0.0069	0.0167	0.9923	0.0132	0.0185
hm20r	0	0	0	0.9980	0	0
hm21g	0	0	0	0.9870	0	0
hm21m	0.0362	0.0538	0.1000	0.9908	0.1429	0.2000
hm21r	0	0	0	0.9916	0	0
hm22g	0	0	0	0.9793	0	0
hm22m	0	0	0	0.9862	0	0
hm22r	0	0	0	0.9793	0	0
hm23g	0	0	0	0.9515	0	0
hm23m	0.0613	0.0699	0.3333	0.9892	0.2000	0.3333

continued...

Continuing Table A.1

Data set	PC	Nucleotide level			Site level	
		Sensitivity	PPV	Specificity	Sensitivity	PPV
hm23r	0	0	0	0.9838	0	0
hm24g	0	0	0	0.9719	0	0
hm24m	0.1722	0.2826	0.3059	0.9849	0.3750	0.3750
hm24r	0	0	0	0.9780	0	0
hm25g	0.1111	0.1429	0.3333	0.9785	0.2000	0.3333
hm25m	0	0	0	0.9785	0	0
hm25r	0.0909	0.1429	0.2000	0.9570	0.2000	0.2000
hm26g	0	0	0	0.9909	0	0
hm26m	0	0	0	0.9886	0	0
hm26r	0	0	0	0.9931	0	0
Human	0.0206	0.0287	0.0683	0.9929	0.0526	0.0679

Table A.2: Transition probabilities for a first order Markov model (MC1), obtained by a genome-wide analysis of binucleotides in the *D. melanogaster* genome. Each probability is defined as $p(a_1|a_0)$. The MC1 table consists of 16 probabilities, i.e. 12 free parameters. This table can directly be used as input for the "mksequ" program in order to generate random sequences under a Markov model.

First	Second nucleotide a_1			
a_0	A	C	G	T
A	0.3504	0.1812	0.1884	0.2800
C	0.3254	0.2223	0.1972	0.2552
G	0.2607	0.2715	0.2223	0.2455
T	0.2172	0.1926	0.2399	0.3504

Table A.3: Transition probabilities for a second order Markov model (MC2), obtained by a genome-wide analysis of trinucleotides in the *D. melanogaster* genome. Each probability is defined as $p(a_2|a_0a_1)$. The MC2 table consists of 64 probabilities, i.e. 48 free parameters. This table can directly be used as input for the "mksequ" program in order to generate random sequences under a Markov model.

First	Third	Second nucleotide a_1			
a_0	a_2	A	C	G	T
A	A	0.3625	0.4159	0.3220	0.2603
A	C	0.1628	0.2154	0.2035	0.1767
A	G	0.2326	0.0485	0.2419	0.2591
A	T	0.2421	0.3203	0.2326	0.3039
C	A	0.2501	0.3459	0.2345	0.1875
C	C	0.2064	0.2559	0.2167	0.2278
C	G	0.2834	0.0478	0.2654	0.2874
C	T	0.2601	0.3503	0.2835	0.2973
G	A	0.3369	0.3474	0.3081	0.2220
G	C	0.1762	0.2515	0.2094	0.2031
G	G	0.2720	0.0468	0.2558	0.2883
G	T	0.2148	0.3543	0.2266	0.2865
T	A	0.3122	0.3266	0.2695	0.2181
T	C	0.1807	0.2539	0.1970	0.2208
T	G	0.2102	0.0348	0.2355	0.1986
T	T	0.2969	0.3846	0.2980	0.3625

Table A.4: Sequence fragments added to the negative training set. All fragments originate from *Drosophila melanogaster* and are upstream regions of genes dedicated to cell cycle. If the start position is larger than the end position, the fragment was reversed complemented. Abbreviations: Chr=Chromosome.

Code	Adjacent gene downstream	or fragment upstream	Type	Chr	Start	End	Length
promCycA	Cyclin A	CG12521	Promoter	3L	11,840,434	11,826,678	13,757
promCycB	Cyclin B	st1	Promoter	2R	18,696,491	18,694,491	2,001
promRux	roughex	CG5941	Promoter	X	5,931,563	5,931,210	354
promE2f	E2F	CG6353	Promoter	3R	17,486,907	17,486,128	780
intronE2f	E2F-RB first exon	E2F-RB second exon	Intron	3R	17,485,937	17,471,266	14,672
promDap	dacapo	CG10459	Promoter	2R	5,596,045	5,599,827	3,783
promDp	Umbrea	CG15636	Promoter	2L	4,606,963	4,591,964	15,000
promCdc2c	cdc2c	CG31205	Promoter	3R	16,558,265	16,561,341	3,077
VanIntrFor-Cdc37	dlt-RA second exon	dlt-RA first exon	Intron	3L	1,793,501	1,795,198	1,698
promCycE	Cyclin E	CG13240	Promoter	2L	15,749,608	15,748,156	1,453
promCdk4	Cyclin-dependent kinase 4	CG8317	Promoter	2R	12,465,534	12,466,980	1,447
promCdk8	Cyclin-dependent kinase 8	GTPase-activating protein 1	Promoter	3L	9,832,194	9,831,751	444

Bibliography

[1] International Human Genome Sequencing Consortium (Oct, 2004) Finishing the euchromatic sequence of the human genome. *Nature,* **431**(7011), 931–945.

[2] Adams, M. D., et al. (Mar, 2000) The genome sequence of Drosophila melanogaster. *Science,* **287**(5461), 2185–2195.

[3] Ringrose, L. and Paro, R. (2004) Epigenetic regulation of cellular memory by the Polycomb and Trithorax group proteins. *Annu Rev Genet,* **38**, 413–443.

[4] Lifanov, A. P., Makeev, V. J., Nazina, A. G., and Papatsenko, D. A. (Apr, 2003) Homotypic Regulatory Clusters in Drosophila. *Genome Res,* **13**(4), 579–588.

[5] Ringrose, L., Rehmsmeier, M., Dura, J.-M., and Paro, R. (2003) Genome-Wide Prediction of Polycomb/Trithorax Response Elements in Drosophila melanogaster. *Dev Cell,* **5**, 759–771.

[6] Buck, M. J. and Lieb, J. D. (Mar, 2004) ChIP-chip: considerations for the design, analysis, and application of genome-wide chromatin immunoprecipitation experiments. *Genomics,* **83**(3), 349–360.

[7] Bantignies, F. and Cavalli, G. (2006) Cellular memory and dynamic regulation of polycomb group proteins. *Curr Opin Cell Biol,* **18**(3), 275–283.

[8] Lehmann, M. (Jan, 2004) Anything else but GAGA: a nonhistone protein complex reshapes chromatin structure. *Trends Genet,* **20**(1), 15–22.

[9] Brown, J. L., Grau, D. J., DeVido, S. K., and Kassis, J. A. (2005) An Sp1/KLF binding site is important for the activity of a Polycomb group response element from the Drosophila engrailed gene. *Nucleic Acids Res,* **33**(16), 5181–5189.

[10] Déjardin, J., Rappailles, A., Cuvier, O., Grimaud, C., Decoville, M., Locker, D., and Cavalli, G. (2005) Recruitment of Drosophila Polycomb group proteins to chromatin by DSP1. *Nature,* **434**, 533–538.

[11] Blastyák, A., Mishra, R. K., Karch, F., and Gyurkovics, H. (2006) Efficient and specific targeting of Polycomb group proteins requires cooperative interaction between Grainyhead and Pleiohomeotic. *Mol Cell Biol,* **26**(4), 1434–1444.

[12] Sanchez-Elsner, T., Gou, D., Kremmer, E., and Sauer, F. (Feb, 2006) Noncoding RNAs of trithorax response elements recruit Drosophila Ash1 to Ultrabithorax. *Science,* **311**(5764), 1118–1123.

[13] Petruk, S., Sedkov, Y., Riley, K. M., Hodgson, J., Schweisguth, F., Hirose, S., Jaynes, J. B., Brock, H. W., and Mazo, A. (Dec, 2006) Transcription of bxd noncoding RNAs promoted by trithorax represses Ubx in cis by transcriptional interference. *Cell,* **127**(6), 1209–1221.

[14] Müller, J. and Kassis, J. A. (Oct, 2006) Polycomb response elements and targeting of Polycomb group proteins in Drosophila. *Curr Opin Genet Dev,* **16**(5), 476–484.

[15] Klymenko, T., Papp, B., Fischle, W., Köcher, T., Schelder, M., Fritsch, C., Wild, B., Wilm, M., and Müller, J. (May, 2006) A Polycomb group protein complex with sequence-specific DNA-binding and selective methyl-lysine-binding activities. *Genes Dev,* **20**(9), 1110–1122.

[16] Petruk, S., Sedkov, Y., Brock, H. W., and Mazo, A. (2007) A model for initiation of mosaic HOX gene expression patterns by non-coding RNAs in early embryos. *RNA Biol,* **4**(1), 1–6.

[17] Papp, B. and Müller, J. (Aug, 2006) Histone trimethylation and the maintenance of transcriptional ON and OFF states by trxG and PcG proteins. *Genes Dev,* **20**(15), 2041–2054.

[18] Schwartz, Y. B., Kahn, T. G., Nix, D. A., Li, X.-Y., Bourgon, R., Biggin, M., and Pirrotta, V. (Jun, 2006) Genome-wide analysis of Polycomb targets in Drosophila melanogaster. *Nat Genet,* **38**(6), 700–705.

[19] Ebert, A., Lein, S., Schotta, G., and Reuter, G. (2006) Histone modification and the control of heterochromatic gene silencing in Drosophila. *Chromosome Res,* **14**(4), 377–392.

[20] Rudolph, T., Yonezawa, M., Lein, S., Heidrich, K., Kubicek, S., Schäfer, C., Phalke, S., Walther, M., Schmidt, A., Jenuwein, T., and Reuter, G. (Apr, 2007) Heterochromatin formation in Drosophila is initiated through active removal of H3K4 methylation by the LSD1 homolog SU(VAR)3-3. *Mol Cell,* **26**(1), 103–115.

[21] Ringrose, L. (Jun, 2007) Polycomb comes of age: genome-wide profiling of target sites. *Curr Opin Cell Biol,* **19**(3), 290–297.

[22] Tolhuis, B., deWit, E., Muijrers, I., Teunissen, H., Talhout, W., vanSteensel, B., and van-Lohuizen, M. (Jun, 2006) Genome-wide profiling of PRC1 and PRC2 Polycomb chromatin binding in Drosophila melanogaster. *Nat Genet,* **38**(6), 694–699.

[23] Nègre, N., Hennetin, J., Sun, L. V., Lavrov, S., Bellis, M., White, K. P., and Cavalli, G. (Jun, 2006) Chromosomal distribution of PcG proteins during Drosophila development. *PLoS Biol,* **4**(6), e170.

[24] Prum, B., Rodolphe, F., and deTurckheim, E. (1995) Finding Words with Unexpected Frequencies in Deoxyribonucleic Acid Sequences. *J R Stat Soc,* **B 57**, 205–220.

[25] Lifton, R. P., Goldberg, M. L., Karp, R. W., and Hogness, D. S. (1978) The organization of the histone genes in Drosophila melanogaster: functional and evolutionary implications. *Cold Spring Harb Symp Quant Biol,* **42 Pt 2**, 1047–1051.

[26] Kutach, A. K. and Kadonaga, J. T. (Jul, 2000) The downstream promoter element DPE appears to be as widely used as the TATA box in Drosophila core promoters. In *Mol Cell Biol* [125] pp. 4754–4764.

[27] Ohler, U., Liao, G., Niemann, H., and Rubin, G. M. (2002) Computational analysis of core promoters in the Drosophila genome. In *Genome Biol* [125] pp. 1–87.

[28] Staden, R. (1989) Methods for calculating the probabilities of finding patterns in sequences. *Comput Appl Biosci,* **5**(2), 89–96.

[29] Robin, S. and Daudin, J. J. (1999) Exact distribution of word occurrences in a random sequence of letters. *J Appl Probab,* **36**(1), 179–193.

[30] Reinert, G., Schbath, S., and Waterman, M. Applied Combinatorics on Words chapter 6. Statistics on words with applications to biological sequences, pp. 251–328 Number 105 in Encyclopedia of Mathematics and its Applications Cambridge University Press (2005).

[31] Blom, G. (1982) On the mean number of random digits until a given sequence occurs. *J Appl Probab,* **19**, 136–143.

[32] Blom, G. and Thorburn, D. (1982) How many random digits are required until given sequences are obtained?. *J Appl Probab,* **19**, 518–531.

[33] Liébecq, C., (ed.) (1992) Biochemical Nomenclature and Related Documents, Portland Press, 2nd edition.

[34] Down, T. A., Bergman, C. M., Su, J., and Hubbard, T. J. P. (Jan, 2007) Large-scale discovery of promoter motifs in Drosophila melanogaster. *PLoS Comput Biol,* **3**(1), e7.

[35] Bergman, C. M., Carlson, J. W., and Celniker, S. E. (Apr, 2005) Drosophila DNase I footprint database: a systematic genome annotation of transcription factor binding sites in the fruitfly, Drosophila melanogaster. *Bioinformatics,* **21**(8), 1747–1749.

[36] Galas, D. J. and Schmitz, A. (Sep, 1978) DNAse footprinting: a simple method for the detection of protein-DNA binding specificity. *Nucleic Acids Res,* **5**(9), 3157–3170.

[37] Pollock, R. and Treisman, R. (Nov, 1990) A sensitive method for the determination of protein-DNA binding specificities. *Nucleic Acids Res,* **18**(21), 6197–6204.

[38] Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., and Wootton, J. C. (Oct, 1993) Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science,* **262**(5131), 208–214.

[39] Thompson, W., Rouchka, E. C., and Lawrence, C. E. (Jul, 2003) Gibbs Recursive Sampler: finding transcription factor binding sites. *Nucleic Acids Res,* **31**(13), 3580–3585.

[40] Thompson, W. A., Newberg, L. A., Conlan, S., McCue, L. A., and Lawrence, C. E. (Jul, 2007) The Gibbs Centroid Sampler. *Nucleic Acids Res,* **35**(Web Server issue), W232–W237.

[41] Kassis, J. A., Desplan, C., Wright, D. K., and O'Farrell, P. H. (1989) Evolutionary conservation of homeodomain-binding sites and other sequences upstream and within the major transcription unit of the Drosophila segmentation gene engrailed. *Mol Cell Biol,* **9**(10), 4304–4311.

[42] Mihaly, J., Mishra, R. K., and Karch, F. (1998) A Conserved Sequence Motif in Polycomb-Response Elements. *Dev Cell,* **1**(7), 1065–1066.

[43] Brown, J. L., Mucci, D., Whiteley, M., Dirksen, M.-L., and Kassis, J. A. (1998) The Drosophila Polycomb Group Gene pleiohomeotic Encodes a DNA Binding Protein with Homology to the Transcription Factor YY1. *Mol Cell,* **1**(7), 1057–1064.

[44] Fritsch, C., Brown, J. L., Kassis, J. A., and Müller, J. (1999) The DNA-binding polycomb group protein pleiohomeotic mediates silencing of a Drosophila homeotic gene. *Development,* **126**(17), 3905–3913.

[45] Brown, J. L., Fritsch, C., Mueller, J., and Kassis, J. A. (2003) The Drosophila pho-like gene encodes a YY1-related DNA binding protein that is redundant with pleiohomeotic in homeotic gene silencing. *Development,* **130**, 285–294.

[46] Schneider, T. D. and Stephens, R. M. (Oct, 1990) Sequence logos: a new way to display consensus sequences. *Nucleic Acids Res,* **18**(20), 6097–6100.

[47] Crooks, G. E., Hon, G., Chandonia, J.-M., and Brenner, S. E. (Jun, 2004) WebLogo: a sequence logo generator. *Genome Res,* **14**(6), 1188–1190.

[48] Bailey, T. L. and Gribskov, M. (1998) Combining evidence using p-values: application to sequence homology searches. *Bioinformatics,* **14**(1), 48–54.

[49] Wu, T. D., Nevill-Manning, C. G., and Brutlag, D. L. (2000) Fast Probabilistic Analysis of Sequence Function Using Scoring Matrices. *Bioinformatics,* **16**(3), 233–244.

[50] Tronche, F., Ringeisen, F., Blumenfeld, M., Yaniv, M., and Pontoglio, M. (Feb, 1997) Analysis of the distribution of binding sites for a tissue-specific transcription factor in the vertebrate genome. *J Mol Biol,* **266**(2), 231–245.

[51] Mishra, R. K., Mihaly, J., Barges, S., Spierer, A., Karch, F., Hagstrom, K., Schweinsberg, S. E., and Schedl, P. (2001) The iab-7 Polycomb Response Element Maps to a Nucleosome-Free Region of Chromatin and Requires Both GAGA and Pleiohomeotic for Silencing Activity. *Mol Cell Biol,* **21**(4), 1311–1318.

[52] Barash, Y., Elidan, G., Kaplan, T., and Friedman, N. (2005) CIS: compound importance sampling method for protein-DNA binding site p-value estimation. *Bioinformatics,* **21**(5), 596–600.

[53] Claverie, J.-M. and Audic, S. (1996) The statistical significance of nucleotide position-weight matrix matches. *Comput Appl Biosci,* **12**(5), 431–439.

[54] Tatusov, R. L., Altschul, S. F., and Koonin, E. V. (Dec, 1994) Detection of conserved segments in proteins: iterative scanning of sequence databases with alignment blocks. *Proc Natl Acad Sci U S A,* **91**(25), 12091–12095.

[55] Quandt, K., Frech, K., Karas, H., Wingender, E., and Werner, T. (1995) MatInd and MatInspector: new fast and versatile tools for detection of consensus matches in nucleotide sequence data. *Nucleic Acids Res,* **23**(23), 4878–4884.

[56] Elkon, R., Linhart, C., Sharan, R., Shamir, R., and Shiloh, Y. (2003) Genome-Wide In Silico Identification of Transcriptional Regulators Controlling the Cell Cycle in Human Cells. *Genome Res,* **13**(5), 773–780.

[57] Beckstette, M., Strothmann, D., Homann, R., Giegerich, R., and Kurtz, S. (2004) PoSSuMsearch: Fast and Sensitive Matching of Position Specific Scoring Matrices using Enhanced Suffix Arrays. *Proc. GCB 2004,* **P-53**, 53–64.

[58] Abouelhoda, M. I. and Ohlebusch, E. (2003) A Local Chaining Algorithm and Its Applications in Comparative Genomics. *Algorithms in Bioinformatics,* **2812**, 1–16.

[59] Mosig, A., Chen, J. J.-L., and Stadler, P. F. (2007) Homology Search with Fragmented Nucleic Acid Sequence Patterns. *Algorithms in Bioinformatics,* **4645**, 335–345.

[60] Grillo, G., Licciulli, F., Liuni, S., Sbisà, E., and Pesole, G. (Jul, 2003) PatSearch: A program for the detection of patterns and structural motifs in nucleotide sequences. *Nucleic Acids Res,* **31**(13), 3608–3612.

[61] Finn, R. D., Mistry, J., Schuster-Böckler, B., Griffiths-Jones, S., Hollich, V., Lassmann, T., Moxon, S., Marshall, M., Khanna, A., Durbin, R., Eddy, S. R., Sonnhammer, E. L. L., and Bateman, A. (Jan, 2006) Pfam: clans, web tools and services. *Nucleic Acids Res,* **34**(Database issue), D247–D251.

[62] Finn, R. D., Tate, J., Mistry, J., Coggill, P. C., Sammut, S. J., Hotz, H.-R., Ceric, G., Forslund, K., Eddy, S. R., Sonnhammer, E. L. L., and Bateman, A. (Jan, 2008) The Pfam protein families database. *Nucleic Acids Res,* **36**(Database issue), D281–D288.

[63] Gallo, S. M., Li, L., Hu, Z., and Halfon, M. S. (Feb, 2006) REDfly: a Regulatory Element Database for Drosophila. *Bioinformatics,* **22**(3), 381–383.

[64] Adryan, B. and Teichmann, S. A. (Jun, 2006) FlyTF: a systematic review of site-specific transcription factors in the fruit fly Drosophila melanogaster. *Bioinformatics,* **22**(12), 1532–1533.

[65] Crosby, M. A., Goodman, J. L., Strelets, V. B., Zhang, P., Gelbart, W. M., and Consortium, F. (Jan, 2007) FlyBase: genomes by the dozen. *Nucleic Acids Res,* **35**(Database issue), D486–D491.

[66] Lyne, R., Smith, R., Rutherford, K., Wakeling, M., Varley, A., Guillier, F., Janssens, H., Ji, W., McLaren, P., North, P., Rana, D., Riley, T., Sullivan, J., Watkins, X., Woodbridge, M., Lilley, K., Russell, S., Ashburner, M., Mizuguchi, K., and Micklem, G. (Jul, 2007) FlyMine: an integrated database for Drosophila and Anopheles genomics. *Genome Biol,* **8**(7), R129.

[67] Matys, V., Fricke, E., Geffers, R., Gössling, E., Haubrock, M., Hehl, R., Hornischer, K., Karas, D., Kel, A. E., Kel-Margoulis, O. V., Kloos, D.-U., Land, S., Lewicki-Potapov, B., Michael, H., Münch, R., Reuter, I., Rotert, S., Saxel, H., Scheer, M., Thiele, S., and Wingender, E. (Jan, 2003) TRANSFAC: transcriptional regulation, from patterns to profiles. *Nucleic Acids Res,* **31**(1), 374–378.

[68] Strutt, H., Cavalli, G., and Paro, R. (1997) Co-localization of Polycomb protein and GAGA factor on regulatory elements responsible for the maintenance of homeotic gene expression. *EMBO J,* **16**(12), 3621–3632.

[69] Hur, M.-W., Laney, J. D., Jeon, S.-H., Ali, J., and Biggin, M. D. (2002) Zeste maintains repression of Ubx transgenes: support for a new model of Polycomb repression. *Development,* **129**, 1339–1343.

[70] Wilkins, R. C. and Lis, J. T. (Jun, 1998) GAGA factor binding to DNA via a single trinucleotide sequence element. *Nucleic Acids Res,* **26**(11), 2672–2678.

[71] Omichinski, J. G., Pedone, P. V., Felsenfeld, G., Gronenborn, A. M., and Clore, G. M. (Feb, 1997) The solution structure of a specific GAGA factor-DNA complex reveals a modular binding mode. *Nat Struct Biol,* **4**(2), 122–132.

[72] Wilkins, R. C. and Lis, J. T. (Oct, 1997) Dynamics of potentiation and activation: GAGA factor and its role in heat shock gene regulation. *Nucleic Acids Res,* **25**(20), 3963–3968.

[73] vanSteensel, B., Delrow, J., and Bussemaker, H. J. (2003) Genomewide analysis of Drosophila GAGA factor target genes reveals context-dependent DNA binding. *Proc Natl Acad Sci USA,* **100**(5), 2580–2585.

[74] Benson, M. and Pirrotta, V. (Dec, 1988) The Drosophila zeste protein binds cooperatively to sites in many gene regulatory regions: implications for transvection and gene regulation. *EMBO J,* **7**(12), 3907–3915.

[75] Mohrmann, L., Kal, A. J., and Verrijzer, C. P. (Dec, 2002) Characterization of the extended Myb-like DNA-binding domain of trithorax group protein Zeste. *J Biol Chem,* **277**(49), 47385–47392.

[76] Berman, B. P., Nibu, Y., Pfeiffer, B. D., Tomancak, P., Celniker, S. E., Levine, M., Rubin, G. M., and Eisen, M. B. (2002) Exploiting transcription factor binding site clustering to identify cis-regulatory modules involved in pattern formation in the Drosophila genome. *Proc Natl Acad Sci USA,* **99**(2), 757–762.

[77] Berman, B. P., Pfeiffer, B. D., Laverty, T. R., Salzberg, S. L., Rubin, G. M., Eisen, M. B., and Celniker, S. E. (2004) Computational identification of developmental enhancers: conservation and function of transcription factor binding-site clusters in Drosophila melanogaster and Drosophila pseudoobscura. *Genome Biol,* **5**(9), R61.

[78] Chen, J. D. and Pirrotta, V. (May, 1993) Multimerization of the Drosophila zeste protein is required for efficient DNA binding. *EMBO J,* **12**(5), 2075–2083.

[79] Laney, J. D. and Biggin, M. D. (1996) Redundant control of Ultrabithorax by zeste involves functional levels of zeste protein binding at the Ultrabithorax promoter. *Development,* **122**(7), 2303–2311.

[80] Mahmoudi, T., Zuijderduijn, L. M. P., Mohd-Sarip, A., and Verrijzer, C. P. (2003) GAGA facilitates binding of Pleiohomeotic to a chromatinized Polycomb response element. *Nucleic Acids Res,* **31**(14), 4147–4156.

[81] Li, M., Ma, B., and Wang, L. (2002) Finding Similar Regions in Many Sequences. *J Comput Syst Sci,* **65**, 73–96.

[82] Das, M. K. and Dai, H.-K. (2007) A survey of DNA motif finding algorithms. *BMC Bioinformatics,* **8 Suppl 7**, S21.

[83] Sinha, S. and Tompa, M. (Dec, 2002) Discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Res,* **30**(24), 5549–5560.

[84] Sinha, S. and Tompa, M. (Jul, 2003) YMF: A program for discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Res,* **31**(13), 3586–3588.

[85] Bailey, T. L., Williams, N., Misleh, C., and Li, W. W. (Jul, 2006) MEME: discovering and analyzing DNA and protein sequence motifs. *Nucleic Acids Res,* **34**(Web Server issue), W369–W373.

[86] Hertz, G. Z. and Stormo, G. D. (1999) Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics,* **15**(7-8), 563–577.

[87] Roth, F. P., Hughes, J. D., Estep, P. W., and Church, G. M. (Oct, 1998) Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nat Biotechnol,* **16**(10), 939–945.

[88] Thijs, G., Lescot, M., Marchal, K., Rombauts, S., Moor, B. D., Rouzé, P., and Moreau, Y. (Dec, 2001) A higher-order background model improves the detection of promoter regulatory elements by Gibbs sampling. *Bioinformatics,* **17**(12), 1113–1122.

[89] Liu, X., Brutlag, D. L., and Liu, J. S. (2001) BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. *Pac Symp Biocomput*, pp. 127–138.

[90] Wei, Z. and Jensen, S. T. (Jul, 2006) GAME: detecting cis-regulatory elements using a genetic algorithm. *Bioinformatics,* **22**(13), 1577–1584.

[91] Chan, T.-M., Leung, K.-S., and Lee, K.-H. (Feb, 2008) TFBS identification based on genetic algorithm with combined representations and adaptive post-processing. *Bioinformatics,* **24**(3), 341–349.

[92] Leung, H. C. M. and Chin, F. Y. L. (Sep, 2006) Finding motifs from all sequences with and without binding sites. *Bioinformatics,* **22**(18), 2217–2223.

[93] Sinha, S. (Jul, 2006) On counting position weight matrix matches in a sequence, with application to discriminative motif finding. *Bioinformatics,* **22**(14), e454–e463.

[94] Redhead, E. and Bailey, T. (Oct, 2007) Discriminative motif discovery in DNA and protein sequences using the DEME Algorithm. *BMC Bioinformatics*, **8**(1), 385.

[95] Larsson, E., Lindahl, P., and Mostad, P. (2007) HeliCis: a DNA motif discovery tool for colocalized motif pairs with periodic spacing. *BMC Bioinformatics,* **8**, 418.

[96] Li, N. and Tompa, M. (2006) Analysis of computational approaches for motif discovery. *Algorithms Mol Bio,* **1**, 8.

[97] Workman, C. T. and Stormo, G. D. (2000) ANN-Spec: a method for discovering transcription factor binding sites with improved specificity. *Pac Symp Biocomput,* pp. 467–478.

[98] Jensen, S. T., Shen, L., and Liu, J. S. (Oct, 2005) Combining phylogenetic motif discovery and motif clustering to predict co-regulated genes. *Bioinformatics,* **21**(20), 3832–3839.

[99] Jensen, S. T. and Liu, J. S. (Jul, 2004) BioOptimizer: a Bayesian scoring function approach to motif discovery. *Bioinformatics,* **20**(10), 1557–1564.

[100] Kankainen, M. and Löytynoja, A. (2007) MATLIGN: a motif clustering, comparison and matching tool. *BMC Bioinformatics,* **8**, 189.

[101] Milligan, G. W. and Cooper, M. C. (Jun, 1985) An examination of procedures for determining the number of clusters in a data set. *Psychometrika,* **50**(2), 159–179.

[102] Tompa, M., Li, N., Bailey, T. L., Church, G. M., Moor, B. D., Eskin, E., Favorov, A. V., Frith, M. C., Fu, Y., Kent, W. J., Makeev, V. J., Mironov, A. A., Noble, W. S., Pavesi, G., Pesole, G., Régnier, M., Simonis, N., Sinha, S., Thijs, G., vanHelden, J., Vandenbogaert, M., Weng, Z., Workman, C., Ye, C., and Zhu, Z. (Jan, 2005) Assessing computational tools for the discovery of transcription factor binding sites. *Nat Biotechnol,* **23**(1), 137–144.

[103] Schones, D., Smith, A., and Zhang, M. (Jan, 2007) Statistical significance of cis-regulatory modules. *BMC Bioinformatics,* **8**(1), 19.

[104] Fiedler, T. and Rehmsmeier, M. (2006) jPREdictor: a versatile tool for the prediction of cis-regulatory elements. *Nucleic Acids Res,* **34**, W546–W550.

[105] Metsker, S. J. (2002) Design patterns Java workbook, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[106] Baeza-Yates, R. and Gonnet, G. H. (1992) A new approach to text searching. *Commun. ACM,* **35**(10), 74–82.

[107] Knuth, D. E., James H. Morris, J., and Pratt, V. R. (1977) Fast Pattern Matching in Strings. *SIAM Journal on Computing,* **6**(2), 323–350.

[108] Bulyk, M. L., McGuire, A. M., Masuda, N., and Church, G. M. (2004) A Motif Co-Occurrence Approach for Genome-Wide Prediction of Transcription-Factor-Binding Sites in Escherichia coli. *Genome Res,* **14**, 201–208.

[109] Busturia, A., Lloyd, A., Bejarano, F., Zavortink, M., Xin, H., and Sakonju, S. (Jun, 2001) The MCP silencer of the Drosophila Abd-B gene requires both Pleiohomeotic and GAGA factor for the maintenance of repression. *Development,* **128**(11), 2163–2173.

[110] Déjardin, J. and Cavalli, G. (2004) Chromatin inheritance upon Zeste-mediated Brahma recruitment at a minimal cellular memory module. *EMBO J,* **23**(4), 857–868.

[111] Shimell, M. J., Peterson, A. J., Burr, J., Simon, J. A., and O'Connor, M. B. (Feb, 2000) Functional analysis of repressor binding sites in the iab-2 regulatory region of the abdominal-A homeotic gene. *Dev Biol,* **218**(1), 38–52.

[112] Kehle, J., Beuchle, D., Treuheit, S., Christen, B., Kennison, J. A., Bienz, M., and Müller, J. (Dec, 1998) dMi-2, a hunchback-interacting protein that functions in polycomb repression. *Science,* **282**(5395), 1897–1900.

[113] Qian, S., Capovilla, M., and Pirrotta, V. (Jun, 1991) The bx region enhancer, a distant cis-control element of the Drosophila Ubx gene and its regulation by hunchback and other segmentation genes. *EMBO J,* **10**(6), 1415–1425.

[114] Poux, S., Kostic, C., and Pirrotta, V. (Sep, 1996) Hunchback-independent silencing of late Ubx enhancers by a Polycomb Group Response Element. *EMBO J,* **15**(17), 4713–4722.

[115] Akaike, H. (May, 1981) Likelihood of a model and information criteria. *Journal of Econometrics,* **16**(1), 3–14.

[116] Frith, M. C., Hansen, U., Spouge, J. L., and Weng, Z. (2004) Finding functional sequence elements by multiple local alignment. *Nucleic Acids Res,* **32**(1), 189–200.

[117] Thijs, G., Marchal, K., Lescot, M., Rombauts, S., and Moor, B. D. (2001) A Gibbs Sampling method to detect over-represented motifs in the upstream regions of co-expressed genes. *RECOMB,* **5**, 305–312.

[118] Favorov, A. V., Gelfand, M. S., Gerasimova, A. V., Ravcheev, D. A., Mironov, A. A., and Makeev, V. J. (May, 2005) A Gibbs sampler for identification of symmetrically structured, spaced DNA motifs with improved estimation of the signal length. *Bioinformatics,* **21**(10), 2240–2245.

[119] Ao, W., Gaudet, J., Kent, W. J., Muttumu, S., and Mango, S. E. (Sep, 2004) Environmentally induced foregut remodeling by PHA-4/FoxA and DAF-12/NHR. *Science,* **305**(5691), 1743–1746.

[120] Eskin, E. and Pevzner, P. A. (2002) Finding composite regulatory patterns in DNA sequences. *Bioinformatics,* **18 Suppl 1**, S354–S363.

[121] vanHelden, J., André, B., and Collado-Vides, J. (Sep, 1998) Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *J Mol Biol,* **281**(5), 827–842.

[122] Regnier, M. and Denise, A. (2004) Rare events and conditional events on random strings. *Discrete Math Theor Comput Sci,* **6**(2), 191–214.

[123] Pavesi, G., Mauri, G., and Pesole, G. (2001) An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics,* **17 Suppl 1**, S207–S214.

[124] Pavesi, G., Mereghetti, P., Mauri, G., and Pesole, G. (Jul, 2004) Weeder Web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. *Nucleic Acids Res*, **32**(Web Server issue), W199–W203.

[125] Smale, S. T. and Kadonaga, J. T. (2003) The RNA polymerase II core promoter. *Annu Rev Biochem*, **72**, 449–479.