Pattern Discovery: Methods and Software

Broňa Brejová[1], Tomáš Vinař[1], Ming Li[1,2]

[1] Department of Computer Science, University of Waterloo, ON N2L 3G1, Canada
[2] Department of Computer Science, University of California Santa Barbara, CA 93106, U.S.A.

Contents

1 Introduction **2**

2 Pattern Discovery **3**
2.1 What is a pattern . . . 3
2.2 Pattern discovery vs. pattern matching . . . 5
2.3 Goals of pattern discovery . . . 5

3 Algorithms for Pattern Discovery **6**
3.1 Exhaustive search . . . 6
3.1.1 Enumerating all patterns . . . 6
3.1.2 Exhaustive search on graphs . . . 8
3.2 Creating long patterns from short patterns . . . 9
3.2.1 TEIRESIAS algorithm . . . 9
3.2.2 Improvement of running time . . . 11
3.3 Iterative heuristic methods . . . 11
3.3.1 Gibbs sampling . . . 11
3.3.2 Other iterative methods . . . 12
3.3.3 From iteration to PTAS . . . 13
3.4 Machine learning methods . . . 14
3.4.1 Expectation maximization . . . 14
3.4.2 Hidden Markov models . . . 15
3.4.3 Enhancing HMM models . . . 16
3.5 Methods using additional information . . . 17
3.5.1 Identifying motifs in aligned sequences . . . 17
3.5.2 Global properties of a sequence . . . 17
3.5.3 Using phylogenetic tree . . . 18
3.5.4 Use of secondary/tertiary structure . . . 18
3.6 Finding homologies between two sequences . . . 19

4 Assessment of Pattern Quality **20**
4.1 Background model . . . 21
4.2 Pattern significance . . . 22
4.3 Information content . . . 22
4.4 Sensitivity and specificity of classification . . . 23

5 Concluding Remarks **23**

References **24**

1 Introduction

A pattern is a feature that occurs repeatedly in biological sequences, typically more often than expected at random. Patterns often correspond to functionally or structurally important elements in proteins and DNA sequences. Pattern discovery is one of the fundamental problems in bioinformatics. It has applications in multiple sequence alignment, protein structure and function prediction, drug target discovery, characterization of protein families, and promoter signal detection.

Regions important to the structure or function of the molecule tend to evolve more slowly. In particular an occurrence of a conserved motif in a protein may imply that the region in question may be involved in the interaction with some other protein, may comprise the active site of an enzyme or may be important for the tertiary structure of the protein. Attempts have been made to organize proteins and protein domains from different organisms to families based on their evolutionary relations, structural and functional similarity [Linial et al., 1997]. Sequences in one family often share one or several common motifs and these motifs are used to characterize the family. Several databases containing motifs characterizing protein families have been established. Newly discovered proteins can be assigned to a family by searching such a database of motifs. We may then associate the function or structure to the new protein based on the knowledge we have about the other members of the family.

Nucleotide sequences outside of protein coding regions in general tend to be less conserved, except where they are important for function, that is, where they are involved in the regulation of gene expression. Regulatory elements are mostly located in promoter regions upstream from genes. Identifying promoters in genomic sequences is difficult, especially in eukaryotic genomes because they do not have a common core promoter but rather consist of multiple regulatory factors distributed over long distances. Adding to the complexity is low number of available annotated promoters. The best of programs tested in a review [Fickett and Hatzigeorgiou, 1997] were able to identify only about half of eukaryotic promoters.

If a pattern characteristic for a binding site of a certain transcription factor is known, we can find occurrences of this pattern in promoter regions of known genes. This helps us to understand how these genes are regulated, under which conditions they are transcribed and it may even help to infer a function of a gene [Yada et al., 1997].

New binding motifs of transcription factors can be discovered by considering upstream regions of co-regulated genes and identifying motifs that occurs in these regions more frequently than elsewhere [Mironov et al., 1999, Hughes et al., 2000]. For example groups of co-regulated genes can be identified by analyzing gene expression data [Chiang et al., 2001]. Provided that a transcription factor is conserved between two species we may discover its binding sites by identifying conserved sequences in promoter regions of these two genomes [Gelfand et al., 2000].

Related molecules, usually proteins or RNA, sometimes do not display significant similarity at the sequence level. However, significant similarity can be found in their secondary or tertiary structure. Discovery of structural motifs in proteins and RNA molecules has also been studied [Eidhammer et al., 2000], but this work is not in the scope of this chapter.

The availability of several fully sequenced genomes has enabled scientists to identify homologies shared between two genomes. Such conserved regions are likely to correspond to functionally important elements – this information was applied to predict genes [Batzoglou et al., 2000], discover new regulatory elements [Hardison et al., 1997] and reveal evolutionary relationships between species and types of evolutionary changes to genome organization [Riechmann et al., 2000]. Identifying possibly long homologies between two long sequences can be considered a special case of pattern discovery, yet the large amounts of data require special consideration from the computer science point of view.

In this chapter we will introduce basic ideas of algorithms used to discover patterns. We also discuss the goals of such algorithms and how to statistically verify their results. Many programs for pattern discovery and databases of biologically relevant patterns are available. We provide an overview and links to these important tools in a supplement located in the accompanying CD-ROM.

2 Pattern Discovery

2.1 What is a pattern

A pattern is an element that has multiple occurrences in a given set of biological sequences. In this section we outline how to represent a pattern. Probably the simplest representation of a pattern is a list of its occurrences in the given sequences.

Although the list of occurrences is sufficient to specify the pattern, it is not convenient for further use. For example it is difficult to decide for a new sequence whether or not it contains occurrences of the pattern. Therefore we usually represent a pattern by describing properties shared by all its occurrences. Such representation is more succinct and allows easier searches for new occurrences.

Deterministic patterns. The simplest kind of a pattern is a *consensus sequence*. For example `TATAAAA` is the TATA box consensus sequence. Whenever we find string `TATAAAA` we say it is an occurrence of this pattern. Of course not all such occurrences correspond to a real TATA box, and not every TATA box perfectly matches the consensus. The latter of these two problems can be solved by allowing a certain degree of flexibility in the pattern. This can be achieved by adding some of the following frequently used features.

Let Σ be the alphabet of all possible characters occurring in the sequences (i.e. $\Sigma = \{A, C, G, T\}$ for DNA sequences and Σ is a set of all 20 amino acids for protein sequences).

- **Ambiguous character** is a character corresponding to a subset of Σ. An ambiguous character matches any character from this subset. Such a subset is denoted by a list of its elements enclosed in square brackets, e.g., `[LF]` is a set containing L and F. `A-[LF]-G` is a pattern in a notation used in PROSITE database. This pattern matches 3-character subsequences starting with A, ending with G and having either L or F in the middle.

  For nucleotide sequences there is a special letter for each set of nucleotides, where `R=[AG]`, `Y=[CT]`, `W=[AT]`, `S=[GC]`, `M=[AC]`, `K=[GT]`, `B=[CGT]`, `D=[AGT]`, `H=[ACT]`, `V=[ACG]`, `N=[ACGT]`.

- **Wild-card or don't care** is a special kind of ambiguous character that matches any character from Σ. Wild-cards are denoted N in nucleotide sequences, X in protein sequences. They may also be denoted by a dot '.'. A group of one or several consecutive wild-cards is called a *gap* and patterns allowing wild-cards are often called gapped patterns.

- **Flexible gap** is a gap of variable length. In the PROSITE database it is denoted by `x(i,j)` where i is the lower bound on the gap length and j is an upper bound. Thus `x(4,6)` matches any gap with length 4, 5, or 6. Fixed gap of length i is denoted `x(i)` (e.g. `x(3)` = `xxx`). Finally `*` denotes a gap of any length (possibly 0).

String `F-x(5)-[LF]-x(2,4)-G-*-H` is an example of a PROSITE pattern containing all of the above features.

Patterns with mismatches. One can further extend the expressive power of deterministic patterns by allowing a certain number of mismatches. The most commonly used type of mismatches are substitutions. In this case subsequence S matches pattern P with at most k mismatches, if there is a sequence S' exactly matching P that differs from S in at most k positions.

Sometimes we may also allow insertions or deletions, i.e., the number of mismatches would be an edit distance between the substring S and a closest string matching the pattern P.

Position weight matrices. So far we have explored only deterministic patterns. A deterministic pattern either matches the given string or not. However, even the most complicated deterministic patterns cannot capture some subtle information hidden in a pattern. Let us assume we have a pattern that contains on the first position C in 40% cases and G in 60% cases. The ambiguous symbol `[CG]` gives the same importance

PWM with relative frequencies

A	0.26	0.22	0.00	0.00	0.43	1.00	0.11
C	0.17	0.18	0.59	0.00	0.26	0.00	0.35
G	0.09	0.15	0.00	0.00	0.30	0.00	0.00
T	0.48	0.45	0.41	1.00	0.00	0.00	0.54

PWM with log-odd scores (using $f(c) = \frac{1}{4}$)

A	-3.94	-4.18	$-\infty$	$-\infty$	-3.22	-2.00	-5.18
C	-4.56	-4.47	-2.76	$-\infty$	-3.94	$-\infty$	-3.51
G	-5.47	-4.74	$-\infty$	$-\infty$	-3.74	$-\infty$	$-\infty$
T	-3.06	-3.15	-3.29	-2.00	$-\infty$	$-\infty$	-2.89

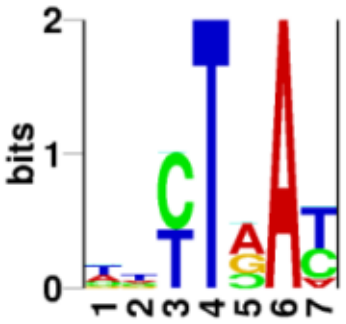

Figure 1: Position weight matrix of vertebrate branch point in the form of a table and corresponding visual representation as a sequence logo. The sequence logo was created using RNA Structure Logo an on-line tool at `http://www.cbs.dtu.dk/gorodkin/appl/slogo.html`

to both nucleotides. It does not matter in strong patterns, but it may be important in weak patterns, where we need to use every piece of information to distinguish the pattern from a random sequence.

Probabilistic patterns are probabilistic models that assign a probability to each sequence that it was generated by the model. The higher the probability, the better the match between the sequence and the pattern.

The simplest type of probabilistic pattern is a position-weight matrix (PWM). PWMs are also sometimes called a position-specific score matrix (PSSM), or a profile (however, term 'profile' is also used for more complicated patterns allowing gaps). PWM is a simple ungapped pattern specified by a table. This table shows the relative frequency of each character at each position of the pattern (see Figure 1 for an example).

Assume that the pattern (i.e. PWM) has length k (number of columns of the table). The score of a sequence segment $x_1 \dots x_k$ of length k is

$$\prod_{i=1}^{k} \frac{A[x_i, i]}{f(x_i)}$$

where $A[c, i]$ is an entry of position weight matrix corresponding to position i of the pattern and character c, and $f(c)$ is the background frequency of character c in all sequences considered. This product represents the odd-score that the sequence segment $x_1 \dots x_k$ belongs to the probability distribution represented by the PWM [Dorohonceanu and Nevill-Manning, 2000]. In order to simplify the computation of the score we can store log-odd scores $A'[c, i] = \log_2 A[c, i]/f(c)$ in a table, in place of using plain frequencies $A[c, i]$. Then the following formula gives us log-odd score instead of odd score:

$$\sum_{i=1}^{k} A'[x_i, i].$$

Position-weight matrices can be visualized in the form of sequence logos [Schneider and Stephens, 1990] (see Figure 1). Each column of a sequence logo corresponds to one position of the pattern. Relative heights of the characters in one column are proportional to the frequencies $A[c, i]$ at the corresponding position of the pattern. The characters are ordered according to their frequency, with the most frequent character on top. Each column is scaled so that its total height is proportional to the information content of the position, computed as

$$\log_2 |\Sigma| + \sum_{c} A[c, i] \log_2 A[c, i].$$

Value $\log_2 |\Sigma|$ is added in order to obtain positive values. It depends on the size of alphabet Σ. Sequence logos have been developed further to consider background distribution, and to invert characters that occur less frequently than expected [Gorodkin et al., 1997a].

An examination of a sequence logo reveals the most conserved positions (highest columns) and consensus characters at all positions (highest letter in the column). Notice that the size of characters in different columns cannot be directly compared.

Stochastic models. All types of patterns discussed thus far are explicit in a sense that the user can easily see important characteristics of the occurrences of a pattern. Sometimes it is advantageous to represent a pattern in a more implicit form, usually as a discrimination rule, which decides whether a given sequence is an occurrence of the modeled pattern or not. It can be based on a stochastic model, such as hidden Markov model (HMM), or can employ machine learning methods, such as neural networks.

It is questionable whether such rules constitute a pattern at all. Obviously, they can be trained (which corresponds to pattern discovery) and then they can be used for discrimination (which corresponds to pattern matching). Therefore, they are applicable in pattern-related tasks such as protein family classification and binding site discovery. In some cases (such as HMMs with simple topology) it is even possible to obtain some information about the pattern modeled, for example relative frequencies of characters at individual conserved positions.

2.2 Pattern discovery vs. pattern matching

There are two fundamentally different tasks related to identifying new patterns in biological sequences. The first one is called *pattern matching*. This involves finding new occurrences of a known pattern. Many consensus sequences are known in biology and it is important to have tools that will allow one to find occurrences of known patterns in new sequences.

There are specialized software tools for pattern matching. Some are quite general, i.e., they allow the user to specify a pattern as part of the input in a specific form. Others are built to recognize only one pattern. Many specialized tools are available for recognizing splicing signals, different regulatory elements, and special structural elements. Authors of such specialized tools fine-tune the parameters of the system to increase the accuracy of the prediction.

Although pattern matching is very important, we will concentrate on a different kind of pattern-related problem, called *pattern discovery*. The task is to identify a new pattern in a set of several sequences.

The input for pattern discovery programs consists of several sequences, expected to contain the pattern. Input sequences are typically related in some way, e.g., they are members of the same protein family, functionally related sequences, or upstream regions of co-regulated genes.

2.3 Goals of pattern discovery

The goal of pattern discovery is to identify an unknown pattern in a given set of sequences. There are a great number of potential patterns and it is often difficult to decide which of them are the most promising. Defining the "best" pattern depends on the intended use of the pattern. We will consider two possible scenarios: classification, i.e., we want to characterize members of some sequence family and distinguish them from non-members, and identification of significant patterns, i.e., we want to discover patterns that are unlikely to occur by chance and would therefore probably have functional or structural significance.

Classification. Formalization of pattern discovery as a classification problem has been reviewed in [Brazma et al., 1998]. In a classification scenario we want to identify motifs that best characterize a given protein family. The motifs thus identified are then used as classifiers. For example, given an unknown protein we can classify it as a member or non-member of a family, based on whether it contains the motifs characteristic for that family. This is a typical machine learning problem: given a set of sequences belonging to the family (positive examples) and a set of sequences not belonging to the family (negative examples), find a function f which decides for each protein whether it belongs to the family or not. In context of motif discovery we consider classes of functions f which involve matching some discovered patterns against the unknown sequence. Note, that negative examples are simply other known proteins taken from protein databases such as SWISS-PROT.

The common strategy in pattern discovery is to use only positive examples. The most significant motif in the family is found in a hope that it will not occur elsewhere. Negative examples are used only for evaluation of the prediction. Thus, the task is converted to the second scenario, described below.

Identifying significant patterns. Motif discovery is not always formulated as a classification problem. For example if we want to identify a regulatory element, we have a set of regions likely to contain this element. However, it does not mean that this element cannot occur in other places in the genome or that all of these sequences must contain a common regulatory element. Also in a context of protein family motifs we are interested in identifying conserved regions that may indicate structurally or functionally important elements, regardless whether they have enough specificity to distinguish this family from other families. In this context it is more complicated to precisely formulate the question.

The usual approach is to find the *highest scoring* pattern within a well-defined class of patterns (e.g. PROSITE patterns as they were defined in Section 2.1) that has sufficient *support*. Various approaches use different scoring functions and support measures and consider different classes of patterns.

Support of a pattern is usually the number of sequences in which the pattern occurs. We can require that the pattern should occur in all sequences or there is a minimum number of occurrences specified by the user. In some cases the number of occurrences is not specified but it is a part of the scoring function – a longer pattern with fewer occurrences is sometimes more interesting than a shorter pattern with more occurrences. The situation is more complicated in the case of probabilistic patterns, such as Hidden Markov models. Deterministic patterns either match the sequence or not (zero or one), whereas probabilistic models give a probability between 0 and 1. Therefore, there are different degrees of "matching". It is necessary to set some threshold on what should be considered a match or to integrate these matching probabilities to the scoring scheme.

Methods for scoring patterns also differ. A score can reflect the pattern itself only (e.g. its length, degree of ambiguity etc.) or it can be based on the occurrences of the pattern (their number, how much these occurrences differ from the pattern). Scoring functions are sometimes based on statistical significance. For example we may ask, what is the probability that the pattern would have so many occurrences if the sequences were generated at random. If this probability is small, the pattern is statistically significant. More detailed discussion of statistical significance of patterns can be found in Section 4.

The goal of an algorithm is to find the highest scoring pattern, or to find several best scoring patterns, or all patterns with some predefined level of support and score.

3 Algorithms for Pattern Discovery

3.1 Exhaustive search

It has been proved that many computer science problems related to pattern discovery are computationally hard tasks. It means that one cannot hope to find a fast algorithm that would guarantee the best possible solution. Thus, many approaches are based on the exhaustive search. Although such algorithms in the worst case may run in exponential time, they often use sophisticated pruning techniques that make the search feasible for typical input data.

3.1.1 Enumerating all patterns

The simplest exhaustive search works as follows. All possible patterns satisfying constraints given by the user are enumerated. For each such pattern the program finds its occurrences in input sequences and based on these occurrences assigns a score or statistical significance to the pattern. We can then output patterns with highest score or all patterns with scores above some threshold.

For example if we want to identify the most significant nucleotide pattern of length 10 with at most 2 mismatches, we can enumerate all possible strings of length 10 over the alphabet $\{A, C, G, T\}$ (there are

$4^{10} = 1,048,576$ such strings). Each string is a potential pattern. We find all its occurrences with at most 2 mismatches in input sequences and compute the score. We report the pattern with the highest score.

This method is suitable only for short and simple patterns, because the running time increases exponentially with the length of the pattern. The number of possibilities is even larger if we allow patterns containing wild-cards, ambiguous characters, and gaps. On the other hand, the advantage of this method is that with increasing length of the input sequences the running time usually increases linearly. Therefore the enumeration approach is suitable for identifying short patterns in a large amount of data.

Exhaustive search is guaranteed to identify the best pattern. We may easily output an arbitrary number of high scoring patterns, we may also choose relatively complicated scoring functions, as far as they can be easily computed based on the pattern and its occurrences. We can also allow mismatches, even insertions and deletions.

Application of enumerative methods. Many protein binding sites in DNA are actually short ungapped motifs, with certain variability. They can be quite well modeled with simple patterns allowing a small number of mismatches. Therefore we can apply exhaustive search to identify these types of binding sites. This method has been recently used in [van Helden et al., 1998, Tompa, 1999]. Simplicity of the exhaustive search allowed the authors to develop sophisticated methods for pattern statistical significance estimation.

Enumerating gapped patterns. In some contexts it is more reasonable to search for patterns with gaps. MOTIF [Smith et al., 1990] is an example of such system. MOTIF finds patterns with 3 conserved amino acids separated by two fixed gaps (for example `A...Q....I`). The gaps can have length $0, 1, \ldots, d$ where d is a parameter specified by the user. The number of possible patterns is $20^3 d^2$. MOTIF does not allow any mismatches, but the pattern does not need to occur in all sequences.

If the sequences contain a conserved region of more than 3 positions, there will be many patterns, each containing a different subset of the conserved positions from this region. Therefore, in the following step the algorithm removes the patterns occurring close to each other. For each of the remaining patterns all occurrences of the pattern are aligned. Based on the alignment the pattern is extended by finding consensus in the columns of the alignment. Patterns are also extended to both sides where possible.

Pruning pattern enumeration. If we want to identify longer or more ambiguous patterns, we cannot use a straightforward exhaustive search. For example, assume that we want to identify a long ungapped pattern occurring (possibly with some mismatches) in at least k sequences. We will start from short patterns (for example patterns of length 1) that appear in at least k sequences and extend them while the support does not go below k. In each step we extend the pattern in all possible ways and check whether the new patterns still occur in at least k sequences. Once we obtain a pattern that cannot be extended without loss of support, this pattern is maximal and can be written to the output. This search strategy is actually a depth first search of the tree of all possible sequences (see Figure 2). We prune branches that cannot yield supported patterns.

Improvements of this kind perform well in practice. However, in the worst-case scenario the running time still can be exponential. The main advantage of such improvements is that they allow to search for longer and more complicated patterns than simple exhaustive search. Examples of this strategy include Pratt algorithm described in detail below and the first, scanning phase of TEIRESIAS algorithm [Rigoutsos and Floratos, 1998b] (see also part 3.2.1).

Pratt. Pratt [Jonassen, 1996] is an advanced algorithm based on the idea of a depth first search in a tree of patterns. Pratt discovers patterns containing flexible gaps (with lower and upper bound of the gap length) and ambiguous symbols. Each pattern discovered is required to match exactly at least some predetermined number of sequences. The user has to specify several parameters that restrict the type of patterns. These include the maximum total length of the pattern, maximum number of gaps, maximum number of flexible gaps, and set of allowed ambiguous symbols.

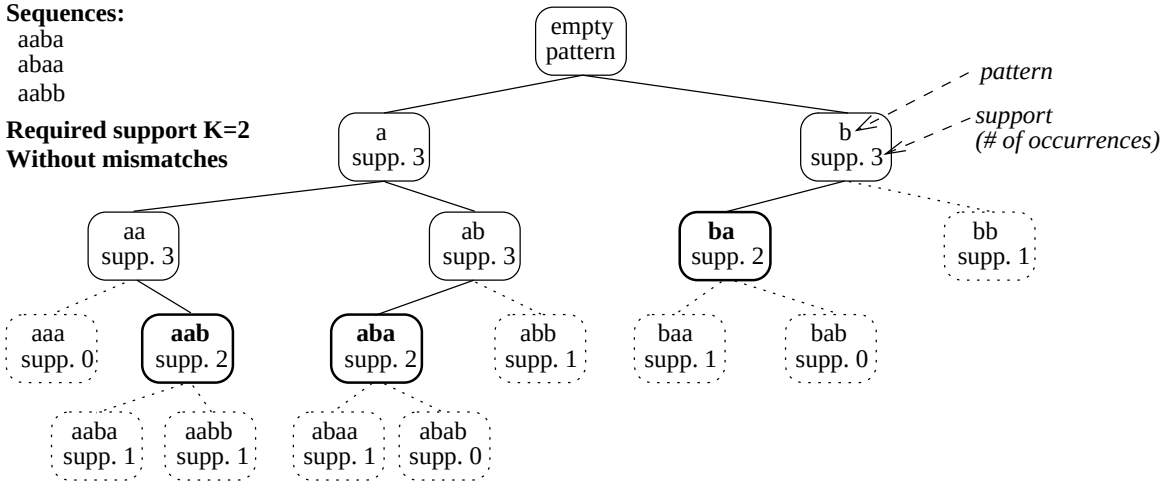

Figure 2: One way to improve exhaustive search is to search in a tree of all possible patterns. When we discover a node corresponding to a pattern that does not have enough support, we do not continue to search its subtrees. Dashed nodes do not have enough support. Bold nodes are patterns that cannot be further extended.

To reduce the size of the output and the running time the program does not report patterns that are less specific than other discovered patterns. Here pattern A is more specific than pattern B, if any sequence that matches A must also match B (for example B is less specific if it can be obtained from A replacing non-ambiguous character with ambiguous, or making a gap more flexible). This is achieved by a special scoring function that gives a higher score to more specific patterns.

In each step of the depth first search we take an existing pattern with sufficient support and we add a gap (possibly of length 0) and another character or ambiguous character. All such possibilities are tried. New patterns without a sufficient support are then discarded. This is done by a special data structure which makes the search faster. Additional optimizations are carried out by discarding patterns that cannot be extended to the most significant pattern.

The Pratt algorithm is guaranteed to find the pattern with the highest score when no flexible gaps are allowed. If we allow flexible gaps, the returned pattern may not be the highest scoring since a heuristic is used to speed up the search.

3.1.2 Exhaustive search on graphs

Not all exhaustive search methods enumerate all relevant patterns. It is also possible to enumerate all combinations of substrings of given sequences that can be possible occurrences of a pattern. Assume we have n sequences and we want to identify a pattern of given length L which occurs in all sequences with at most d mismatches. Then any two occurrences of such pattern differ in at most $2d$ positions, because they both differ from the pattern in at most d positions. Therefore, we can identify the pattern by finding a group of n substrings of length L, each from a different sequence, such that any two substrings differ in at most $2d$ positions [Pevzner and Sze, 2000].

This can be formulated as a problem in graph theory as follows. Each substring of length L will be a vertex of a graph G. Vertices corresponding to two substrings will be connected by an edge if the substrings are taken from different sequences and differ in at most $2d$ positions (see Figure 3a). This graph is n-partite, which means that it can be partitioned to n partitions so that there is no edge between vertices in the same partition. In this case partitions correspond to individual sequences. We want to find a set of n vertices such that any two vertices are connected by an edge. Such a set of vertices is called a clique.

The problem of finding a clique is known to be NP-hard. NP-hard problems are computationally difficult

$n = 4$, $d = 1$, $L = 3$
Sequences:
`abde`
`afcg`
`hbci`
`jbck`

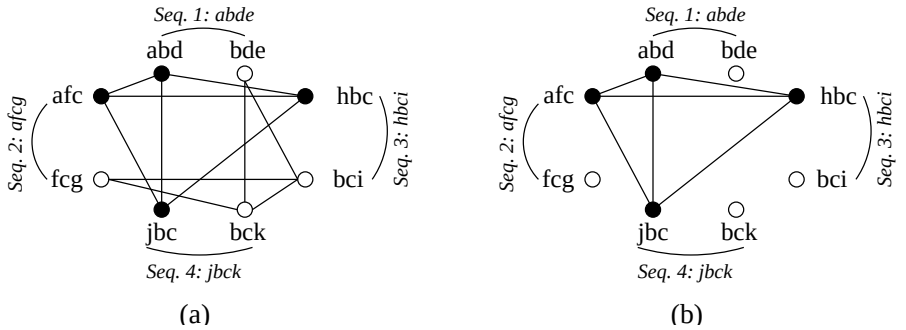

Figure 3: Part (a) shows the graph corresponding to the depicted set of sequences. Part (b) shows the same graph after removing edges with the WINNOWER algorithm. This graph contains exactly one clique corresponding to pattern `abc`.

problems. There is no known algorithm for solving an NP-hard problem in polynomial running time.

One possibility how to find a clique is to enumerate combinations of vertices and test each combination for being a clique. Such an approach has an exponential running time because there are many possible combinations. In order to make the software practical we need to add careful pruning that would eliminate large groups of vertex combinations that are guaranteed not to contain a clique.

Algorithm WINNOWER [Pevzner and Sze, 2000] eliminates combinations by first modifying the graph itself. It attempts to reduce the number of edges in the graph, removing only edges that cannot be part of any clique of size n. In this way we may obtain a graph with less edges that will be easier to search for a clique.

Even if we find a combination of n substrings of length L, each two differing in at most $2d$ positions, it does not guarantee that we have found a pattern. For example assume that we want a pattern of length $L = 4$ with at most one mismatch and we have found the following 3 occurrences: `AAAA`, `BBAA` and `CCAA`. Any two occurrences differ in exactly $2 = 2d$ positions but there is no pattern that would differ from each occurrence in at most one position. However, we may assume that this would not happen very often and that most combinations found will actually correspond to a pattern.

Usually the user wants to know not only the set of occurrences but also the corresponding pattern. One possibility to identify the pattern is to enumerate all patterns that occur within distance d from one chosen occurrence. There are at most $\binom{L}{d}(|\Sigma| - 1)^d$ such patterns. This number is exponential in d but not in L, and d is typically small. For each possible pattern we verify whether it is within distance d from all other occurrences as well. The search can be further pruned by using knowledge about the other occurrences. Alternatively we can use the set of occurrences as a starting point of Gibbs sampling or other iterative method (see part 3.3.1, 3.3.2). This is of course not guaranteed to find the pattern with specified parameters, even if one exists.

3.2 Creating long patterns from short patterns

A pattern cannot be significant unless it is sufficiently long. However long patterns are more difficult to identify using enumerative techniques. One possible approach for identifying long patterns is to start with shorter patterns and then combine them together. Perhaps the most elegant example of such an algorithm is TEIRESIAS [Rigoutsos and Floratos, 1998b]. This algorithm is based on a well-organized exhaustive search through possible combinations of shorter patterns. In the worst case the algorithm runs in exponential time, but in practice it works very well. Further effort has yielded a different algorithm that runs in polynomial time (see part 3.2.2).

3.2.1 TEIRESIAS algorithm

TEIRESIAS searches for $\langle L, W \rangle$ patterns defined as follows (L and W are constants specified by the user).

Definition 1 *Pattern P is an $\langle L, W \rangle$ pattern if it meets the following rules:*

- *P consists of characters from Σ and wild-cards '.'*
- *P starts and ends with a character from Σ (i.e non-wildcard)*
- *any substring of P that starts and ends with a non-wildcard and contains exactly L non-wildcards has length at most W (this condition is called density constraint).*

The density constraint eliminates patterns with long stretches of wildcards. Consider for example $L = 3$ and $W = 5$. String `AF..CH..E` is a valid $\langle 3, 5 \rangle$ pattern, however string `AF.C.H..E` is not (substring `C.H..E` has length 6).

Maximal patterns. TEIRESIAS discovers all $\langle L, W \rangle$ patterns that occur in at least K input sequences ($K \geq 2$ is also specified by the user). However, out of several patterns having the same set of occurrences it outputs only one pattern. This is selected as follows.

Pattern P is said to be *more specific* than pattern Q if Q can be derived from P by removing several (possibly 0) characters from both ends of P and replacing several (possibly 0) non-wildcards with wildcards. For example `AB.CD.E` is more specific than `AB..D`.

If pattern P is more specific than pattern Q, then every occurrence of P is also an occurrence of Q. If Q has the same number of occurrences as P, it is not useful to report both P and Q because they have the same set of occurrences and P contains more information. Therefore the algorithm only outputs pattern P if there exists no other more specific pattern with the same number of occurrences. Patterns reported by the algorithm are called *maximal*.

Note that if P is more specific than Q and Q has more occurrences than P, i.e. Q has greater support, Q is outputted as well. This is because although Q has a smaller specificity, it has greater support.

Algorithm. The TEIRESIAS algorithm is based on the concept that if a pattern P is a $\langle L, W \rangle$ pattern occurring in at least K sequences, then its subpatterns are also $\langle L, W \rangle$ patterns occurring in at least K sequences. Therefore the algorithm assembles the maximal patterns from smaller subpatterns.

TEIRESIAS works in two phases. In the first phase (called *scanning phase*) it finds all $\langle L, W \rangle$ patterns occurring in at least K sequences that contain exactly L non-wildcards. This is carried out by a pruned exhaustive search (see 3.1.1). In the second, *convolution phase* these elementary patterns are extended by gluing them together. In order to determine whether two patterns P and Q can be glued together we compare the suffix of P containing exactly $L - 1$ non-wildcards to the prefix of Q containing exactly $L - 1$ non-wildcards. If the suffix and the prefix are equal, P and Q can be glued together so that the $L - 1$ non-wildcards overlap. The list of occurrences of the resulting pattern can be constructed from the lists of occurrences of P and Q (we do not need to scan all sequences). Only when the resulting pattern occurs at least K times, is it retained.

For example let P =`AB.CD.E` and Q =`DFE.G` (with $L = 3$, $W = 5$). In this case P and Q cannot be glued together, because `D.E`$\neq$`DF`. However if Q =`D.E.G` we can glue them together obtaining `AB.CD.E.G`. If occurrences of P are $(1, 1), (2, 3), (2, 6), (4, 7)$ (each pair gives a sequence and a position in the sequence) and occurrences of Q are $(1, 5), (2, 8), (2, 10)$, then the list of occurrences for the new pattern is $(1, 1), (2, 6)$.

In the convolution phase we take each elementary pattern, and we try to extend it on both sides by gluing it with other elementary patterns in all possible ways (depth first search). Any pattern that cannot be extended without loss of support can potentially be maximal. However we can still obtain non-maximal patterns in the output and some patterns can be generated more than once. Therefore a list of patterns written to the output is maintained. In this manner we can check any newly generated pattern with the list and if the list contains a more specific pattern with the same set of occurrences we simply discard the new pattern.

The TEIRESIAS algorithm is an exact algorithm. It is guaranteed to find all $\langle L, W \rangle$ maximal patterns supported by at least K sequences. The number of such patterns can be exponential [Parida et al., 2000]. In such cases TEIRESIAS will require exponential time to complete. However, such a situation is not likely to occur in real data. For example, entire GenPept database with 120 million amino acids contains only 27 million maximal patterns (see [Rigoutsos et al., 2000]). Experimental studies suggest that running time of TEIRESIAS algorithm is linear in the number of patterns it outputs [Rigoutsos and Floratos, 1998a].

Patterns discovered by the TEIRESIAS algorithm are not very flexible. First of all, the only mismatches allowed are wildcard characters. Newer versions of TEIRESIAS ([Rigoutsos et al., 2000]) can now also identify patterns containing ambiguous characters representing pre-specified groups of characters from Σ. Second, TEIRESIAS patterns do not allow gaps with flexible length. This problem can be addressed by the post-processing phase, where the found patterns are combined into larger patterns separated by flexible gaps ([Rigoutsos and Floratos, 1998a]). However, such methods do not guarantee that all patterns of the specified form will be found.

3.2.2 Improvement of running time

Irredundant patterns. One of the drawbacks of TEIRESIAS is the potentially exponential size of the output and thus potentially exponential running time. This issue has been addressed [Parida et al., 2000]. This new algorithm computes only a subset of maximal patterns, called irredundant patterns. Any pattern can be easily obtained from the set of irredundant patterns. In any input of length n there are at most $3n$ irredundant patterns and these patterns can be found in $O(n^3 \log n)$ time. This is a substantial theoretical improvement compared to traditional exponential algorithms. However neither implementation of this algorithm, nor experimental study demonstrating the application of this approach is available to date.

3.3 Iterative heuristic methods

So far we have considered algorithms guaranteed to identify the best pattern. However for more complicated types of patterns we cannot hope to do so. We have to use heuristic approaches that do not necessarily find the best pattern, but may converge to a local maximum. The most important example of such technique is Gibbs sampling.

3.3.1 Gibbs sampling

A heuristic algorithm for pattern discovery based on Gibbs sampling method was presented by [Lawrence et al., 1993]. In the simplest version, we are looking for the best conserved ungapped pattern of fixed length W in the form of position weight matrix. We assume that the pattern occurs in all sequences.

The algorithm is carried out in iterations. The result of each iteration is a set of subsequences of length W – one from each sequence. This set of subsequences represents the occurrences of the pattern. We can compute a position weight matrix characterizing the pattern from this set of occurrences. The algorithm works as follows:

- Randomly select one subsequence of length W from each input sequence. These subsequences will form our initial set of occurrences. Denote o_i occurrence in sequence i.
- **Iteration step.**
  - Randomly select one sequence i.
  - Compute position weight matrix based on all occurrences except o_i. Denote this position weight matrix P.
  - Take each subsequence of sequence i of length W and compute a score of this subsequence according to matrix P.

- Select a new occurrence o'_i randomly among all subsequences of i of length W using the probability distribution defined by the scores (higher score means higher probability).
- Replace o_i with o'_i in the set of occurrences.

- Repeat iteration, until a stop condition is met.

The Gibbs sampling algorithm does not guarantee that the position weight matrix and set of occurrences giving the best score will be found. Instead, the algorithm can converge to a local maximum. The method is fast, which makes it suitable for many applications.

Several problems related to Gibbs sampling have been identified and addressed in subsequent work.

- **Phase shifts.** Assume that the optimal set contains occurrences starting at positions 8, 14, 22 and 7 of the corresponding sequences. If we start with position 21 in the third sequence, the whole system is likely to converge to the set of occurrences 7, 13, 21 and 6 instead.

  The problem was addressed [Lawrence et al., 1993] by introducing an additional randomized step. In this step scores of the occurrences shifted by several characters are computed. One random shift is selected with probability distribution corresponding to the scores. Authors of PROBE [Neuwald et al., 1997] reduce or extend pattern on both sides in a similar manner.

- **Multiple patterns.** Sometimes it is appropriate to define a pattern as a sequence of several consecutive subsequences of fixed length separated by variable length gaps. It means that in this case each occurrence is represented by several short subsequences in the sequence rather than one. It is possible to identify such patterns by a modified Gibbs sampling [Lawrence et al., 1993, Neuwald et al., 1997] using dynamic programming in the process of ranking and choosing a new candidate occurrence. Lengths of subsequences and their number is specified beforehand.

- **Pattern width.** We have assumed, that the pattern width is fixed and is specified by the user. Most of the time it is not a reasonable assumption, especially if we are looking for multiple patterns separated by variable length gaps.

  In PROBE [Neuwald et al., 1997] a genetic algorithm is used to determine the parameters of patterns (i.e. the number of subsequences and their lengths). Two sets of parameters can be recombined (take part of the first and part of the second set) and in this manner a better set of parameters may be obtained. Sets of parameters for recombination are chosen at random with a distribution proportional to their score (called *fitness*). Fitness of the set of parameters is determined by the Gibbs sampling procedure.

- **Gapped patterns.** Not all positions within a continuous block of length W are necessarily important for the function of this block. Rather we want to create a pattern, which is gapped, i.e., only $J < W$ positions are used to form the model.

  This issue has been addressed [Liu et al., 1995]. The authors suggest to introduce yet another randomized step, in which we replace one of the J positions included in the pattern by one of the $W - J + 1$ positions, which are not included in the pattern. The choice is again random with a distribution of probabilities proportional to the corresponding scores.

3.3.2 Other iterative methods

Several other approaches use iterative methods similar to Gibbs sampling. Typically the algorithm starts with some pattern and finds the best fitting occurrence of this pattern in each sequence. Based on these occurrences it builds a pattern that best matches the occurrences. This process is then repeated with the new pattern until no improvement is obtained. The main difference between this algorithm and Gibbs sampling is that all sequences are used to define the new pattern and subsequently the position of the new pattern is refined in all sequences. The process is completely deterministic, and of course has no guarantee to find the global optimum.

This strategy was used in [Pevzner and Sze, 2000] to identify ungapped deterministic pattern of a given length that matches all sequences with mismatches. The goal is to minimize the total number of mismatches. By using different methods a set of occurrences of some unknown candidate pattern is obtained which can be refined by an iterative method. In each step a new pattern is computed by taking the most frequent character in each position (based on the frequencies in the occurrences). The method is further improved to remove non-significant columns from consideration, obtaining a gapped pattern.

An iterative method was also be used to detect coiled coil regions in histidine kinase receptors [Singh et al., 1998]. Coiled coils were previously detected in other protein families, therefore the statistical properties of such regions are known, although they may be somewhat different for this family. In this example the goal was to identify distribution of residues and pairs of residues at different distances apart in a sliding window of fixed length, provided that the window is from a coiled coil region. The process started with taking the known distribution from other families. Based on this information each position of sliding window was scored and the best scoring positions were the candidates for coiled coil region. A random sample of these candidates was used to compute a new distribution. This process was iterated. In each step a pseudocount from the known distribution of other families was added. In contrast to the previous method, this is randomized, and due to pseudocounts the result cannot diverge too much from the original pattern.

The iterative approach can be also used to improve position weight matrices [Zhang, 1998]. Here the author starts from a PWM computed for several signals from vertebrate genomes and refines them by iteration to obtain a PWM specific for human.

In general it seems that the simple iterative methods are suitable for improvement of patterns obtained by other methods or from different data. However, this approach is not sufficient to discover patterns without any prior knowledge.

3.3.3 From iteration to PTAS

The Consensus Pattern problem is another formulation of pattern discovery [Li et al., 1999]. The problem is defined as follows: find a pattern P and one occurrence of P in each sequence so that the total number of mismatches over all occurrences is minimized.

Some problems associated with pattern discovery are NP-hard. This means that it is unlikely that any polynomial time algorithm for such problems exists. The Consensus Pattern problem is one of them.

Since there is no algorithm guaranteed to find the best solution of the Consensus Pattern problem in a reasonable time, we may wish to have a guarantee that the cost of the found pattern (i.e. the total number of mismatches) is at most α times the cost of the optimal pattern. Value α is called the approximation ratio. For example if $\alpha = 2$ we are guaranteed to find a pattern that has at most twice as many mismatches as the best possible pattern.

For some problems it is possible to construct an algorithm that works for any α (supplied by the user). However, the smaller the approximation ratio, the longer the algorithm runs. This type of algorithm is called the polynomial approximation scheme, or PTAS. The PTAS for Consensus Pattern problem is based on a simple iterative idea repeated many times with different initial patterns [Li et al., 1999].

The PTAS requires input sequences, the desired length L of a pattern and a parameter r. It finds all possible combinations of r substrings of length L taken from input sequences. Each combination may contain zero, one, or several substrings from each sequence, some substrings may even repeat more than once. If the total length of all sequences is N, there are $O(N^r)$ combinations. For each combination of r substrings the following steps are performed:

- The majority pattern P of the r substrings is computed. This pattern has in each position the character occurring most frequently in this position in the r substrings.
- Find the best occurrence of P in each input sequence.
- Compute a new majority pattern P'.
- Find the best occurrences of P' in all sequences and compute the number of mismatches (cost of P').

The result will be the pattern P' which achieves the minimum cost. Notice, that the algorithm performs one step of iteration with the pattern obtained from each possible combination. The running time of the algorithm is $O(N^{r+1}L)$ and its approximation ratio is $1 + (4|\Sigma|A - 4)/(\sqrt{e}(\sqrt{4r+1} - 3))$ for $r \geq 3$.

This result is very interesting from the point of view of theoretical computer science. However the algorithm is not very practical. For example, if we choose $r = 3$ and $\Sigma = \{A, C, G, T\}$, the algorithm will identify the pattern with at most 13 times as many mismatches as the optimal pattern. The running time is $O(N^4L)$, impractical for large inputs. In order to achieve $\alpha = 2$, r needs to be at least 21 which gives an algorithm with prohibitive running time $O(N^{22})$. Of course, the approximation ratio is only an upper bound of the possible error. For some inputs the optimal or close-to-optimal results can be obtained even for small r, however there is no guarantee.

A program called COPIA [Liang et al., 2000] is based on the ideas of this PTAS with several changes that considerably reduce the running time. The enumeration of all possible combinations of r substrings is replaced by random sampling of combinations. The consensus pattern obtained from each randomly chosen combination of substring is improved by the iterative method until there is no further improvement (similarly as in [Pevzner and Sze, 2000]). COPIA runs in reasonable time for real data but it does not have the same guarantees of pattern quality as the PTAS algorithm.

3.4 Machine learning methods

Sometimes a pattern cannot be described well by a simple deterministic pattern and one may wish to express it in a form of a stochastic model, such as Hidden Markov model or position weight matrix (which is a simpler version of HMM). This kind of pattern is discovered using iterative expectation maximization techniques that do not necessarily converge to the global maximum.

3.4.1 Expectation maximization

First we will consider a simpler case of position weight matrices. A simple learning algorithm called expectation maximization (EM) is used to estimate parameters of the stochastic model of a pattern that occurs once at an unknown position in each input sequence [Lawrence and Reilly, 1990]. The algorithm can be easily extended to more complicated models, e.g., patterns with flexible gaps, a finite mixture model [Bailey and Elkan, 1994].

The algorithm is iterative. It starts with some initial model parameters (usually randomly set). Each iteration consists of two steps as follows:

- **E step.** For every sequence s and for every position in s compute the probability that the occurrence of the pattern in s starts at this position. The probability is based on the model from the previous iteration (or initial model for the first iteration).
- **M step.** For every position in the pattern compute new probabilities of characters at this position. This is based on all possible occurrences of the pattern weighted by probabilities computed in E step. These values will form new parameters of the model.

Notice that the algorithm uses all possible occurrences of the pattern to obtain a new matrix, instead of only one occurrence in each string. Similarly to other iterative methods, the EM algorithm converges to a local maximum depending on initial parameters of the model, instead of the global maximum likelihood. The other problem is the assumption that every pattern occurs exactly once in every sequence.

These two problems are addressed in the MEME algorithm [Bailey and Elkan, 1995], which is a modification of EM algorithm. The algorithm is based on the assumption that the pattern found should closely resemble at least one subsequence found in the dataset. They also modified formulas given in [Lawrence and Reilly, 1990] so that several or no possible occurrences of the pattern in a sequence could be considered. The algorithm proceeds as follows:

1. Form an initial model for each subsequence in the dataset. The initial model is a position weight matrix. For every position the character at the corresponding position in the subsequence has probability p (p is usually between 0.5 and 0.8), and all other characters have probability $(1-p)/(|\Sigma|-1)$.

2. One iteration of EM algorithm is performed on each such initial model. The likelihood score is computed for the resulting models.

3. The model with the largest likelihood score is selected as an initial model for EM algorithm.

The algorithm can be forced to report more patterns by erasing all occurrences of the found pattern from the dataset and rerunning the entire process.

3.4.2 Hidden Markov models

Hidden Markov Models (HMMs) can be used to model a family of sequences. A thorough introduction to HMMs and related algorithms has been provided [Durbin et al., 1998]. In brief, given an HMM and a sequence, it is possible to compute the most probable path through the model for this sequence (in $O(nm)$ time using Viterbi algorithm, where n is the length of the sequence and m is the number of states in the model). This path represents the most probable occurrence of the pattern in the sequence. The probability P that the sequence was generated by the model can also be computed (in $O(nm)$ time using forward algorithm). Value $-\log P$ is called a *NLL score of the sequence* with respect to the model. A higher probability of generating the sequence corresponds to a lower NLL score of the sequence.

There are three issues, that need to be addressed, if HMMs are used to represent a sequence family:

- **Topology of HMM.** Topology specifies the scheme of the Hidden Markov model that is used to represent a sequence family.
- **Training process.** The training process is needed to estimate the parameters of the model so that the sum of scores of sequences in the family is optimized.
- **Search for sequences.** The searching process should allow one to distinguish between sequences which belong to the family and sequences which do not.

Topology of HMM. A common HMM topology [Krogh et al., 1994, Hughey and Krogh, 1996] for sequence analysis is depicted in Figure 4. The model consists of three types of states. **Match states** model conserved parts of the sequence (motifs). Match states specify the probability distribution of characters at each conserved position. There can be any number of match states in the model. We assume that this number is given by the user beforehand. **Insertion states** model possible gaps between match states. Gaps can be arbitrarily long. Probability assigned to a self-loop in an insertion state determines probability distribution of possible gap lengths (the probability distribution is geometric, and the mean value can be easily computed). Finally, **deletion states** allow one to model occurrences of the pattern that do not contain some of the conserved positions.

Training of the model. Given a topology of HMM and a family of sequences to be modeled by the HMM, we can estimate the parameters of the model so that the model will generate sequences similar to those in the family with high probability.

The Baum-Welch algorithm (see [Durbin et al., 1998]) can be used to perform this task. It is an iterative algorithm very similar to the EM algorithm used to estimate parameters of PWMs. We start with arbitrary parameters (if we have some prior knowledge about the sequence family, we can use this knowledge to set the initial parameters). Then in each step, the probabilities of all paths for all sequences are computed, and the model parameters are reestimated to minimize the NLL score (maximize the probability) of the training sequences.

The algorithm does not guarantee finding of global optimum. It converges to a local minimum depending on initial parameter settings.

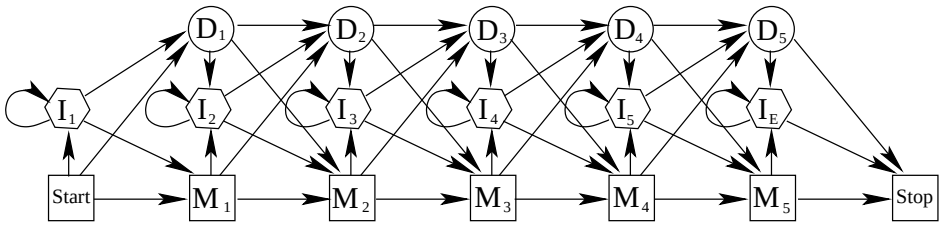

Figure 4: A common HMM topology for pattern discovery. States $M_1, \ldots, M_5$ are match states, $I_1, \ldots, I_E$ are insertion states, and $D_1, \ldots, D_5$ are deletion states.

Search for sequences. Search for the pattern in the form of HMM is more complicated than in the case of simpler patterns. Given a sequence one can efficiently compute the most probable alignment of the sequence to the pattern and compute its NLL score. The NLL score gives a measure of how well the sequence can be aligned to the pattern. However, NLL scores highly depend on a sequence length. In general, shorter sequences have smaller NLL scores than longer ones. Therefore, we cannot use a fixed threshold on NLL score to discriminate between members and non-members of the family. This is possible only if all the input sequences have approximately the same length. Fortunately it was observed by [Hughey and Krogh, 1996] that sequences, which do not belong to the sequence family, form a line corresponding to a linear dependency. NLL scores for family members significantly drop below this line. Thus, it is possible to identify members of the family by a statistical test using a z-score for some window of sequence length. In order to estimate parameters needed for z-score we need to compute NLL score for many background sequences of different lengths not belonging to the family.

3.4.3 Enhancing HMM models

Reducing the number of parameters. The greater the number of parameters the model has the greater the amount of data required to properly train the model. Too many parameters can cause overfitting, where the model fits the training data very well but does not generalize to new sequences. There are several possibilities of how to reduce the number of parameters of the model.

- **Model surgery.** Model surgery [Hughey and Krogh, 1996] adjusts the model topology during the training in order to reduce the number of parameters of the model. In particular, this technique avoids two common problems arising during the training:
  - **Some match states are used only by a few sequences.** If the number of sequences using a match state drops below a given threshold (typically one half), the state is removed. In this way we force sequences to either use the insertion state at this point, or significantly change their alignment.
  - **Some insertion states are used by too many sequences.** If an insertion state is used by more sequences than a given threshold (typically one half), then the state is replaced by a chain of match states. The number of inserted match states is equal to the expected number of insertions in the replaced insertion state.
- **Different initial topology**. Meta-MEME [Grundy et al., 1997] uses a different program to report simple short patterns in the sequence. These patterns are transformed into matching states in HMM. The patterns are combined together using insertion states as shown in Figure 5.

Discovering subfamilies. Sometimes a family of sequences consists of several subfamilies. In such cases the family is represented by several motifs rather than by one. This problem can be solved by combining several HMMs with standard topology to one larger HMM as shown in Figure 6 [Krogh et al., 1994].

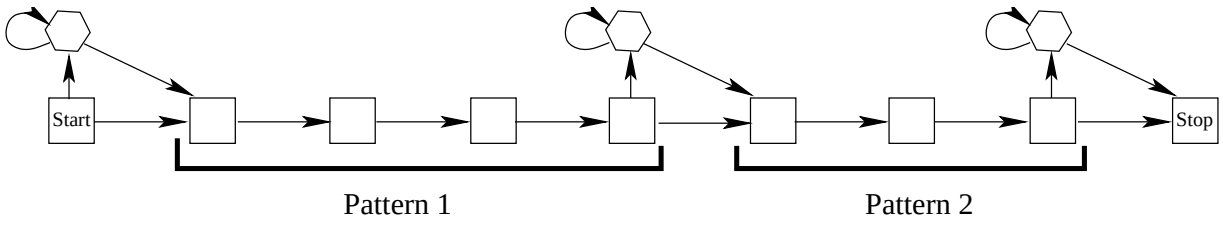

Figure 5: Meta-MEME uses much simpler initial topology. Patterns found by other programs are connected together by insertion states.

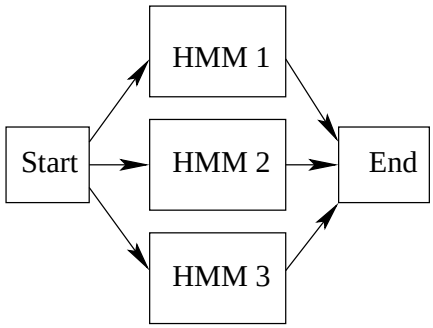

Figure 6: Topology of HMM suitable for representing families of sequences with several subfamilies. HMM 1, 2, and 3 are models representing individual subfamilies.

If we do not have any preliminary knowledge of how to set initial parameters of such a model, it might be difficult to accurately train the model. In this case it is appropriate to use the Viterbi algorithm for training. The Viterbi training algorithm uses only the best path to reestimate the parameters, in contrast to the Baum-Welch algorithm which uses all paths weighted by their probabilities. Therefore, once the parameters of a part of the model reflect a bias to one subfamily, only sequences in this subfamily are used to train this part of the model in the Viterbi algorithm.

3.5 Methods using additional information

Many biologically significant patterns are difficult to discover in the sequences. In such cases additional sources of information can be used to guide the search.

3.5.1 Identifying motifs in aligned sequences

Pattern discovery and multiple local alignment are closely related tasks. One can easily obtain a pattern (in a form of consensus sequence, PWM etc.) from a given local alignment by taking each column of the alignment as one position of the pattern. The question begins to be interesting if we assume that the input contains errors (i.e. some sequences are not aligned correctly) or it contains several subfamilies. In this case we may try to identify a pattern which does not match all sequences. This task is addressed by EMOTIF [Nevill-Manning et al., 1998].

EMOTIF searches for motifs containing characters, ambiguous characters, and wild-cards. The set of possible ambiguous characters is fixed. Each character (normal, ambiguous or wild-card) corresponds to one column of the local alignment. For each pattern of this form it is possible to compute its specificity (how likely it is to occur by random) and sensitivity (how many training sequences it covers). EMOTIF identifies many motifs with different values of specificity and coverage. For several motifs with the same specificity

only the one with the greatest coverage is reported and vice versa. Motifs are found by a pruned exhaustive search.

3.5.2 Global properties of a sequence

Computational recognition of eukaryotic promoters is a difficult task. In addition to local information in the form of transcription factor binding sites, it is also necessary to consider global properties of DNA in surrounding regions [Pedersen et al., 1999]. One example are CpG islands – 1-2kb long regions with higher frequency of CpG than found elsewhere in the genome [Pedersen et al., 1999, Durbin et al., 1998]. CpG islands often demarcate the 5' region flanking constitutively expressed vertebrate genes. In addition, downstream regions usually have low flexibility [Pedersen et al., 1999] (flexibility, or bendability of DNA can be estimated from sequence-based models of DNA structure).

In the pattern matching problem we can use global information, such as CpG islands and flexibility, to distinguish random occurrences of a pattern from those that have functional significance. In pattern discovery we may use this kind of prior knowledge to choose appropriate parts of the genome as our input set in which we search for patterns.

3.5.3 Using phylogenetic tree

One of the basic assumptions in identifying patterns in biological sequences is that regions conserved in evolution are functionally important. Therefore it is natural to use known phylogenetic relations among sequences to guide the pattern search.

Assume we want to identify a regulatory element. Instead of using regulatory regions from many co-regulated genes of the same species we will use regulatory regions of the same gene taken from many related species. We assume that the evolutionary tree of these species is known. Now we may try to identify the short pattern best conserved in the evolution.

The best conserved pattern can be identified using a parsimony measure [Blanchette et al., 2000] as follows. We are given the length of pattern k. We want to associate a sequence t_w of length k with each node w. In each leaf, t_w is required to be a subsequence of the input sequence associated with this leaf (t_w corresponds to an occurrence of the pattern). In internal nodes t_w can be arbitrary string. We want to minimize the sum $d(t_v, t_w)$ over all tree edges (v, w), where $d(t_v, t_w)$ is a distance between strings t_v and t_w (in this case number of substitutions).

The algorithm works as follows. First the tree is rooted in an arbitrary internal node. For each node w and each possible string t of length k let $d^*_w(t)$ be the best possible parsimony score that can be achieved in the subtree rooted at w provided $t_w = t$ (i.e. string t is stored in node w). Scores $d^*_w(t)$ can be computed in a leaves-to-root fashion.

The scores are easily found for leaf nodes: if t is a substring of the input string associated with the leaf w, then $d^*_w(t) = 0$, otherwise the score will be ∞. Once we know all scores for both children w_1 and w_2 of a certain internal node w, we can compute the scores for w. If we assume that the node w_1 stores sequence t_{w_1}, w_2 stores t_{w_2} and w stores t_w, then the parsimony of the subtree rooted at w will be:

$$[d^*_{w_1}(t_{w_1}) + d(t_{w_1}, t_w)] + [d^*_{w_2}(t_{w_2}) + d(t_{w_2}, t_w)].$$

For each possible t_w we want to find t_{w_1} and t_{w_2} that minimize this sum and store the sum as $d^*_w(t_w)$. After we compute scores for all nodes, we can retrieve the overall minimum parsimony as the smallest score computed for the root. We can use the stored intermediate results to reconstruct the entire optimal solution in a root-to-leaves manner. Strings t_w stored in the leaves represent the occurrences of the pattern.

With additional optimizations the algorithm can be implemented to run in time $O(nk|\Sigma|^k)$ where n is the number of leaves. This is exponential in k but the problem is NP-hard, so we cannot hope to find polynomial algorithm.

Using sequences from different genomes we can discover regulatory elements that regulate only very small number of genes in one genome. This cannot be done if we only use information from one genome. The phylogenetic tree helps to solve another problem, namely inputs containing groups of highly similar

sequences. Such sequences force other methods to find a pattern that characterizes similarity among the sequences in the group, but not features common to all sequences in the input. Therefore it is often necessary to first cluster close homologs together and choose only one member from each cluster as an input to pattern discovery programs. Since close homologs are grouped together in the phylogenetic tree, their weight is not so great – the pattern still has to agree with other parts of the tree.

3.5.4 Use of secondary/tertiary structure

Positions important for secondary and tertiary structure of proteins are usually well conserved. If we know the structure of proteins in question, we can try to locate regions important for achieving this structure. These regions are good candidates to identify occurrences of our pattern. One possibility is to choose points of contact between two secondary structure elements as candidate spots for conserved positions. This approach has been used to construct a sparse deterministic pattern containing ambiguous characters and flexible gaps [Ison et al., 2000]. A pattern should cover the entire length of a protein. Sequences are aligned so that the points of contact align together, if possible. In such an alignment we can choose positions that are well-preserved (among those columns that contain many points of contact). Some steps of this process were carried out manually, but certainly it is possible to implement a similar process as a program.

Secondary structure and search for motifs are also closely knit together in algorithms for identifying conserved patterns in RNA sequences [Gorodkin et al., 1997b]. RNA molecules are more related in their structure than in their sequence. Identifying RNA secondary structure of a set of related RNA sequences is best accomplished by first aligning the sequences. Alignment in turn requires to discover similarities. Therefore the algorithm attempts simultaneously discover the alignment, secondary structure features, and conserved patterns.

3.6 Finding homologies between two sequences

Finding homologies between two DNA or protein sequences is a special case of the general pattern discovery problem. Here, the problem becomes simpler in principle but with larger amounts of data, the challenge shifts to efficiency and scalability. It is not our intention to survey the entire field of homology searching, we will only consider a specific problem of comparing two very long genome sized DNA sequences, to shed some light on this problem. We do not discuss programs that compare or translate into protein sequences.

In theory, this problem is easily solved by standard dynamic programming techniques [Pevzner, 2000, Gusfield, 1997, Smith and Waterman, 1981]. However, when sequences are long, the Smith-Waterman local alignment dynamic programming and FASTA [Lipman and Pearson, 1985] strategies become too expensive, and scalable heuristics are required.

Two strategies have lead to improvements. The first is exemplified by the popular Blast family of algorithms [Altschul et al., 1990, Gish, 2001, Altschul et al., 1997, Zhang et al., 2000, Tatusova and Madden, 1999]. This approach finds short exact "seed" matches (hits), which are then extended into longer alignments. However, when comparing two very long sequences, SIM [Huang and Miller, 1991], Blastn (BL2SEQ [Tatusova and Madden, 1999]), WU-Blast [Gish, 2001], and Psi-Blast [Altschul et al., 1997] run slowly and require large amounts of memory. SENSEI [States and Agarwal, 1996] is somewhat faster and uses less memory than the above, but it is currently limited to ungapped alignments. MegaBlast [Zhang et al., 2000] runs quite efficiently without a gap open penalty and a large seed length of 28 yielding a much lower sensitivity.

Another strategy, exemplified by MUMmer [Delcher et al., 1999], QUASAR [Burkhardt et al., 1999] and REPuter [Kurtz and Schleiermacher, 1999], uses suffix trees. Suffix trees suffer from two problems: They were designed to deal with precise matches and are limited to comparison of highly similar sequences [Delcher et al., 1999, Burkhardt et al., 1999, Kurtz and Schleiermacher, 1999] and handle mismatches in an awkward manner. MUMmer and QUASAR implement various ways of linking neighboring precisely matched blocks. The second problem with suffix trees is that they have an intrinsic large space requirement.

A new program PatternHunter has recently been developed [Ma et al., 2000]. PatternHunter uses optimized spaced seeds for high sensitivity and improved search and alignment algorithms. It is implemented in Java and runs at the speed of MegaBlast and the suffix tree program family while producing outputs at

default Blastn sensitivity. PatternHunter was tested against the newly improved Blastn (using BL2SEQ) and MegaBlast, downloaded from the NCBI website on July 9th 2001. All experiments were performed on a 700 MHz Pentium III PC with 1Gbyte of memory. The experiments were run on *M. pneumoniae* vs *M. genitalium*, *E. coli* vs *H. influenza*, and *A.thaliana* chromosome 2 vs *A.thaliana* chromosome 4. All programs were run without filtering (bl2seq option -F F) to ensure identical input to the actual matching engines. With filter on, Figures 7, 8 essentially remain the same. All comparisons were based on same scoring reward and penalties to insure the output results were comparable. MB28 is MegaBlast with seed size 28. PH is PatternHunter at sensitivity similar to Blastn length 10 seed, and PH2 is PatternHunter at sensitivity similar to Blastn length 11 seed. Figures 7, 8, 9 show the output quality of PatternHunter vs Blastn and MegaBlast. In Figure 7, MegaBlast using seed weight 28 (MB28) misses over 700 high scoring alignments. Using the same parameters, PatternHunter outputs comparable or better results than Blastn. It is 20 times faster and uses one tenth the memory, Figure 8. Figure 9 shows that MegaBlast produces alignments with significantly lower scores compared to PatternHunter (PH2), which uses only one fifth the time and one quarter the space, on arabidopsis chromosomes. Figure 10 shows a genome alignment of *M. Pneumoniae* versus *M. Genitalium*, by PatternHunter and MUMmer [Delcher et al., 1999]. The table in Figure 11 compares the time and space used by PatternHunter (PH2). For example, the comparison of human chromosome 22 (35M bases) vs human chromosome 21 (26.2M bases). only required one hour to complete.

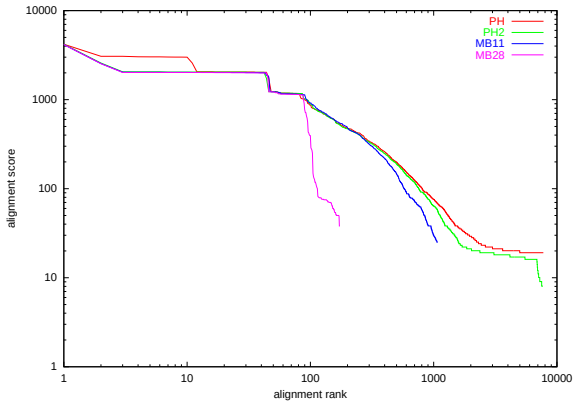

Figure 7: Input: *H. influenza* and *E. coli*. Score is plotted as a function of the rank of the alignment, with both axes logarithmic. MegaBlast (MB28) misses over 700 alignments of score at least 100. MB11 is MegaBlast with seed size 11 (it is much slower and uses more memory), indicating the missed alignments by MB28 are mainly due to seed size.

4 Assessment of Pattern Quality

A wide variety of methods for pattern discovery are available. They differ in what they consider to be the best pattern and most are not guaranteed to identify the best pattern. In consideration of this we need to evaluate the quality of the discovered patterns. Various statistical methods have been employed to address the question how likely is it that the pattern occurs in our sequences merely by chance. The smaller the likelihood the bigger is the chance that the pattern discovered has biological meaning. Statistical significance of a pattern is used to evaluate the performance of different algorithms, but also directly in the algorithms as a scoring function to rank discovered patterns for a user, or to guide a search for the most significant pattern.

Although statistical significance is an important tool that allows one to distinguish artifacts of search algorithms from significant patterns one must keep in mind that the goal of pattern discovery is to identify elements of certain biological importance. Even a very significant pattern may not be what we are looking

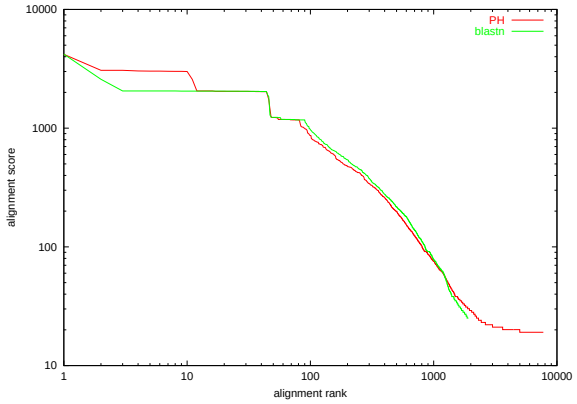

Figure 8: Input: *H. influenza* and *E. coli*. PatternHunter produces better quality output than Blastn while running 20 times faster.

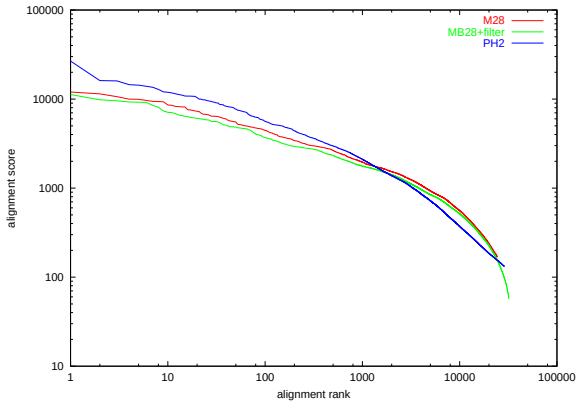

Figure 9: Input: *A. thaliana* chr 2 and chr 4. PatternHunter (PH2) outscores MegaBlast in one sixth of the time and one quarter the memory. Both programs used MegaBlast's non-affine gap costs (with gapopen 0, gapextend -7, match 2, and mismatch -6) to avoid MegaBlast from running out of memory. For comparison we also show the curve for MegaBlast with its default low complexity filtering on, which decreases its runtime more than sixfold to 3305 seconds.

for. For example we may want to discover functionally important sites but the pattern was conserved because it was essential for structure instead. Therefore it is important to verify patterns by appropriate biological experiments (for example mutagenesis or appropriate biological assay to verify function of a protein, x-ray crystallography or NMR to determine the structure of a protein, DNA footprinting to verify binding sites etc.)

4.1 Background model

If we want to understand what is the probability that a pattern occurs by random we need to define "random". In other words we need to select a background model. The simplest background model assumes that all possible characters of the alphabet are equally likely and individual positions of the sequence are independent. Therefore all possible sequences of characters of the same length are equally likely.

This model is usually not adequate because different characters of the alphabet (i.e. individual nucleotides or amino acids) occur in biological sequences with different frequencies. For example in an AT rich sequence we can expect that for example string '`TAATA`' will be more frequent that '`CGGCG`'. We can solve this problem

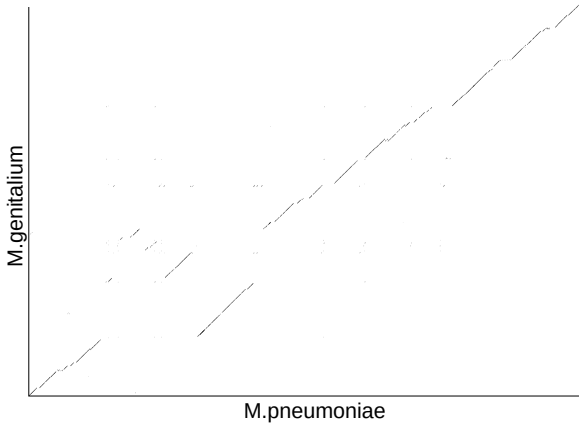

Figure 10: Highscoring Segment Pairs plot of *M. Pneumoniae* versus *M. Genitalium.*

Seq1	Size	Seq2	Size	PH	PH2	MB28	Blastn
M. pneumoniae	828K	*M. genitalium*	589K	10s/65M	4s/48M	1s/88M	47s/45M
E. coli	4.7M	*H. influenza*	1.8M	34s/78M	14s/68M	5s/561M	716s/158M
A.thaliana chr 2	19.6M	*A.thaliana* chr 4	17.5M	5020s/279M	498s/231M	21720s/1087M	∞

Figure 11: Performance Comparison: If not specified, all with match 1, mismatch -1, gap open -5, gap extension -1. Table entries under PH, PH2, MB28 and Blastn indicate time (seconds) and space (megabytes) used; ∞ means out of memory or segmentation fault.

by using the *Bernoulli model*. In this model each character of the alphabet potentially has a different probability but individual positions of the sequence are still independent.

An even more complicated model is *Markov chain* in which probability of each character on position j depends on characters on positions $j-1$, $j-2$,..., $j-k$, where k is a parameter called order of the Markov chain. Markov chain of order 0 is identical to the Bernoulli model. Markov chains take into account the fact that some combinations of characters occur less or more frequently than expected based on the frequencies of their constituents.

Parameters of background models (i.e. probabilities of individual characters) can be estimated as a function of the observed frequencies in the input sequences or in some larger databases.

4.2 Pattern significance

Given a deterministic pattern P and a sequence of length L we can simply count the number of occurrences of P in the sequence. Denote this number N_P. Let $E(X_{P,L})$ be the expected number of occurrences of pattern P in a sequence of length L generated by a background model. If the observed number of occurrences N_P is much higher than expected value $E(X_{P,L})$, P is then a significant pattern. Standard measure used in this context is z-score

$$z_P = \frac{N_P - E(X_{P,L})}{\sigma(X_{P,L})}$$

where $\sigma(X_{P,L})$ is the standard deviation of the number of occurrences of P in a random sequence of length L. This measure gives the number of standard deviations by which the observed value N_P differs from the expected value.

The simplest approach towards computing z-score is to generate a large number of sequences using the chosen background model, count the number of occurrences of the pattern and estimate the expected value and standard deviation from this random sample [Pesole et al., 2000]. This works for any kind of background model and pattern but it has a high running time (especially if we need to evaluate z-score for many patterns) and also the obtained values are only estimates. The mean and variance of the distribution of the number of occurrences of a given pattern can often be computed exactly. For example algorithms for a large class of patterns and for the background distribution being a Markov chain of order k are known [Nicodème et al., 1999].

4.3 Information content

In case of probabilistic models such as position weight matrices we do not have strictly defined occurrences but rather a score between 0 and 1 for any string. One possibility is to set a threshold on what we consider to be an occurrence and then evaluate z-score or other appropriate statistical measures. Alternatively for evaluating position weight matrices an *information content* (also called relative entropy) measure is used. This tells how much the distribution defined by the PWM differs from the (Bernoulli-type) background distribution. Relative entropy is computed as follows:

$$\sum_i \sum_c A[c,i] \log_2 \frac{A[c,i]}{f(c)}$$

where $A[c,i]$ is the frequency of character c in column i of the matrix and $f(c)$ is the background frequency of the character c. Relative entropy has two disadvantages. First, it does not depend on the number of occurrences of the pattern in the sequences. A strong pattern with very few occurrences has a higher relative entropy than a weaker pattern with few occurrences. Therefore, it is an appropriate measure only in situations where the pattern is required to occur in all sequences. This is often the case in Gibbs sampling methods. Second, relative entropy of one column is always non-negative and therefore if we add columns which are not well-conserved to the pattern, we can obtain a better score. Therefore relative entropy is not suitable for comparing patterns of different lengths. This can be solved by subtracting the appropriate term from the contribution of each column so that the expected contribution is zero [Rocke and Tompa, 1998].

4.4 Sensitivity and specificity of classification

One application of the pattern discovery methods is to identify patterns that characterize a given family of related proteins. In this context we need to measure how well we can distinguish members of the family from non-members based on the occurrence of the pattern. For this purpose a test set consisting of proteins with a known family is required. We find all occurrences of the motif in the test set and compute the following four scores: TP (*true positives*) are proteins that contain the motif and belong to the family in question, TN (*true negatives*) are proteins that do not belong to the family and do not contain the motif, FP (*false positives*) are proteins that contain the motif but do not belong to the family and FN (*false negatives*) are proteins that do not contain the motif but belong to the family. Thus $TP + TN$ is the number of correct predictions and $FN + FP$ is the number of wrong predictions. Based on counts of TP, TN, FP, FN we can define various measures [Brazma et al., 1998]. *Sensitivity* (also called coverage) is defined as $TP/(TP+FN)$ and *specificity* is defined as $TN/(TN+FP)$. A pattern has maximum sensitivity, if it occurs in all proteins in the family (regardless of the number of false positives) and it has maximum specificity, if it does not occur in any sequence outside the family. Score called *correlation coefficient* gives overall measure of prediction success:

$$C = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FN)(TN+FP)}}$$

This expression grows from -1 to 1 as the number of correct prediction increases.

5 Concluding Remarks

In this chapter we have presented an overview of methods available for pattern discovery. The tools developed by computer scientists are common today in many biological laboratories. They are required to handle large-scale data, including annotation of newly sequenced genomes, organization of proteins into families of related sequences, or identifying regulatory elements in co-expressed genes. They are also important in smaller scale projects because they can be used to detect possible sites of interest and assign putative structure or function to proteins. Thus, they can be used to guide biological experiments in "wet labs", decreasing the time and money spent in discovering new biological knowledge.

Indeed, there are many examples, where computational tools have helped biologists to make important discoveries. Pattern discovery tools helped to identify a number of putative secretory proteins in Mycobacterium tuberculosis genome [Gomez et al., 2000]. Subsequently, 90% of the predicted candidates were experimentally confirmed. Identification of M. tuberculosis secretory proteins is a first step to the design of more effective vaccines against tuberculosis.

In order to fully understand the meaning of the output of a pattern discovery tool, biologists need to understand the basics of the algorithm. It is very useful to know the performance guarantees of the algorithm. Tools that cannot guarantee finding best or all patterns, might find only low-scoring patterns. However, that does not mean, that high-scoring patterns do not exist.

The pattern discovery process is often a computationally intensive task. Therefore many databases are maintained and updated containing results of pattern discovery applied to particular tasks. These databases often contain also experimental evidence from biological literature and other useful information. In supplement to this chapter on the accompanying CD-ROM we provide a list of such databases together with related links and short descriptions of database contents. An overview of the software tools is included on the CD-ROM.

Acknowledgements

Parts of this chapter are based on technical report [Brejová et al., 2000] and some material from paper [Ma et al., 2000]. We would like to thank co-authors of the report Chrysanne DiMarco, Sandra Romero Hidalgo, Gina Holguin, and Cheryl Patten and co-authors of the paper Bin Ma and John Tromp for cooperation and helpful discussion. We want to also thank Stephen Krawetz, Jonathan Badger, Paul Kearney, and Marthenn Salazar who kindly reviewed parts of the material.

References

[Altschul et al., 1990] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410.

[Altschul et al., 1997] Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3392.

[Bailey and Elkan, 1994] Bailey, T. L. and Elkan, C. (1994). Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the 2nd International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 28–36.

[Bailey and Elkan, 1995] Bailey, T. L. and Elkan, C. (1995). Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning*, 21(1/2):51–80.

[Batzoglou et al., 2000] Batzoglou, S., Pachter, L., Mesirov, J. P., Berger, B., and Lander, E. S. (2000). Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Research*, 10(7):950–958.

[Blanchette et al., 2000] Blanchette, M., Schwikowski, B., and Tompa, M. (2000). An exact algorithm to identify motifs in orthologous sequences from multiple species. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 37–45.

[Brazma et al., 1998] Brazma, A., Jonassen, I., Eidhammer, I., and Gilbert, D. (1998). Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):279–305.

[Brejová et al., 2000] Brejová, B., DiMarco, C., Vinař, T., Hidalgo, S. R., Holguin, G., and Patten, C. (2000). Finding Patterns in Biological Sequences. Technical Report CS-2000-22, Dept. of Computer Science, University of Waterloo.

[Burkhardt et al., 1999] Burkhardt, S., Crauser, A., Ferragina, P., Lenhof, H.-P., Rivals, E., and Vingron, M. (1999). q-gram based database searching using a suffix array (QUASAR). In *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 77–83, Lyon, France.

[Chiang et al., 2001] Chiang, D. Y., Brown, P. O., and Eisen, M. B. (2001). Visualizing associations between genome sequences and gene expression data using genome-mean expression profiles. *Bioinformatics*, 17(S1):S49–S55.

[Delcher et al., 1999] Delcher, A. L., Kasif, S., Fleischmann, R. D., Peterson, J., White, O., and Salzberg, S. L. (1999). Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376.

[Dorohonceanu and Nevill-Manning, 2000] Dorohonceanu, B. and Nevill-Manning, C. G. (2000). Accelerating protein classification using suffix trees. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 128–133.

[Durbin et al., 1998] Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis*. Cambridge University Press.

[Eidhammer et al., 2000] Eidhammer, I., Jonassen, I., and Taylor, W. R. (2000). Structure comparison and structure patterns. *Journal of Computational Biology*, 7(5):685–716.

[Fickett and Hatzigeorgiou, 1997] Fickett, J. W. and Hatzigeorgiou, A. G. (1997). Eukaryotic promoter recognition. *Genome Research*, 7(9):861–868.

[Gelfand et al., 2000] Gelfand, M. S., Koonin, E. V., and Mironov, A. A. (2000). Prediction of transcription regulatory sites in Archaea by a comparative genomic approach. *Nucleic Acids Research*, 28(3):695–705.

[Gish, 2001] Gish, W. (2001). WU-Blast website. `http://blast.wustl.edu/`.

[Gomez et al., 2000] Gomez, M., Johnson, S., and Gennaro, M. L. (2000). Identification of secreted proteins of Mycobacterium tuberculosis by a bioinformatic approach. *Infection and Immunity*, 68(4):2323–2327.

[Gorodkin et al., 1997a] Gorodkin, J., Heyer, L. J., Brunak, S., and Stormo, G. D. (1997a). Displaying the information contents of structural RNA alignments: the structure logos. *Computer Applications in the Biosciences*, 13(6):583–586.

[Gorodkin et al., 1997b] Gorodkin, J., Heyer, L. J., and Stormo, G. D. (1997b). Finding the most significant common sequence and structure motifs in a set of RNA sequences. *Nucleic Acids Research*, 25(18):3724–3732.

[Grundy et al., 1997] Grundy, W. N., Bailey, T. L., Elkan, C. P., and Baker, M. E. (1997). Meta-MEME: motif-based hidden Markov models of protein families. *Computer Applications in the Biosciences*, 13(4):397–406.

[Gusfield, 1997] Gusfield, D. (1997). *Algorithms on strings, trees and sequences: computer science and computational biology*. Chapman & Hall, New York.

[Hardison et al., 1997] Hardison, R. C., Oeltjen, J., and Miller, W. (1997). Long human-mouse sequence alignments reveal novel regulatory elements: a reason to sequence the mouse genome. *Genome Research*, 7(10):959–966.

[Huang and Miller, 1991] Huang, X. and Miller, W. (1991). A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics*, 12(3):337–357. See SIM website `http://www.expasy.ch/tools/sim.html`.

[Hughes et al., 2000] Hughes, J. D., Estep, P. W., Tavazoie, S., and Church, G. M. (2000). Computational identification of cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. *Journal of Molecular Biology*, 296(5):1205–1214.

[Hughey and Krogh, 1996] Hughey, R. and Krogh, A. (1996). Hidden Markov models for sequence analysis: extension and analysis of the basic method. *Computer Applications in the Biosciences*, 12(2):95–107.

[Ison et al., 2000] Ison, J. C., Blades, M. J., Bleasby, A. J., Daniel, S. C., Parish, J. H., and Findlay, J. B. (2000). Key residues approach to the definition of protein families and analys is of sparse family signatures. *Proteins*, 40(2):330–331.

[Jonassen, 1996] Jonassen, I. (1996). Efficient discovery of conserved patterns using a pattern graph. Technical Report 118, Department of Informatics, University of Bergen, Norway.

[Krogh et al., 1994] Krogh, A., Brown, M., Mian, I. S., Sjolander, K., and Haussler, D. (1994). Hidden Markov models in computational biology. Applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1501.

[Kurtz and Schleiermacher, 1999] Kurtz, S. and Schleiermacher, C. (1999). REPuter: fast computation of maximal repeats in complete genomes. *Bioinformatics*, 15(5):426–427.

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

[Lawrence and Reilly, 1990] Lawrence, C. E. and Reilly, A. A. (1990). An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, 7(1):41–51.

[Li et al., 1999] Li, M., Ma, B., and Wang, L. (1999). Finding Similar Regions in Many Strings. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 473–482, Atlanta.

[Liang et al., 2000] Liang, C., Li, M., and Ma, B. (2000). COPIA: A New Software for Finding Consensus Patterns in Protein Sequences. To appear.

[Linial et al., 1997] Linial, M., Linial, N., Tishby, N., and Yona, G. (1997). Global self-organization of all known protein sequences reveals inherent biological signatures. *Journal of Molecular Biology*, 268(2):539–546.

[Lipman and Pearson, 1985] Lipman, D. J. and Pearson, W. R. (1985). Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441.

[Liu et al., 1995] Liu, J. S., Neuwald, A. F., and Lawrence, C. E. (1995). Bayesian Models for Multiple Local Sequence Alignment and Gibbs Sampling Strategies. *Journal of the American Statistical Association*, 90(432):1156–1170.

[Ma et al., 2000] Ma, B., Tromp, J., and Li, M. (2000). Super seed for faster and more sensitive homology search. Manuscript.

[Mironov et al., 1999] Mironov, A. A., Koonin, E. V., Roytberg, M. A., and Gelfand, M. S. (1999). Computer analysis of transcription regulatory patterns in completely sequenced bacterial genomes. *Nucleic Acids Research*, 27(14):2981–2989.

[Neuwald et al., 1997] Neuwald, A. F., Liu, J. S., Lipman, D. J., and Lawrence, C. E. (1997). Extracting protein alignment models from the sequence database. *Nucleic Acids Research*, 25(9):1665–1667.

[Nevill-Manning et al., 1998] Nevill-Manning, C. G., Wu, T. D., and Brutlag, D. L. (1998). Highly specific protein sequence motifs for genome analysis. *Proceedings of the National Academy of Sciences of the United States of America*, 95(11):5865–5871.

[Nicodème et al., 1999] Nicodème, P., Salvy, B., and Flajolet, P. (1999). Motif statistics. In Nesetril, J., editor, *Algorithms - ESA '99, 7th Annual European Symposium*, volume 1643 of *Lecture Notes in Computer Science*, pages 194–211, Prague. Springer.

[Parida et al., 2000] Parida, L., Rigoutsos, I., Floratos, A., Platt, D., and Gao, Y. (2000). Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 297–308.

[Pedersen et al., 1999] Pedersen, A. G., Baldi, P., Chauvin, Y., and Brunak, S. (1999). The biology of eukaryotic promoter prediction–a review. *Computers and Chemistry*, 23(3-4):191–207.

[Pesole et al., 2000] Pesole, G., Liuni, S., and D'Souza, M. (2000). PatSearch: a pattern matcher software that finds functional elements in nucleotide and protein sequences and assesses their statistical significance. *Bioinformatics*, 16(5):439–440.

[Pevzner, 2000] Pevzner, P. A. (2000). *Computational molecular biology: an algorithmic approach*. The MIT Press.

[Pevzner and Sze, 2000] Pevzner, P. A. and Sze, S. H. (2000). Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 269–278.

[Riechmann et al., 2000] Riechmann, J. L., Heard, J., Martin, G., Reuber, L., Jiang, C., Keddie, J., Adam, L., Pineda, O., Ratcliffe, O. J., Samaha, R. R., Creelman, R., Pilgrim, M., Broun, P., Zhang, J. Z., Ghandehari, D., Sherman, B. K., and Yu, G. (2000). Arabidopsis transcription factors: genome-wide comparative analysis among eukaryotes. *Science*, 290(5499):2105–2110.

[Rigoutsos and Floratos, 1998a] Rigoutsos, I. and Floratos, A. (1998a). Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm. *Bioinformatics*, 14(1):55–67. Published erratum appears in Bioinformatics, 14(2):229.

[Rigoutsos and Floratos, 1998b] Rigoutsos, I. and Floratos, A. (1998b). Motif discovery without alignment or enumeration (extended abstract). In *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 221 – 227, New York.

[Rigoutsos et al., 2000] Rigoutsos, I., Floratos, A., Parida, L., Gao, Y., and Platt, D. (2000). The emergence of pattern discovery techniques in computational biology. *Metabolic Engineering*, 2(3):159–167.

[Rocke and Tompa, 1998] Rocke, E. and Tompa, M. (1998). An algorithm for finding novel gapped motifs in DNA sequences. In Istrail, S., Pevzner, P., and Waterman, M., editors, *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 228–233, New York. ACM Press.

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

[Singh et al., 1998] Singh, M., Berger, B., Kim, P. S., Berger, J. M., and Cochran, A. G. (1998). Computational learning reveals coiled coil-like motifs in histidine kinase linker domains. *Proceedings of the National Academy of Sciences of the United States of America*, 95(6):2738–2743.

[Smith et al., 1990] Smith, H. O., Annau, T. M., and Chandrasegaran, S. (1990). Finding sequence motifs in groups of functionally related proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 87(2):826–830.

[Smith and Waterman, 1981] Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197.

[States and Agarwal, 1996] States, D. J. and Agarwal, P. (1996). Compact encoding strategies for DNA sequence similarity search. In *Proceedings of the 4th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 211–217. See SENSEI website `http://stateslab.wustl.edu/software/sensei/`.

[Tatusova and Madden, 1999] Tatusova, T. A. and Madden, T. L. (1999). BLAST 2 Sequences, a new tool for comparing protein and nucleotide sequences. *FEMS Microbiology Letters*, 174(2):247–250.

[Tompa, 1999] Tompa, M. (1999). An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 262–271.

[van Helden et al., 1998] van Helden, J., Andre, B., and Collado-Vides, J. (1998). Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *Journal of Molecular Biology*, 281(5):827–832.

[Yada et al., 1997] Yada, T., Totoki, Y., Ishii, T., and Nakai, K. (1997). Functional prediction of B. subtilis genes from their regulatory sequences. In *Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 354–357.

[Zhang, 1998] Zhang, M. Q. (1998). Statistical features of human exons and their flanking regions. *Human Molecular Genetics*, 7(5):919–922.

[Zhang et al., 2000] Zhang, Z., Schwartz, S., Wagner, L., and Miller, W. (2000). A greedy algorithm for aligning DNA sequences. *Journal of Computational Biology*, 7(1-2):203–204.