BIORED - A Genetic Algorithm for Pattern Detection in Biosequences

Pedro Pereira[1], Fernando Silva[1], and Nuno A. Fonseca[2]

[1] CRACS & Department of Computer Science, Faculty of Science, Universidade do Porto
Rua do Campo Alegre 1021/1055, 4169-007 Porto, Portugal, pdr,fds@dcc.fc.up.pt
[2] Instituto de Biologia Molecular e Celular (IBMC) & CRACS, Universidade do Porto
Rua do Campo Alegre 823, 4150-180 Porto, Portugal, nf@ibmc.up.pt

Summary. We present a new, efficient and scalable tool, named BIORED, for pattern discovery in proteomic and genomic sequences. It uses a genetic algorithm to find interesting patterns in the form of regular expressions, and a new efficient pattern matching procedure to count pattern occurrences. We studied the performance, scalability and usefulness of BIORED using several databases of biosequences. The results show that BIORED was successful in finding previously known patterns, thus an excellent indicator for its potential. BIORED is available for download under the GNU Public License at `http://www.dcc.fc.up.pt/biored/`. An online demo is available at the same address.

Keywords: motif discovery, genetic algorithm

1 Introduction

The identification of interesting patterns (or subsequences) in biosequences has an important role in computational biology. Databases of genomic and proteomic sequences have grown exponentially, and therefore pattern discovery still is a hard problem requiring clever algorithmic to achieve manageable levels of efficiency and powerful pattern languages to be useful.

Patterns often have an important biological significance, hence pattern discovery is an important problem in computational biology. It is, however, a computationally hard task, given the combinatorial involved. The rationale behind pattern discovery in biosequences (proteomic and genomic) is that the patterns correspond to subsequences preserved through evolution, and the reason for being preserved is because they are important to the function or structure of the molecule.

In this paper we describe BIORED, a new efficient and scalable tool to discover interesting patterns in genomic and proteomic sequences. It accepts a powerful pattern language that is a subset of regular expressions and uses a novel genetic algorithm to discover patterns together with a new efficient pattern matching procedure to count pattern occurrences in the sequences. We validate BIORED by applying it to several databases in order to try to rediscover previous known patterns and we study its sequential performance. BIORED is capable of efficiently finding patterns in very large sequence databases and be used to find considerably large patterns.

2 Background

The problem of pattern discovery here addressed can be stated as follows. Let Σ be an alphabet of residues (proteomic or genomic). Given a set of sequences S, each sequence composed with characters not restricted to the alphabet Σ, and a pattern size k, the goal is to find the best interesting pattern p with size k accordingly to some evaluation function.

We consider deterministic patterns with wild-cards and ambiguous characters. More specifically, the pattern language is a subset of regular expressions. Every position in the regular expression can be only composed by classes of characters belonging to Σ. A class is represented within brackets. The "`.`" (referred to as don't care character) is used to denote a class of characters composed by all elements in Σ. For compactness of representation, it is also possible to negate the class. In this case, all characters belonging to the alphabet and not shown in the class, are the ones that compose the class. The negation is denoted by "`^`". For instance, the pattern with length 3 "`[GT].A`" has two matches in the sequence $AA\underline{TAA}G\underline{TTA}A$.

The chosen pattern language is a compromise between simplicity and power. The idea is to allow the discovery of complex patterns while having a sufficiently fast matching algorithm. Although interesting patterns may have gaps, which may be the result of deletions or insertions, many others have undergone smaller mutations and have an equal length. The principle is that we can usually find sub-patterns of larger patterns and later extend them. Another advantage of using (a subset) of regular expressions is that the resulting language is well supported by a considerable number of programs (e.g., grep, sed, emacs, etc) and programming languages (e.g., Perl, PHP, etc).

3 A Genetic Algorithm for Pattern Discovery in Biosequences

BIORED uses a genetic algorithm (GA) [1] to perform pattern discovery. It receives as input a database containing a set of sequences, the length of a pattern k, and some other parameters (such as the maximum number of generations i), and tries to find an interesting pattern of length k.

The implementation of a GA requires the prior definition of a (1) a genetic representation of a pattern (solution), and (2) a fitness function to evaluate the patterns. The implementation of the fitness function involves counting the number of matches of a pattern in the input sequences. This can be a limiting performance factor for the algorithm, therefore we devised an efficient matching procedure.

We next describe the genetic operators used, the fitness function (interestingness metric) and a sequential algorithm for counting the matches.

3.1 Genetic Operators

A genetic operator [1] is a process that aims at maintaining genetic diversity. The operators are analogous to those that occur in the natural world: survival of the fittest, or selection; sexual or asexual reproduction, or crossover; and mutation.

BIORED implements a rank selection operator that sorts the individuals in the population by comparing their fitness value. Each individual is then given a probability of being chosen for reproduction depending on its position. For n individuals, a $n(n+1)/2$ slots roulette-wheel is constructed, with the fittest individual receiving n slots, the second fittest $n-1$, and so on, with the least fit individual receiving just one slot.

During the alternation (or reproduction) phase of the GA, we use three classical genetic operators: mutation, crossover and elitism. The crossover operator selects a character position in the individual to be generated. It then sets the first part with the contents of the first individual and the second part with the contents of the second individual (both selected using a rank selection operator). The mutation operator randomly flips some of the bits that compose the chromosome. The elitism operator selects some of the best individuals to be copied verbatim to the next generation, without suffering any mutation.

3.2 Interestingness based on statistical significance

To guide the search for a pattern and for ranking a set of patterns one needs some measure to assess, in some way, their quality or interestingness. In a GA context, such measure is called fitness function. In complex problems, such as pattern discovery, GAs have a tendency to converge towards local optima rather than the global optimum of the problem. This problem may be alleviated by using a different fitness function, or by using techniques to maintain a diverse population of solutions. Therefore, two fitness functions were considered based on statistical interestingness.

Several approaches have been proposed to determine if a pattern is statistically interesting [2, 3, 4], i.e., if the number of occurrences of a pattern in a set of sequences is greater than the expected value. A pattern is considered statistically interesting if it is overrepresented in the sequences where it occurs. To measure the over-representation, we need to consider the expected number of occurrences and the standard deviation of this value. Equivalently, we need to know how the values are distributed.

We assumed that the probability of the symbols (from Σ) to appear in S are independent and identically distributed. Under these assumptions, the word probability follows a Binomial distribution. The Binomial distribution gives the discrete probability $b(x;\ n, p)$ of obtaining exactly x successes (matches) out of n Bernoulli trials (pattern positions). We consider every character position, that can be a possible place for the word occurrence, as a Bernoulli trial. For example, if we have the sequence `ACGATCAGTACA` and the pattern that we are computing the statistics for has length 5 then there are exactly 8 places where the pattern can occur. Generalizing, having a sequence and a word of length S_n and W_n respectively, there are $S_n - W_n + 1$ places where the word can appear if $S_n \geq W_n$ or zero otherwise. Each Bernoulli trial is true with probability p. The probability p is the multiplication of the probabilities of the individual pattern positions. In turn, the pattern positions probabilities is the sum of the probabilities of the symbols that compose the position. For efficiency reasons, the binomial distribution is approximated by the Poisson distribution for large values of n and small values of p, with $\lambda = np$, or equivalently $p(x;\ \lambda) \approx b(x;\ n, \lambda/n)$ [5].

We are interested to know if the pattern is overrepresented, therefore we calculate the probability of the pattern to appear at least the same number of times in a database as it effectively appears. Equivalently, we compute the complementary cumulative distribution function (F_c) of the Poisson distribution for $x - 1$: $Z = F_c(x-1) = P(X > x-1)$. Since, the Z can take very small values we use the negative logarithmic of Z, more specifically, $-\log(Z)$. We next denote $-\log(Z)$ as $\mathcal{I}$.

The first fitness function relates the interestingness of the pattern with its complexity,

$$f_1 = \frac{\mathcal{I}}{\text{complexity}^x}$$

for $x = 0, 1, 2, 3$. The *complexity* is the sum of the number of characters recognized by each pattern position. For instance, `ACGT` has complexity 4, while `[AC]CGT` has complexity 5 and `[AC][CG][GT][TA]` has complexity 8. The parameter x is used to reduce the patterns complexity, thus improving their quality. Generally, the low quality patterns are a direct result of being too general.

The second fitness function (f_2) borrows ideas from the evaluation function F-measure,

$$f_2 = \frac{2 \times \text{logpn} \times \text{cpx}}{\text{logpn} + \text{cpx}}$$

where p is the probability of the pattern, m is the maximum complexity with same length and alphabet can have, $\text{cpx} = 1 - \frac{\text{complexity}}{m}$, $\text{logpn} = \mathcal{I}/10000$. A ceiling of 10000 is assumed to the value of $\mathcal{I}$.

In general, it is not possible to determine which fitness function behaves better in a set of sequences without some kind of experimentation. This experimentation needs only to be done once for each sequence, and can be done automatically by executing the programs for all the possible fitness functions and choosing the one that achieves the best results.

3.3 Counting Matches

The fitness function, or interestingness metric, requires knowing the number of (overlapping) occurrences of every pattern in the sequence. For example, in the sequence `AAATAA`, the pattern `AA` occurs three times and the pattern `AA[AT]` occurs twice.

Counting the number of occurrences of a single pattern can be troublesome. For instance, if the sequences have total length of n and the pattern is composed by either symbols or unit-length don't care characters with length m, the best algorithm runs in $O(n \log m)$ time (worst-case) [6]. If we could come up with an algorithm with an equal complexity for the worst-case, the best we could do would be $O(ni \log m)$, where i is the number of different patterns (number of individuals of the population). However, since unit-length don't care characters are a subset of classes of characters, the chosen pattern language is more powerful than the pattern language referred in [6].

Since the GA generates several individuals (patterns) in each generation that need to be evaluated, we tried to devise an efficient method to evaluate them simultaneously.

If the algorithm could only handle a single pattern, then it is possible to use a linear solution based on bit-parallelism [7] if the pattern length is small (only a few machine words are needed). The bits are used to simulate a non-deterministic finite automaton (NFA) that describes the pattern.

To expand the algorithm to evaluate several patterns at once, a window with length k is moved through the sequence. Note that all patterns have the same length k. For each window position every pattern is checked for a match. In a sequence with size n, the number of window positions (window size is k) is $n - k + 1$ (assuming that $n \geq k$).

The counting matches algorithm worst-case complexity is $O(nik)$ with the input size n, i the number of individuals in the population, and k the length of the patterns. However, the algorithm is on average much faster, achieving a complexity of $O(ni/w)$, where w is the number of bits in a machine word. The average complexity is directly linked to the average case of the naive string matching.

In spite of the effort to have an efficient counting operation, it remains the bottleneck of the matching algorithm. A parallel version of BIORED was thus developed that achieved linear speedup up to 22 processors [8].

3.4 Implementation

The BIORED was implemented using the C language because the speed was crucial and to perform an extreme control on memory usage. For the statistic functions we used the R [9] library. Note that BIORED can be executed in a variety of platforms, such as clusters and in GRIDs.

The alphabet letters (representing nucleotides or residues) are implemented using an unsigned integer with 32 bits. This representation has the advantage of being simple to apply the genetic operators, namely the crossover and the mutation. This means that a population with i individuals, each having length k, uses exactly $4ik$ bytes of memory using the DNA alphabet. In general, the algorithm uses $|\Sigma|ik/8$ bytes of memory, where Σ is the alphabet used.

We use a binary vector as a chromosome to represent a pattern. The binary vector can, conceptually, be seen as signaling if a character belonging to Σ is present or not at a determinate pattern position. For example, the DNA pattern `[AC]T[ACGT]G` is represented as `1100,0001,1111,0010`, if `A` is represented with the bit-mask `1000`, `C` `0100`, `G` `0010` and `T` `0001`.

The initial population (set of patterns) is randomly initialized. Each bit in an individual has the probability 0.7 of being activated (this value was selected after performing several experiments). The probability was empirically chosen to guarantee the diversity of the population, representing patterns that actually occur in the data.

The probability of undergoing crossover was set to 0.75 and the mutation probability to 0.01. Only the fittest individual is considered an elite. These values were chosen after some experiments with DNA and residue sequences and are the values that proved to work better. By default, the program halts after completing 500 generations. This value was chosen based on the performance experiments done.

Finally, it is worth mention that BIORED includes an option setting to allow the use of symbol probabilities (distribution) different from those observed in the sequences. This requires the user to give an extra file (with sequences) to the program from where the distribution is computed. An example of the usefulness of this option is demonstrated in Section 5.

4 Performance Evaluation

We study the performance of BIORED and the behavior of the GA in terms of convergence and execution time. The databases used in the experiments are indicated in Table 1 and were obtained from the release 38 of the Ensembl project [10]. All experiments were ran in a Cluster with Dual core "AMD Opteron Processor 250" computers, with 4 gigabytes of RAM (but only 600 Mb free) running the Linux operating system (kernel 2.6).

Organism	Length (bp)
Saccharomyces cerevisiae (whole genome)	12156606
Anopheles gambiae (chromosome 2R)	61545105
Drosophila melanogaster (whole genome)	144141726

Table 1. Organisms used for evaluation.

Figure 1 shows the effect on the runtime when we alter a single parameter, such as the population size or the pattern length. Theoretically, the runtime is expected to double when the population size is doubled. However, the optimizations performed in the algorithm makes the runtime vary.

The three organisms used (see Table 1) can be processed in about (largest to smallest) 27, 10 and 4 hours, running for one-thousand generations with a population size of 128 individuals and searching for patterns with length of 64. These values for running times are, in our view, excellent for a sequential execution, considering the relative large size of the data used. In fact, other pattern discovery tools failed to cope with the same data, thus making it impossible to compare relative efficiency (details are discussed in Section 6).

When the pattern length is increased something apparently strange happens. Until a certain pattern length the runtime increases and then it decreases. This is, once

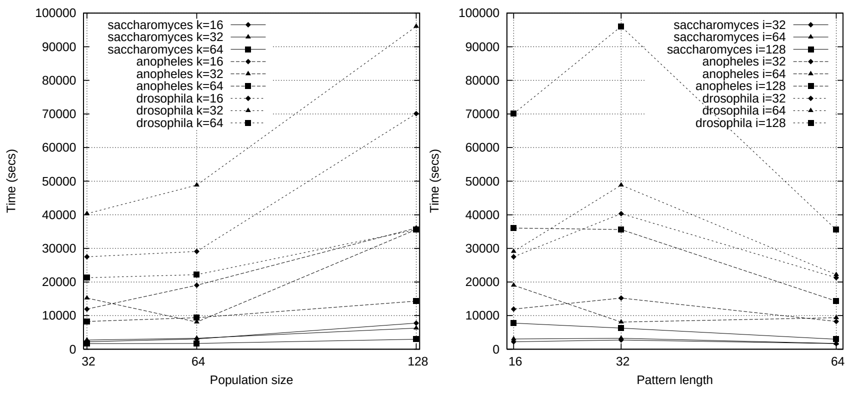

Fig. 1. Run time variation with different populations and pattern lengths (in seconds).

again, related to the size of the search space. When the search space grows too much, the genetic algorithm has difficulty in finding an admissible pattern. A possible solution to this problem could be to initialize the population with statistically interesting words (naturally, found with another tool).

The results show a very small variation on the runtimes when the population sizes increase from 32 to 64 individuals. This effect is a direct consequence of the implemented bit-parallelism technique and the execution on 64-bit architectures.

We evaluated BIORED's convergence and discovered that when the pattern length increases, the population size must also be increased for the convergence to be smoother. This happens because it is more difficult to obtain an admissible large pattern. This was expected, since the search space of a DNA pattern with length 64 is $2^{4\times 64}$. Furthermore, as the size of the pattern decreases, the faster the algorithm converges. This was also expected since the search space becomes exponentially smaller as the size of the pattern decreases.

5 Validation

We demonstrate the usefulness of BIORED in two case studies. The goal is to rediscover some already known patterns.

Human Gene for Proinsulin

In the first case study we chose a database with the human gene for proinsulin from chromosome 11 [11]. BIORED was configured to run with a population of 32 individuals, pattern length of 14, and to stop after one-thousand generations. It yielded the pattern `[CG][AT]GGGG[AT][CG][AT]GGGG[AT]` with a score of 381.6, occurring 48 times and with a probability of 0.00000133. The pattern found is very similar to a previously reported pattern `ACAGGGGTGTGGGG` [12].

Drosophila Melanogaster

In the second case study, we used a database with the whole genome of the Drosophila melanogaster. More concretly, we used the organism disjoint introns (sections of DNA that are spliced out after transcription but before the RNA is used) as input to BIORED, and configured it to use a population of 64 individuals, and a pattern length of 27. The symbols probabilities were gathered from the whole genome. The best pattern after 4096 generations was `ATTGTAAGTCTTTAAATATATTCGTGT` with a score of 7309.4, occurs 256 times and has a probability smaller than 10^{-9}. Curiously, this pattern is a sub-word of the consensus described in [13]. The consensus was manually converted to a regular expression producing (after some simplification) the following pattern: `[^G][AG]AGTT[CT]GT[^A][GT]C[CT]T[AG]AGTCTTT[CT]GTTT`. Note that the original pattern, as it is, achieves a score of 904.2 on the entire genome, i.e., the entire genome was used to compute the distribution of the symbols, while the pattern found by BIORED achieves a score of 7309.4

6 Related Work

Several pattern discovery tools and algorithms have been developed [14, 15, 4]. Some approaches are based on exhaustive search that guarantee that the best pattern (accordingly to some specifications) is found. An heuristic approach does not guarantee that the best pattern is found, instead it finds a good "enough" pattern. The advantage of the heuristic approach is that it is often faster than the exhaustive search, but may not find the best solution (pattern).

The Teiresias [14] is closely related to our proposal in terms of the pattern language. It is based on well-organized exhaustive search based on combinations of shorter patterns. The Teiresias algorithm guarantees that all maximal [14] patterns are reported. The algorithm needs to receive as input the minimum number of literals that a pattern can contain, L. Another required parameter is W that indicates the maximum distance between any consecutive L literals. In general, if we use the same L parameter and increase the W parameter, the execution time of the scanning phase, which is the phase where the algorithm gathers seed patterns with the desired L and W characteristics, greatly increases. In the worst case the algorithm is $O(n^3 log\ n)$, but it is reported to work very well when the inputs are highly regular and the parameters W and L are small.

The admissible patterns are similar to the ones we consider. The original Teiresias algorithm only supported one wild card equivalent to our "`.`". Newer versions support equivalency classes. In an equivalency class the user needs to specify the characters that are to be treated as equal in the actual pattern discovery process. These are similar to the classes of characters supported by BIORED.

A critical problem with Teiresias is that it has a very high memory usage. In an attempt to compare the performance of Teiresias with BIORED we configured Teiresias to support the same classes of characters as BIORED, and tried to identify the previously discovered pattern in the human gene of proinsulin. Teiresias crashed after 8 minutes with a memory consumption of several gigabytes. These parameters were chosen to verify if it could identify the previously discovered pattern in the human gene of proinsulin using the BIORED.

In conclusion, Teiresias seems to be unable to cope with classes that overlap each other (which is exactly the classes supported by BIORED) since it has an extremely high memory consumption that prevents any empirical comparison since it crashes even for small sequences.

Pratt [15] is another tool to discover patterns conserved in sets of unaligned protein sequences. The patterns that can be found are a subset of the patterns that can be described using Prosite notation [16]. In particular, variable length gaps are allowed. Pratt is very memory intensive, contrasting to BIORED, which is pretty light in memory consumption. Pratt tries to find a pattern that occurs in the greatest number of sequences as possible, while the program presented here considers the total number of occurrences in all sequences.

MEME (Multiple EM for Motif Elicitation) [17] uses a stochastic search to discover patterns. It does not require a pattern length parameter, which can be estimated by the algorithm itself. The algorithm is based on expectation maximization technique. Individual MEME patterns cannot contain gaps, and thus are equivalent to the patterns we consider. The overall complexity of MEME is quadratic in the size of the database and linear in the length of the pattern [17], while our proposal is linear in the size of the database and in the length of the pattern.

7 Conclusion

We presented a new pattern discovery tool that discovers interesting patterns, in the form of a regular expression. using a genetic algorithm. The algorithm has a conservative memory usage of $O(ik|\Sigma|)$ and a worst-case time complexity of $O(nikg)$, where Σ is the alphabet used, i is the number of individuals of the population, k is the length of the pattern, n is the size of the input, and g is the number of generations. However, the algorithm is on average much faster, achieving a complexity of $O(gni/w)$, where w is the number of bits in a machine word. The average complexity is directly linked to the average case of the naive string matching. Experiments showed the usefulness of the algorithm, by demonstrating that it is capable of discovering previously known patterns.

The contributions of this paper are three-fold. First, we describe and evaluate a tool that uses a genetic algorithm to discover patterns in genomic and proteomic sequences. Second, we also propose an efficient pattern matching procedure, a crucial component for achieving high performance in any pattern discovery tool. Finally, for a practitioner we provide a pattern discovery tool that is efficient (in terms of execution time and memory usage), has a powerful pattern language, does not impose restrictions on the pattern length, is general (can handle proteomic and genomic sequences), and can handle large databases of sequences, and can be used in a with number of settings (personal computers, clusters, and grids).

Finally, there is still space for improvement. For instance, BIORED implements a simple and fast statistical approach to determine the interestigness of a pattern. In order to improve the statistical accuracy, we plan to include, as an option, more rigorous tests such us the recently proposed complementary statistical tests for accessing exceptionalities of motif counts [18].

Acknowledgements

This work has been partially supported by projects ILP-Web-Service (PTDC/EIA/70841/-2006), JEDI (PTDC/EIA/66924/2006), STAMPA (PTDC/EIA/67738/2006) and by Fundação para a Ciência e Tecnologia. Nuno Fonseca is funded by the FCT grant SFRH/BPD/26737/2006 and Pedro Pereira by the FCT grant SFRH/BD/30628/2006.

References

1. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (Third, Revised and Extended Edition)*. Springer-Verlag, New York, NY, USA, 1999.
2. J. van Helden, M. del Olmo, and J. Perez-Ortin. Statistical analisys of yeast genome downstream sequences reveals putative polyadenylation signals. *Nucleic Acids Research*, 28(4):1000–1010, 2000.
3. S. Sinha and M. Tompa. A statistical method for finding transcription factor binding sites. *Proceedings of the National Academy of Sciences of the United States of America*, 95(6):2738–2743, 2000.
4. S. Sinha and M. Tompa. An exact method for finding shor motifs in sequences, with application to the ribosome binding site problem. *Proceedings of the 7th International Conference on ISMB*, pages 262–271, 1999.
5. Willian Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, third edition, 1968.
6. Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 592–601, New York, NY, USA, 2002. ACM Press.
7. Gonzalo Navarro. Pattern matching. *Journal of Applied Statistics*, 31(8):925–949, 2004. Special issue on Pattern Discovery.
8. Pedro Pereira, Nuno A. Fonseca, and Fernando Silva. A high performance distributed tool for mining patterns in biological sequences. Technical Report DCC-2006-08, DCC-FC & LIACC, Universidade do Porto, 2006.
9. R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0.
10. T. Hubbard, D. Andrews, and M. Caccamo et. al. Ensembl 2005. *Nucleic Acids Research*, 33(1), January 2005.
11. Rotwein P., Yokoyama S., Didier D. K., and Chirgwin J. M. Genetic analysis of the hypervariable region flanking the human insulin gene. *The American Journal of Human Genetics*, 1986.
12. Amy Lew, William J. Rutter, and Giulia C. Kennedy. Unusual dna structure of the diabetes susceptibility locus iddm2 and its effect on transcription by the insulin promoter factor pur-1/maz. *Proceedings of the National Academy of Sciences of the United States of America*, 97(23):12508–12512, November 2000.
13. Javier Costas, Cristina P. Vieira, Fernando Casares, and Jorge Vieira. Genomic characterization of a repetitive motif strongly associated with developmental genes in drosophila. *BMC Genomics*, 2003.
14. Isidore Rigoutsos and Aris Floratos. Combinatorial pattern discovery in biological sequences: The teiresias algorithm. *Bioinformatics*, 14(1):55–67, 1998.
15. Inge Jonassen, John F. Collins, and Desmond Higgins. Finding flexible patterns in unaligned protein sequences. *Protein Science*, 4(8):1587–1595, 1995.
16. Hulo N., Bairoch A., Bulliard V., Cerutti L., De Castro E., Langendijk-Genevaux P.S., Pagni M., and Sigrist C.J.A. The prosite database. *Nucleic Acids Res.*, 34, 2006.
17. Timothy L. Bailey and Charles Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second International Conference on ISMB*, pages 28–36. AAAI Press, 1994.
18. Stphane Robin, Sophie Schbath, and Vincent Vandewalle. Statistical tests to compare motif count exceptionalities. *BMC Bioinformatics*, 8(84), 2007.