12

Motif Discovery with Compact Approaches - Design and Applications

Cinzia Pizzi
University of Padova
Italy

1. Introduction

In the post-genomic era, the ability to predict the behavior, the function, or the structure of biological entities (such as genes and proteins), as well as interactions among them, plays a fundamental role in the discovery of information to help biologists to explain biological mechanisms.
In this context, appropriate characterization of the structures under analysis, and the exploitation of combinatorial properties of sequences, are crucial steps towards the development of efficient algorithms and data structures to be able to perform the analysis of biological sequences.
Several functional and structural properties, and also evolutionary mechanisms, can be predicted either by the comparison of new elements with already classified elements, or by the comparison elements with a similar structure of function to infer the common mechanism that is at the basis of the observed similar behavior. Such elements are commonly called *motifs*.
Comparison-based methods for sequence analysis find their application in several biological contexts, such as extraction of transcription factor binding sites, identification of structural and functional similarities in proteins, and phylogeny reconstruction. Therefore, the development of adequate methodologies for motif discovery is of undoubt interests for several different fields in computational biology.
In motif discovery in biosequences, it is common to assume that statistically significant candidates are those that are likely to hide some biologically significant property. For this purpose all the possible candidates are ranked according to some statistics on words. Then they are presented in output for further inspection that need to be carried out by a biologist, who identifies the most promising patterns. These, in turn, are tested in laboratory to confirm their biological significance. Therefore, when designing algorithms for motif discovery, besides obviously aim at time and space efficiency, particular attention should be devoted to the output representation. In fact, even considering fixed length strings, the size of the candidate set becomes exponential if exhaustive enumeration is applied. This is already true when only exact matches are considered as candidate occurrences, and worsen when the intrinsic variability of biological sequences is taken into account.
Alternatively to methods based on exhaustive enumeration, heuristics could be used. However, heuristics cannot guarantee to find the optimal solution. Therefore some degree

of uncertainty remains whether motifs, that are statistically as significant as those reported in output, have been left out.
Computational power of nowadays computers can partially reduce the effects of exhaustive enumeration approaches, in particular for short length candidates. However, if the size of the output is too big to be analyzed by human inspection the risk is to provide biologists with very fast tools that produce mostly useless output.
A possible solution to these problems relies on compact approaches. Compact approaches are based on the partition of the search space into classes.
The final user can then be presented with an output that has the size of the partition, rather than the size of the candidate space, with obvious advantages for the human-based analysis that follows the computer-based filtering of the pattern discovery algorithms.
Compact approaches find applications both in searching and discovery frameworks. They can also be applied to several motif models: exact patterns, approximate patterns, position matrices, etc. And under both independent and identically distribution (i.i.d.) and Markov distributions.
The purpose of this chapter is to describe the basis of compact approaches, to provide the readers with the conceptual tools for applying compact approaches to the design of their algorithm for biosequence analysis. This will be achieved by overwieving examples of compact approaches that have been successfully developed for several motif models that will be illustrated with the sustain of examples and experiments to discuss their power.

2. Background

The methodologies to study the Science of Life dramatically changed during the past years. The advent of the web made it possible for the scientific community to share the massive quantity of data produced by high throughput techniques, thus accelerating the analysis of the available data and the discovery of related properties and associations. The development of high throughput technologies has as a consequence not only an increase in the amount of data, but also a diversification of the type of data available, opening new perspectives of investigations. Disciplines such as Bioinformatics and Computational Biology try to combine the efforts and competency of the communities of biologists and computer scientists in a single more powerful combination of human knowledge and efficiency, thanks to automatic approaches to data analysis.

2.1 Definition of the problem

One of the key aspects in the analysis of biological sequences is the identification of interesting patterns. "Interestingness" is a wide concept that may embrace very different definitions depending on the contexts in which the analysis is carried out. At an higher level we can define an interesting pattern as a pattern that shows an unusual behavior from what it is expected in terms of presence within the sequence under analysis.
More in details, searching for shared or over-represented patterns is motivated by a simple commonly accepted principle: if two or more sequences perform the same functions or have the same structure, then the common elements among the sequences might be somehow responsible for the observed similarity.
The problem of finding biologically significant patterns is then moved to the problem of finding statistically significant patterns.

solid word
wildcards-mismatches
insertion-deletion
generalized patterns
alignments
position weight matrices
hidden Markov models

Table 1. Various choices to model motifs in biological sequences. Starting from solid words the level of sensibility increases (allowing for variations), but the level of specificity consequently decreases, thus making more difficult the process of detection of the signal.

The problem of searching for similar regions among biological sequences faces several issues. First of all, the genetic code must be fault-tolerant to deal with errors that may occur during transcription or are due to random mutations, so that an intrinsic variability characterizes biological motifs. This variability has as a consequence the possible explosion of the size of the search space under study, due to the rich underlying combinatorics. Searching the whole pattern space then is feasible only for very short patterns. Note that this is also true for exact words, because their number increases exponentially with the length. On the other hand, heuristics are not guaranteed to find a globally optimal solution.
A critical step of the process is the choice of an appropriate structure to model the motifs. In some cases, deterministic patterns do not have enough expressive power to describe the specificity of the contributions of each symbol in any position of the site. Statistical matrices or graph-based models might offer a better framework in these cases. Several options have been considered during the past decades to model signals in biosequences, and to take into account for this intrinsic variability. Some of these models of choice are listed in Table 1, sorted in increasing order of expressive power (and consequent increase of difficulty in the design of related algorithm, and of their intrinsic complexity).
The choice of an appropriate model to describe motifs is a trade-off between the expressiveness of the model to describe particular biological properties, and the efficiency of the algorithms that can be applied when that model is chosen.
The scoring function chosen to evaluate the output also plays an important role in the identification of the searched sites. However, simple statistics are often unable to discriminate interesting motifs from motifs that are likely to occur by chance, so that different measures of statistical significance need to be considered. In summary, for a given choice of a statistical measure S of a motif m, one could ask three questions:

- What is the value of $S(m)$?
- How surprising is to measure $S(m)$ with respect to the value that was expected according to some background distribution?
- How likely is it for the recorded values to occur by chance?

These three questions can be answered by the means of different computations. The first one, for example, can be answered by exact counts or estimates. To answer the second, we need some score that measures *over-representation*, such as the *z-score*. Finally the third one is solved by resort to so called *p-value* of a statistic.
Once the model and the scoring function have been fixed, the next step is the development of efficient approaches to extract the searched information. There is a vast literature about

algorithms for motif discovery. However, most of them either have been developed to solve very specific instances of the problem, such as the *Motif Challenge Problem* (Pevzner & Sze, 2000), or are based on exhaustive search of the pattern space (among which (Queen et al., 1982; Staden, 1989; Tompa, 1999; van Helden et al., 1998; Waterman et al., 1984)), or rely on heuristics (for example (Hertz & Stormo, 1999; Stormo & III, 1989)). Comparison of several techniques in fact showed how performances substantially depend on the underlying model of the motif to discover (Tompa et al., 2005).

3. Compact scoring

Despite the intrinsic possible explosion of the size of the search space, it is possible to conceive a compact representation of the patterns such that only *representative* patterns are scored, and no critical information is lost in the process.

The classes must be designed in such a way that the score used to rank the candidates has a monotone behavior within each class. This allows the identification of a representative of each class, which is the element with the highest score. Consequently, it suffices to compute, and to report in output, only the score for the representatives. In fact, we are guaranteed that for each element that has not been ranked there is another one (the representative of the class it belongs to) that is at least as significant.

In such a framework, the output size would depend upon the number of classes in which patterns can be grouped, rather than on all the existing patterns that belong to the search space. This approach can also be used as a filter to detect unusual classes of strings that need to be scrutinized further.

The compact scoring needs two important steps to be carried out with critical attention:

1. definition of the search space
2. efficient partitioning

The definition of the search space clearly depends implicitly on the motif model, as discussed above. Nevertheless there are also two working frameworks in which one can pose his search (Brazma et al., 1998).

In the pattern-driven framework the search space consists of all possible patterns (of a given size) that can be generated over a given alphabet Σ. The input sequence is then tested for occurrences of each and every motif in a family of *a priori* generated, abstract *models* (for example (Keich & Pevzner, 2002)). Although more correct in principle, this method may pose severe computational issues.

In the sequence-driven framework the search space consists of strings that actually occur at least once in the given input, or to some more or less controlled neighborhood of those substrings (for example (Lawrence et al., 1993)). This may be less firm methodologically, but leads to time and space savings.

The choice of techniques and data structures that can be used to perform the partition are strictly related to the definition of the model and search space. In turn they affect the space complexity reduction that can be achieved by the partitioning, as it will be clear in the following discussion.

4. Solid words

In this section we describe a methodology for compact representation and scoring of single solid words, and pairs of solid words. We will outline the basic concepts used in the approach. Details of formulas and proofs of theorems can be found in (Apostolico et al., 2003).

4.1 Compact indexes for single words

Let us consider the problem of extracting frequent substrings from a string x of length n. The number of substrings in x is equal to the number of possible choices of starting and ending indexes that univocally identify the substring. These are obviously $O(n^2)$. There are several observations that can be done with respect to this output size:

- for very long strings (consider genome wide analysis) the computation of all the occurrences of all these substrings might become prohibitive in terms of both time and space needed;
- the list of candidates might be too long to allow deep inspection by the final user;
- some substrings, of increasing length, occur the same number of times and in the same exact position: the output set might hide some notable redundancy.

In order to overcome these issues a compact approach might be applied. The compact representation and scoring of solid words can be achieved by exploiting the characteristics of appropriate data structures, such as the *suffix tree*, and by an in-depth analysis of the properties of monotonicity for some measure of over- or under-representation.

4.1.1 Suffix trees

A suffix tree is a data structure used to efficiently store and retrieve information about all the substrings of a given text. Given a text x of length n defined over an alphabet Σ, and a symbol $\$ \notin \Sigma$, the suffix tree associated with x is the digital search trie of all the suffixes of x. There are two versions of suffix tree: the expandend suffix tree (also called suffix trie), and the compact suffix tree. In the expanded version each arc is labelled with a symbol, except for the leaves that are labelled with the corresponding suffix. The space required to store an expanded suffix tree is $O(n^2)$ in the worst case (Aho et al., 1974). On the other hand, in the compact representation chains of nodes with one outgoing edge are collapsed together in a single arc. The symbol \$ guarantees that each node in the compact suffix tree (except the leaf nodes) is branching. This property, together with the observation that there are n leaves, corresponding to the n suffixes, implies that there are $O(n)$ nodes in the suffix tree. Each arc is labeled with two indexes, the start and the end positions of the corresponding substring in the text (or, equivalently, the start position and the length of the path from the root node to the node at which the arc ends). With such a representation, the overall space needed to store a compact suffix tree is $O(n)$. The brute force construction of a suffix tree requires $O(n^2)$ time. However, more clever algorithms allow for linear time construction of the tree (McCreight, 1976; Ukkonen, 1995; Weiner, 1973). The word spelled by a path from the root to a node α is indicated with $w(\alpha)$, and α is called the *proper locus* of $w(\alpha)$. The *locus* of a word w is the unique node β of the suffix tree such that w is a proper prefix of β, and $father(\beta)$ is a proper prefix of w. The frequency of a word w can be obtained in time proportional to the length of w and to the number of its occurrences. For this purpose we reach the locus β of

w and then visit the subtree rooted at β to count the number of its leaves. Alternatively, one could annotate the nodes of the tree in a bottom-up fashion with the count of their children. Then, to know the frequency of any query word w, only the time to reach its locus is needed. Alternatively to the suffix tree, one could use a suffix array (Manber & Myers, 1993). As it is shown in (Abouelhoda et al., 2004) these two data structures are equivalent. The advantage of the suffix array is that it usually takes less space. This comes at expenses of an increase of the difficulty in the designing of related algorithms.

4.1.2 Example

For a string $x = AGCTAGCTAAA$ of length 11, the number of substrings is $\frac{n(n+1)}{2} = 66$ (number of choices of starting and ending position for a substring). In the specific case of the string in our example, if we remove duplications there will remain 52 different substrings. By building the suffix tree (see Fig.1), and considering only the strings corresponding to nodes the number of candidates reduces to 15.

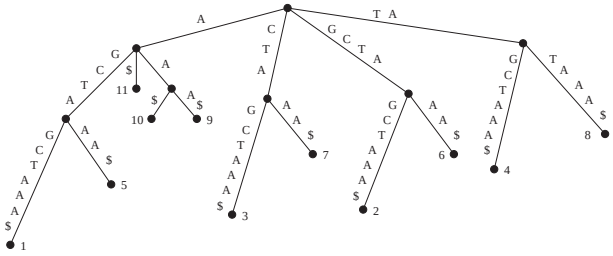


Fig. 1. The suffix tree for the string $x = AGCTAGCTAAA$. The numbering of the leaves indicates the corresponding suffix in x, i.e. their starting position.

4.1.3 Compact representation for the frequency of single words

An interesting property of the suffix tree is that all the words that end within an arc have the same frequency, equal to the frequency of the word corresponding to their locus. Thus a suffix tree partitions the $\Theta(n^2)$ subwords of a text in $O(n)$ classes, such that the words that belong to the same class have the same locus. Take, for example, the strings "C", "CT", and "CTA" in the suffix tree of Fig.1. They all occur at positions 3 and 7 in x, as it can be easily verified by eye inspection of the string. Therefore they start at the same positions in the text, and have the same frequency.

The word $w(\beta)$, corresponding to the locus β of the class C_β can be considered as *maximal in length*, since any extension of $w(\beta)$ will have a different frequency, and thus would not belong to the class C_β. These words are called *representatives*.

In Table 2 we enumerate all the classes identified by the partition induced by the suffix tree of Fig.1.

The compact output related to the frequency analysis of the substrings of x is represented by the following Table 3. It is easy to note by eye inspection that this output gives a much clearer and immediate representation of the frequency distribution of substrings of x with respect to a full enumeration of all 66 possible substrings in it.

$\{\mathbf{A}\} = [1,5,9,10,11]$
$\{AG, AGC, AGCT, \mathbf{AGCTA}\} = [1,5]$
$\{AGTCAG, AGTCAGC, AGTCAGCT, AGTCAGCTA, AGTCAGCTAA, \mathbf{AGTCAGCTAAA}\} = [1]$
$\{AGTCAA, \mathbf{AGTCAAA}\} = [5]$
$\{\mathbf{AA}\} = [9,10]$
$\{\mathbf{AAA}\} = [9]$
$\{C, CT, \mathbf{CTA}\} = [3,7]$
$\{CTAG, CTAGC, CTAGCT, CTAGCTA, CTAGCTAA, \mathbf{CTAGCTAAA}\} = [3]$
$\{CTAA, \mathbf{CTAAA}\} = [7]$
$\{G, GC, GCT, \mathbf{GCTA}\} = [2,6]$
$\{GCTAG, GCTAGC, GCTAGCT, GCTAGCTA, GCTAGCTAA, \mathbf{GCTAGCTAAA}\} = [2]$
$\{GCTAA, \mathbf{GCTAAA}\} = [6]$
$\{T, \mathbf{TA}\} = [4,8]$
$\{TAG, TAGC, TAGCT, TAGCTA, \mathbf{TAGCTAAA}\} = [4]$
$\{TAT, TATA, TATAA, \mathbf{TATAAA}\} = [8]$

Table 2. The maximal partition that is obtained with a suffix tree for the string $x = AGCTAGCTAAA$. Each row enumerates the strings of each class, and their occurring positions. The representative of each class is in bold. The set of starting positions for all the strings in the class are listed within square brackets.

A	5
$AGCTA$	2
CTA	2
AA	2
$GCTA$	2
TA	2
$AGTCAGCTAAA$	1
$AGTCAAA$	1
AAA	1
$CTAGCTAAA$	1
$CTAAA$	1
$GCTAGCTAAA$	1
$GCTAAA$	1
$TAGCTAAA$	1
$TATAAA$	1

Table 3. The compact output for the frequency count of the substrings in x.

4.1.4 Compact representation for single words statistics

In some kind of analysis the frequency count might be insufficient to extract "interesting" patterns, since some of them might occur often simply by chance. In these cases an evaluation of over- or under- representation can give a better solution to the problem of the extraction of interesting patterns. It has been shown in (Apostolico et al., 2003) that the compact approach can be extended to over-representation scores, such as z-scores. Z-scores have the characteristic to compare the counted frequency with the frequency expected assuming a given background distribution. Some examples are $\frac{f}{e}$, $f - e$, $\frac{f-e}{e}$, where f and e are the counted and expected frequency respectively.

The partition induced by the suffix tree is such that the words that belong to a class are prefixes of increasing length of the representative substring. The expected frequency of words decreases with word length. We show this with a simple example. Consider the case of i.i.d. hypothesis, with an uniform distribution, i.e. all the symbols in the alphabet have the same

probability p to occur. The probability of a word of length l is then p^l. Since $0 < p < 1$, we have that the term p^l decreases with l. The reasoning can be extended and proved for a general distribution and under both i.i.d. and Markov chain hypothesis.
Within each class two conditions hold:

1. the counted frequency is constant;
2. the expected frequency is monotonically decreasing and reach its minimum at the representative.

Hence z-scores with a constant frequency f and decreasing frequency e will have a monotonically increasing behavior within each class, and reach their maximum in correspondence of the representative of the class. Using over-representation to extract interesting patterns allows to use a threshold to filtering the results further. The threshold can be arbitrarily fixed (for example we are interested in patterns that occur at least the double of what we expect), or by fixing a p-value (that it is a probability of seeing a pattern by chance) and computing the corresponding absolute threshold for the score (Staden, 1989).
In Table 4 we can see an example of compact output of the z-score $\frac{f}{e}$, for a uniform distribution and for a general distribution. We still refer to the string x of the previous examples. It can be seen that the two background distribution produce a slightly different order in the output, and a quite different score associated to the strings. By setting a threshold $T_h = 1000$ we obtain a further reduction of the output size that in the case of the uniform distribution maintains 9 strings, while in the case of the considered general distribution it maintains 12 strings.

Substring	Score (uniform)	Substring	Score (general)
AGTCAGCTAAA	4194304	*AGTCAGCTAAA*	694444444
GCTAGCTAAA	1048574	*GCTAGCTAAA*	69444444
CTAGCTAAA	262144	*CTAGCTAAA*	41666667
TAGCTAAA	65536	*TAGCTAAA*	4166667
AGTCAAA	16384	*AGTCAAA*	833333
GCTAAA	4096	*TATAAA*	250000
TATAAA	4096	*GCTAAA*	83333
AGCTA	2048	*CTAAA*	50000
CTAAA	2048	*AGCTA*	16667
GCTA	512	*GCTA*	1667
CTA	128	*CTA*	1000
AAA	64	*AAA*	1000
AA	32	*AA*	200
TA	32	*TA*	100
A	20	*A*	50

Table 4. The compact output for the over-representation score of the substrings in x. In the first two columns the substrings are scored according to a uniform distribution. In the last two columns the substrings are scored according to the following distribution:
$p_a = 0.1, p_c = 0.1, p_g = 0.6, p_t = 0.2$

4.2 Compact indexes for co-occurrences

The suffix tree property of partitioning the set of substrings of a string in $O(n)$ classes can be exploited also for the computation of co-occurrences between substrings of an input string.
In (Apostolico et al., 2004) the compact approach was extended to the efficient computation of a table to hold the number of co-occurrences of substrings within a text. In this context

the problem is: given two words y and z, and a distance d, compute the number of times that z follows y at a distance at most d. In (Apostolico et al., 2004) further restrictions were set so that z must follow an occurrence of y but it must occur before the next occurrences of y. Moreover, if many occurrences of z are found at the right side of the same occurrence of y, only the closest one is counted. In Fig.2 it can be seen how the co-occurrence count can vary. A simple count, without restriction would give a value of 4: $\{(p_i, q_k), (p_i, q_{k+1}), (p_i, q_{k+2}), (p_{i+1}, q_{k+2})\}$. If no interleaving occurrences of y are allowed the count would reduce to 3: $\{(p_i, q_k), (p_i, q_{k+1}), (p_{i+1}, q_{k+2})\}$. If also no interleaving occurrences of z are allowed the count would drop to 2: $\{(p_i, q_k), (p_{i+1}, q_{k+2})\}$.

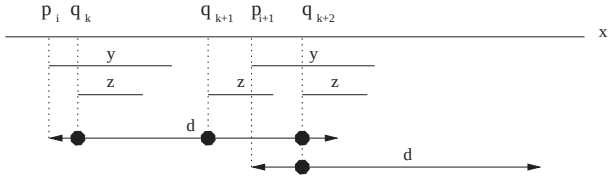


Fig. 2. Illustration of co-occurrences between two substrings y and z.

A naive approach would count the number of co-occurrences between all $O(n^2)$ substrings of x, thus requiring $O(n^4)$ time and space. In contexts other than bioinformatics some algorithms have been developed, on related, even though more general problems (Arimura et al., 2000; Wang et al., 1994). However, they can also solve the given problem in time $O(d^2n^3 \log n)$ and $O(n^3)$ respectively. Here we focus on the description of the algorithm in (Apostolico et al., 2004) that solves the problem of computing the simple frequency. Eliminating multiple occurrences in fact comes at no extra cost but also does not change the space complexity, that is the focus of compact approaches.

The algorithm exploits the suffix tree property described in Sec. 4.1.3 to partition the $O(n^2)$ substrings in an $O(n)$ classes corresponding to the nodes, and computes the co-occurrence count only for words corresponding to node pairs. The key observation is that if a pair (y', z') is left out, then the following conditions hold:

1. there exists already a corresponding pair (y, z) such that y' and z' are prefixes of y and z respectively;
2. the score of (y, z) is at least equal to the score of (y', z').

These facts follow from the property of the suffix tree that all the substrings ending within an arc have the same starting positions of the string corresponding to their locus. Since the distance d is measured from the beginning of the first component, this implies that classes of substrings that share the same set of starting positions will occur, within distance d, the same number of times. Again, the strings corresponding to the proper loci of the suffix tree can be selected as representative of the classes, and indexed. In Fig.3 the pair (ACG, T) co-occurs the same number of times of the pair $(ACGTA, TA)$.

There are $O(n)$ nodes in the suffix tree, and each of them can be chosen as first component. A second suffix tree is used to store the number of co-occurrences between the fixed first component, and all other nodes in the suffix tree. The annotation is made as follows:

1. assign a null weight to all nodes of the suffix tree used for the counting of co-occurrences
2. let y be the first fixed component, and $\{p_1 \dots p_k\}$ its set of occurrences;

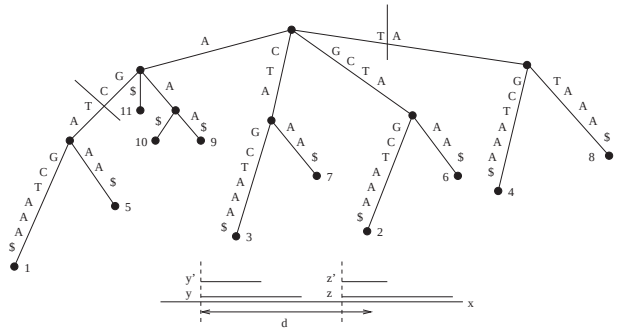


Fig. 3. Illustration of co-occurrences classes.

3. add 1 to all the leaves that correspond to positions $\{p_1, p_1 + 1, \ldots p_1 + d\}$, $\{p_2, p_2 + 1, \ldots p_2 + d\}, \ldots \{p_k, p_k + 1, \ldots p_k + d\}$;
4. annotate the tree bottom-up, so that internal nodes have a weight corresponding to the sum of the weights of their children.

After the annotation the suffix tree will hold the number of co-occurrences between the fixed y and any string z with a proper locus in the tree. The annotation of the tree is performed in linear time. Since this must be repeated for every possible choice of the first component y, the resulting complexity is $O(n^2)$.

4.2.1 Example

Turning back at the example in the previous section, the co-occurrences will be counted only for $15 \times 15 = 225$ pairs of substrings of x. If an exhaustive index would have been build instead, the number of entries of the output would have been 4356, or 2704 if duplication was removed. In the former case we achieved a 95% reduction of the table size, in the latter a 92% reduction.

5. Compact approaches for words with mismatches

Co-occurrence counts are useful for the detection of motifs that are particularly conserved at the sides and allow for high variability in the middle, i.e. so called dyads. However, other distributions of variability are possible and compact approaches to deal with them have also been developed. However, dealing directly with variability implies an higher order of complexity in the solution of the problem. It is possible to identify in literature two main frameworks in which compact approaches have been developed:

1. the candidate motifs occur at least once exactly in the input string; the errors (mismatches) can occur in any position of the motifs;
2. the candidate motifs might never occur in the input string exactly; the position of the errors (wildcards) are fixed in the motif template.

5.1 Compact approaches for motif with mismatches

In (Apostolico & Pizzi, 2007) a compact approach for the extraction of motifs with mismatches, with an *i.i.d.* background, was proposed. The approach was next extended to deal with a

markovian background distribution (Pizzi & Bianco, 2009). The assumptions are that: the motif must occur at least once exactly in the sequences; its instances can appear with exactly, or at most, k mismatches; the mismatches can occur in any position of the motif.
We first observe that, because of the introduction of mismatches, it is not possible to use a suffix tree to partition the search space. In fact if we consider a string w with exactly k mismatches, and its extension wa, with $a \in \Sigma$, with exactly k mismatches we are not guaranteed that the frequency of wa is the same of w, even if both have the same locus. In fact, in the count of wa with k mismatches we would add the number of occurrences of w with k mismatches followed by an a, and the number of occurrences of w with $k-1$ mismatches followed by a symbol $b \neq a$, and finally subtract the number of occurrences of w with k mismatches followed by a symbol $b \neq a$. Depending on how the symbols occur in the text the number of occurrences of wa might be less, equal, or greater than those of w.
Hence, to compact the output, one has to isolate intervals of words of increasing length with the same frequency (with mismatches. However, this is not sufficient. The next step is to verify that within those intervals the expectation with mismatches has indeed a monotone behavior.
In (Apostolico & Pizzi, 2007; 2008), besides the proposal of polynomial algorithms for the computation of the expected frequency, a full study about the score behavior of motifs with mismatches has been presented. We redirect the readers to the original papers for proofs and report here only the main results.

1. the word length is increased, and the number of mismatches is fixed: the expected frequency with mismatches decreases;
2. the number of mismatches is increased, keeping the word length fixed, counting an exact number of mismatches: the expected frequency with mismatches increases (at least until a number of mismatches equal to half of the length of the motif, then it might decrease);
3. the number of mismatches is increased, keeping the word length fixed, counting at most a given number of mismatches: the expected frequency with mismatches increases.

The case 1) is that of interest for the intervals of increasing length and constant counted frequency with mismatches. In fact, in these intervals we have that the z-scores are monotone, and we can take as a representative of the interval the corresponding string.
Experiments to measure the effectiveness of the interval definition can be carried out in the following way. Let us consider words of length $m \pm \delta$, for a given m and δ and compute the score for runs of words with the same frequency (with mismatches) only once.
Tables 5 and 6 report the percentage of entry savings, with respect to a full enumeration, when the frequency is counted for exactly or at most k mismatches respectively. The input string was a sample 10k bases from the yeast genome.

5.2 Compact approaches for words with wildcards

In case of regular patterns the words are defined over an alphabet $\Sigma' = \Sigma \cup \{"."\}$ that includes also the wildcard "." symbol. The classes of equivalence are identified by the subwords that are generated by the corresponding grammars. Moreover, the property of maximality must hold both in length and in composition. A pattern is maximal in composition if the substitution of any of its wildcards with a given symbol of the alphabet Σ alters the number of occurrences of the pattern. Maximal regular patterns have been used to devise combinatorial algorithms

k		m =12	m =13	m =14	m =15	m =16	m =17	m = 18
0	avg length	6.74	6.91	6.96	6.98	6.98	6.98	6.99
	% saving	84.21	85.17	85.49	85.58	85.63	85.65	85.67
1	avg length	2.33	2.35	2.38	2.54	2.74	2.92	2.97
	% saving	17.71	23.04	26.51	29.41	36.41	45.07	50.37
2	avg length	2.30	2.32	2.35	2.37	2.40	2.48	2.61
	% saving	9.46	13.14	18.36	23.43	26.29	28.26	31.67
3	avg length	2.14	2.21	2.28	2.31	2.36	2.40	2.44
	% saving	4.85	7.29	9.98	13.75	18.79	23.53	27.34
4	avg length	2.02	2.07	2.15	2.23	2.28	2.31	2.34
	% saving	1.01	2.91	5.39	7.84	10.69	14.70	19.50

Table 5. Average run length and table size reductions for frequency with exactly k mismatches ($\delta = 3$).

k		m =12	m =13	m =14	m =15	m =16	m =17	m =18
0	avg length	6.74	6.91	6.96	6.98	6.98	6.99	6.99
	% saving	84.20	85.17	85.49	85.59	85.63	85.65	85.67
1	avg length	5.02	5.86	6.50	6.82	6.93	6.97	6.98
	% saving	68.24	77.92	82.75	84.65	85.33	85.53	85.60
2	avg length	3.29	3.96	4.72	5.55	6.27	6.70	6.88
	% saving	37.84	51.70	65.09	75.47	81.32	83.98	85.04
3	avg length	2.32	2.68	3.21	3.84	4.57	5.39	6.19
	% saving	12.43	23.74	36.76	50.53	63.84	74.31	80.65
4	avg length	2.03	2.14	2.34	2.69	3.20	3.83	4.55
	% saving	1.05	4.80	12.61	23.75	36.73	50.44	63.46

Table 6. Average run length and table size reductions for frequency with at most k mismatches ($\delta = 3$).

for the detection of frequent patterns in biosequences (Califano, 2000; Rigoutsos & Floratos, 1998). In both works, the extracted patterns can be said to be a compact representation for motifs in which variability is allowed, but only at specific positions.

Although the number of maximal motifs with wildcard can be exponential, in (Parida et al., 2000) the authors present a way of extracting the inner structure that characterize such type of motif so that the output size could be further reduced. This can be obtained by defining a basis of motifs. A basis B is a subset of all the motifs from which it is possible to recover all the other motifs. More in details a motif is characterized by its list of occurrences, and all motifs not in the basis can be obtained by a combination of the location lists of some of the motifs in B. The motifs that belong to the basis are called *irredundant*. Several works have been done on such motif representation, among which (Apostolico & Parida, 2004; Pelfrêne et al., 2003; Ukkonen, 2009). In (Apostolico, Parida & Rombo, 2008) the concept of irredundant basis was further extended to 2D patterns.

In the context of patterns with wildcards, a pattern is taken into consideration if it occurs for a number of time that is defined by a quorum $q \geq 2$. In (Pisanti et al., 2005) an in dept study about the complexity of the number of irredundant motif showed that there exists a family of motifs for which the number of motifs in the basis is $\Omega(n^2)$ for $q = 2$. In the same paper the authors propose a new definition for a basis, that is stronger than that in (Parida et al., 2000). Therefore their basis is smaller and included in the previous one. Motifs that belong to the

new definition are called *tiling*. For basis on tiling motifs the number of elements is linear in the size of the input string for $q = 2$. However, it has been proved that for all basis there is an exponential dependency in the quorum when $q > 2$, so that no polynomial algorithm exists to extract a basis in this case.
Basis are clearly a compact representation of the motifs space, hence a powerful tool for the analysis of biological sequences when variability is taken into account in the form of wildcards.

6. Position weight matrices

Positions weight matrices are one of the most widely used models for biological signals, being used both in genomic and proteomic studies.
A position weight matrix is a scoring matrix M, where each row represents a position and each column a symbol from the alphabet Σ (in literature it is common to find also the vice versa). The score of the matrix against a segment $x_j x_{j+1} \dots x_{j+m-1}$ of a sequence $x = x_1 x_2 \dots x_n$, is given by:

$$S(M, j) = \sum_{i=0}^{m-1} M[i][x_{j+i}]$$

Given a threshold T, the matrix M is said to have an hit at position j of x if $S(M, j) \geq T$.
The threshold can be given as an absolute value or as a p-value or MSS score T', and then converted in terms of absolute score with respect to the matrix M (Staden, 1989).
The hits of a matrix M in the sequence x can be found naively by an $O(mn)$ algorithm that for each position j check if $S(M, j)$ is above the threshold. The computation of the score takes $O(m)$ steps, and need to be computed for $n - m + 1$ possible starting positions, hence the claimed complexity.

6.1 The look-ahead technique

The look-ahead technique (Wu et al., 2000) can be used to stop the computation of the score whenever one is sure that the threshold will never be reached. Let $S_k(M, j) = \sum_{i=k+1}^{m-1} M[i][x_{j+i}]$ be the score of the segment $x_j \dots x_{j+k-1}$ with respect to the matrix M, the condition to stop the comparison is:

$$S_k(M, j) + \sum_{i=k+1}^{m-1} \max_{s \in \Sigma} M[i][s] < T$$

i.e. even if in the rest of the comparison we will add the maximum score of the matrix for those positions, the final score will be below the threshold.
It is easy to compute an array t_{la} of size m that holds the value of the partial thresholds that need to be reached in order to have the chance to reach the threshold. The entries of the array are given by:

$$t_{la}[i] = T - S_k(M, i + 1)$$

These can be computed in linear time in m, by setting $t_{la}[m - 1] = T$, and recursively computing the other entries by the formula:

$$t_{la}[i] = t_{la}[i+1] - \max_{s \in \Sigma} M[i+1][s], \text{ for } i = m-2 \ldots 0$$

6.2 The Minimum Gain

A dual concept to that of look-ahead score is given by the *Minimum Gain* (Pizzi et al., 2011). The minimum gain relative to a position j in the matrix is the minimum score that can be obtained by summing the scores from positions $j+1$ to m.

The minimum gain can be used similarly to look-ahead to stop the comparison of the matrix against a segment earlier. In this case the condition to be verified is:

$$S_k(M, j) + \sum_{i=k+1}^{m-1} \min_{s \in \Sigma} M[i][s] \geq T$$

If this is the case, then no matter what the following $m-k$ symbols are, there will be a hit to report at position j.

Similarly to look-ahead, we can build a minimum gain array t_{mg} corresponding to the partial thresholds that need to be reached to ensure that the hit. Starting from $t_{mg}[m-1] = 0$, we have:

$$t_{mg}[i] = t_{mg}[i+1] - \min_{s \in \Sigma} M[i+1][s], \text{ for } i = m-1 \ldots 0$$

6.3 Compact approach to automaton construction

In (Pizzi et al., 2011) an algorithm with optimal $O(n)$ searching time was proposed to solve the problem of profile matching (i.e. given a string x, a threshold T and a matrix M, find all the hits of M in x).

This algorithm is based on the classic multi-pattern matching algorithm based on the Aho-Corasick automaton (Aho et al., 1974). In summary, an automaton is built that contains all the words that are a match for the given threshold and matrix.

The look-ahead technique can be used in this context to generate all and only the words of length m that are hits. The words are generated directly in a trie, that will later be annotated with failure links to build the Aho-Corasick automaton. Starting from the empty trie, only symbols which score is above $t_{la}[0]$ are expanded. Each node will take track of the partial score of the path from the root to the node itself, and will recursively expand further levels in a similar way with comparison with the look-ahead partial thresholds.

Depending on the given threshold, and on the distribution of the score within the matrix, the size of the trie is very variable, but can become prohibitive. In fact by decreasing the threshold (e.g. increasing the p-value), the number of words that are hits for the matrix increases, and can possibly reach an exponential number (the worst case given by $|\Sigma|^m$).

The minimum gain can then be used to substantially reduce the size of the automaton, implementing the compact approach philosophy. The idea is to limit the length of the words that are hits for the matrix whenever a prefix of the word is enough to establish that there will be a hit. While building the trie this means that we do not need to build the subtree of this prefix. The cost of the further comparison with t_{mg} at construction time is irrelevant compared to the time saved by avoiding to build the subtree. Moreover, there could be considerable space savings. See, for example what happens when we consider matrix M00003 from the

Jaspar database (Sandelin et al., 2004), and the threshold score , corresponding to MSS=0,85 (see Table 7). The entries of the matrix have been already multiplied by the factors needed to compute the score relative to MSS (Quandt et al., 1995).
The matrix entries have been sorted in ascending order, so each table entry on the first four columns contains a pair (symbol,score), and the last two columns hold the look-ahead and the minimum gain score, respectively.

0	1	2	3	LA	MG
A,0	C,0	T,0	G,18500	8799	53914
A,0	T,0	G,0	C,18500	27279	53914
A,0	T,0	G,0	C,18500	45779	53914
A,110	T,230	G,230	C,355	46134	54024
A,95	T,240	C,285	G,305	46439	54119
T,114	C,264	A,330	G,402	46841	54233
T,176	C,480	A,848	G,1456	48297	54409
T,494	C,608	A,722	G,5266	53503	54903
A,180	T,380	C,1560	C,1580	55083	55083

Table 7. Computation of scores for the matrix M00003 of the Jaspar database.

When we build the trie using the look-ahead technique, we have that at the first level only the symbol *G* has a score that is above the partial threshold $t_{la}[0]$. Hence we will have only one child node for the root, with partial score 18500. At the second level we have again that only the score of one symbol, (C,18500) summed up with the current partial threshold 18500, will give a score (37000) above the partial threshold 27279. Again we add a single child to the previous node, and label the edge with *C*. At the third step only *C* have a score that summed up with the path partial score (37000) will give a value (55500) that is above $t_{la}[2] = 45779$. The trie is then extended a further level using only symbol *C*. From now on we can notice that the partial threshold of this path is already above any following look-ahead partial thresholds. This means that all symbols will be considered at each level, thus obtaining a full subtree of high 6. This implicity means that the prefix GCC is enough to establish whether there is an hit or not. If when building the trie we compare the value of the path partial threshold with the minimum gain partial threshold we can stop early the computation, and set as the final state of the Aho-Corasick automaton the current node.
The minimum gain basically defines classes of equivalence within the space of hits of the matrix. Hits that share the same prefix do not need to be totally inserted in the trie. It suffices to insert their common representative prefix.
In case of matrix M00003, we will save the construction of $\sum_{i=1}^{6} 4^i$ nodes, that is the number of nodes of the full subtrees rooted at the common prefix. This means that instead of having 5464 nodes we just have 4 (including the root).

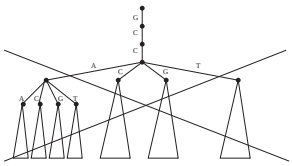


Fig. 4. Space saving with the compact approach for the matrix M00003 of the Jaspar database.

In terms of a compact approach, the minimum gain partitions the set of words that are a hit for a given matrix and threshold in classes in which the representative is the shortest prefix with a score that, summed up with the minimum sum of score that can follow, is above the threshold.

7. Conclusion

In this chapter we discussed the compact approach for the discovery of significant signals (motifs) in biological sequences. Compact approaches are characterized by a significant reduction of the size of the output, without loss of crucial information. Compact approaches that have been developed in these years for several different motif models have been illustrated and discussed, also with the help of practical examples.

8. References

Abouelhoda, M. I., Kurtz, S. & Ohlebusch, E. (2004). Replacing suffix trees with enhanced suffix arrays, *J. Discrete Algorithms* 2(1): 53–86.

Aho, A., Hopcroft, J. & Ullman, J. (1974). *The design and analysis of computer algorithms.*, Addison-Wesley, Reading Mass.

Apostolico, A., Bock, M. E. & Lonardi, S. (2003). Monotony of surprise and large-scale quest for unusual words, *Journal of Computational Biology* 10(2/3): 283–311.

Apostolico, A. & Parida, L. (2004). Incremental paradigms of motif discovery, *Journal of Computational Biology* 11(1): 15–25.

Apostolico, A., Parida, L. & Rombo,S.E. (2008). Motif patterns in 2D, *Theoretical Computer Science* 390(1): 40–55.

Apostolico, A. & Pizzi, C. (2007). Motif discovery by monotone scores, *Discrete Applied Mathematics* 155(6-7): 695–706.

Apostolico, A. & Pizzi, C. (2008). Scoring unusual words with varying mismatch errors, *Mathematics in Computer Science* 1(4): 639–653.

Apostolico, A., Pizzi, C. & Satta, G. (2004). Optimal discovery of subword association in strings, *Proceedings of the seventh International Conference on Discovery Science (DS04)*, Vol. 3245 of *LNCS*, Springer-Verlag, pp. 270–277.

Arimura, H., Arikawa, S. & Shimozono, S. (2000). Efficient discovery of optimal word-association patterns in large text databases, *New Generation Computing* 18: 49–60.

Brazma, A., Jonassen, I., Eidhammer, I. & Gilbert, D. (1998). Approaches to the automatic discovery of patterns in biosequences, *Journal of Computational Biology* 5(2): 277–304.

Califano, A. (2000). Splash: structural pattern localization analysis by sequential histograms, *Bioinformatics* 16(4): 341–357.

Hertz, G. Z. & Stormo, G. D. (1999). Identifying dna and protein patterns with statistically significant alignments of multiple sequences, *Bioinformatics* 15: 563–577.

Keich, U. & Pevzner, P. A. (2002). Finding motifs in the twilight zone, *RECOMB*, pp. 195–204.

Lawrence, C., Altschul, S., Bugoski, M., Liu, J., Neuwald, A. & Wootton, J. (1993). Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment, *Science* 262: 208–214.

Manber, U. & Myers, E. W. (1993). Suffix arrays: A new method for on-line string searches, *SIAM J. Comput.* 22(5): 935–948.

McCreight, E. M. (1976). A space-economical suffix tree construction algorithm, *Journal of the ACM* 23(2): 262–272.

Parida, L., Rigoutsos, I., Floratos, A., Platt, D. E. & Gao, Y. (2000). Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm, *SODA*, pp. 297–308.

Pelfrêne, J., Abdeddaïm, S. & Alexandre, J. (2003). Extracting approximate patterns, *Combinatorial Pattern Matching*, pp. 328–347.

Pevzner, P. A. & Sze, S.-H. (2000). Combinatorial approaches to finding subtle signals in DNA sequences, *Proceedings of the Eight International Conference on Intelligent Systems for Molecular Biology (ISMB00)*, AAAI Press, pp. 269–278.

Pisanti, N., Crochemore, M., Grossi, R. & Sagot, M.-F. (2005). Bases of motifs for generating repeated patterns with wild cards, *IEEE/ACM Trans. Comput. Biology Bioinform.* 2(1): 40–50.

Pizzi, C. & Bianco, M. (2009). Expectation of Strings with Mismatches under Markov Chain Distribution, *Proceedings of the sixteenth International Symposium on String Algoeirthms and Information Retrieval (SPIRE09)*, Vol. 5721 of *LNCS*, Springer-Verlag, pp. 222–233.

Pizzi, C., Rastas, P. & Ukkonen, E. (2011). Finding significant matches of position weight matrices in linear time, *IEEE/ACM Trans. Comput. Biology Bioinform.* 8(1): 69–79.

Quandt, K., Frech, K., Karas, H., Wingender, E. & Werner, T. (1995). Matind and matinspector: new fast and versatile tools for detection of consensus matches in nucleotide sequence data, *Nucleic Acids Research* .

Queen, C., Wegman, M. & Korn, L. (1982). Improvements to a program for dna analysis: a procedure to find homologies among many sequences, *Nucleic Acid Research* 10: 449–456.

Rigoutsos, I. & Floratos, A. (1998). Combinatorial pattern discovery in biological sequences, *Bioinformatics* 14: 55–67.

Sandelin, A., Alkema, W., Engstrom, P., Wasserman, W. & Lanhard, B. (2004). Jaspar: an open-access database for eukaryotic transcription factor binding profiles, *Nucleic Acids Research* 32.

Staden, R. (1989). Methods for calculating the probabilities of finding patterns in sequences, *Computer Applications in the Biosciences* 5(2): 89–96.

Stormo, G. & III, G. H. (1989). Identifying protein-binding sites from unaligned dna fragments., *Proceedings of the National Academy of Science USA*, Vol. 86, pp. 1183–1187.

Tompa, M. (1999). An exact method for finding short motifs in sequences, with application to the ribosome binding site problem, *ISMB*, pp. 262–271.

Tompa, M., Li, N., Bailey, T. L., Church, G. M., Moor, B. D., Eskin, E., Favorov, A. V., Frith, M. C., Fu, Y., Kent, W. J., Makeev, V. J., Mironov, A. A., Noble, W. S., Pavesi, G., Pesole, G., Regnier, M., Simonis, N., Sinha, S., Thijs, G., van Helden, J., Vandenbogaert, M., Weng, Z., Workman, C., Ye, C. & Zhu, Z. (2005). Assessing computational tools for the discovery of transcription factor binding sites, *Nature Biotech.* 23(1): 137 – 144.

Ukkonen, E. (1995). On-line construction of suffix trees, *Algorithmica* 14(3): 249–260.

Ukkonen, E. (2009). Maximal and minimal representations of gapped and non-gapped motifs of a string, *Theor. Comput. Sci.* 410(43): 4341–4349.

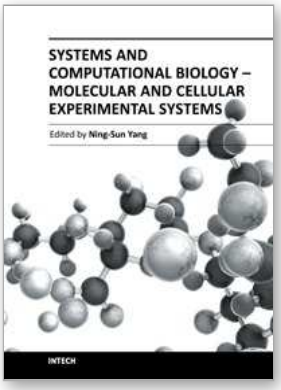
van Helden, J., André, B. & Collado-Vides, J. (1998). Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies, *J. Mol. Biol.* 281: 827–842.

Wang, J. T.-L., Chirn, G.-W., Marr, T. G., Shapiro, B. A., Shasha, D. & Zhang, K. (1994). Combinatorial pattern discovery for scientific data: Some preliminary results, *in* R. T. Snodgrass & M. Winslett (eds), *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994*, ACM Press, pp. 115–125.

Waterman, M., Arratia, R. & Galas, D. (1984). Pattern recognition in several sequences: consensus and alignment, 46: 515–527.

Weiner, P. (1973). Linear pattern matching algorithms.

Wu, T. D., Nevill-Manning, C. G. & Brutlag, D. L. (2000). Fast probabilistic analysis of sequence function using scoring matrices, *Bioinformatics* 16(3): 233–244.



Systems and Computational Biology - Molecular and Cellular Experimental Systems

Edited by Prof. Ning-Sun Yang

ISBN 978-953-307-280-7
Hard cover, 332 pages
Publisher InTech
Published online 15, September, 2011
Published in print edition September, 2011

Whereas some â€œmicroarrayâ€ or â€œbioinformaticsâ€ scientists among us may have been criticized as doing â€œcataloging researchâ€, the majority of us believe that we are sincerely exploring new scientific and technological systems to benefit human health, human food and animal feed production, and environmental protections. Indeed, we are humbled by the complexity, extent and beauty of cross-talks in various biological systems; on the other hand, we are becoming more educated and are able to start addressing honestly and skillfully the various important issues concerning translational medicine, global agriculture, and the environment. The two volumes of this book presents a series of high-quality research or review articles in a timely fashion to this emerging research field of our scientific community.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Cinzia Pizzi (2011). Motif Discovery with Compact Approaches - Design and Applications, Systems and Computational Biology - Molecular and Cellular Experimental Systems, Prof. Ning-Sun Yang (Ed.), ISBN: 978-953-307-280-7, InTech, Available from: http://www.intechopen.com/books/systems-and-computational-biology-molecular-and-cellular-experimental-systems/motif-discovery-with-compact-approaches-design-and-applications

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821