Revista de Sistemas de Informação da FSMA
n. 26 (2020) pp. 10-30

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

A Survey on Solutions for Planted Motif Search Challenging Instances

Deiviston S. Aguena, *Câmpus de Coxim*
Henrique Mongelli, *Faculdade de Computação*
Nalvo F. Almeida, *Faculdade de Computação*
UFMS – Universidade Federal de Mato Grosso do Sul

***Abstract*—In the gene expression process, a transcription factor molecule will bind to a short substring in the promoter region of a gene in order to start the transcription process. This short substring, called motif, appear imperfectly conserved over several genes promoter regions. The discovery of motifs over a set of sequences representing the promoter regions is an important problem in Bioinformatics. Pevzner and Sze, in 2000, have introduced the planted (l, d)-motif search (PMS) problem to find motifs in a set of sequences where l is the motif length and d is the maximum difference between the motif found and its occurrences in the set. Burlher and Tompa, in 2001, studied this problem and, based on their studies, it was possible to classify certain instances of the problem, considered more difficult, as *challenging instances*. Since then, many approaches have been proposed to solve PMS challenging instances, but there are still limitations on the maximum size of instances supported by these approaches. In this work we present a review of solutions for PMS challenging instances.**

***Index Terms*—PMS, Motif-discovery, PMS Challenging instances, Motifs**

I. Introduction

MOTIFS are short recurring patterns of biological interest and they are found in genome sequences of numerous species. They occur in regulatory regions such as promoters and are related with the gene expression process. In this gene expression process, a transcription factor molecule will find a motif in the promoter region to start the mechanism of gene expression. The motif discovery plays an important role in gene expression studies [73, 22].

The way motifs are conceptually modeled is the basis for definition and formalization of search algorithms. For example, a motif can be conceptually modeled as a simple word, i.e, a string of symbols. The algorithm searches for occurrences of a substring of the same size in a set of biological sequences to verify if this word is a motif.

Another way would be to model the motif as an matrix of dimensions Σ x l, where Σ is the alphabet size and l is the motif length. The search algorithm fills the matrices with the frequency of each symbol found in substrings of the set of biological sequences. A function is used to score and classify the matrices and the algorithm tries to locate the motif from the matrices with higher scores.

Corresponding Author: *deiviston.aguena@ufms.br*

Previous reviews

Due to the importance of the motif search problem, numerous approaches have been proposed. In order to improve the understanding of these approaches, several survey papers have been published, each one covering different aspects of the approaches.

In 1997, Brazma *et al.* [7] presented a survey on methods for motif search in biological sequences. They proposed to include these methods in a set of formalisms of the motif search problem, evaluations and classifications of pattern languages and algorithms approaches.

A very comprehensive review on the initial development of motif models and motif search problems is given by Storno (2000) [70].

In 2005, Hu, Li and Kihara [38] presented a set of prediction performance measures for five motif discovery algorithms to conduct an comparative evaluation in terms of their prediction accuracy, scalability and the reliability of their significance score, using datasets from *Escherichia coli* RegulonDB[1].

Tompa, Li, *et al.* [75], in 2005, performed an assessment of 13 computational tools for discovery of transcription factor binding sites. Experts were chosen to test each tool with data sets created from known binding sites. In this way each tool could be tested with a good setting of parameters.

In 2006, another survey on motif search methods was presented by Sandve and Drabløs [67]. This survey also uses an integrated framework to classify the algorithms, according to the organization of the genome in four hierarchical levels: single motifs, composite elements, genes and genomes.

Das and Dai [14], in 2007, reviewed the existing algorithms to motif finding and classified them into three classes: *i*) algorithms based on promoter sequences of coregulated genes; *ii)* algorithms based on phylogenetic footprinting; and *iii)* algorithms based on promoter sequences of coregulated genes and phylogenetic footprinting.

[1]RegulonDB is the primary database on transcriptional regulation in *Escherichia coli* K-12 containing knowledge manually curated from original scientific publications. RegulonDB is available at http://regulondb.ccg.unam.mx/.

All surveys presented between 2014 and 2018, addressed motif find tools for detecting binding site motifs in Chromatin Immunoprecipitation Sequencing (ChIP-Seq) data. Tran and Huang [76], in 2014, provided a review and comparison of nine motif search Web tools that are capable of detecting binding site motifs in ChIP-Seq data. In their work were presented capabilities, advantages and limitations of these tools. Lihu and Holban [49], in 2015, reviewed seven ensemble tools designed to process ChIP-Seq data and observed their limitations and strengths. Liu *et al.* [50], in 2018, provided an algorithmic perspective of *de novo* cis-regulatory motif finding tools based on ChIP-seq data. In that study they reviewed existing motif-finding methods for ChIP-seq data from an algorithmic perspective to provide new computational insight into this field.

The most recent reviews address motif search algorithms in general and attempt to classify them into different groups, as well as to categorize different aspects among them. Hashin *et al.* [33], in 2019, presents a general classification of motif discovery algorithms in four approaches: enumerative, probabilistic, combinatorial and nature inspired. Mohanty and Mohanty [53], in 2019, presents a general review of the genetic algorithms used by researchers in the last decade to search for motifs in biological sequences, their strengths and weaknesses, and the progress in this domain.

The common points observed by most of the authors of these surveys were that, each of the reviewed algorithms, when published, used very different computational experiments. The input set, the models used, and the ways in which the results were presented varied greatly, making it difficult to compare different implementations and provide an effective guidance to their readers.

Planted Motif Search Problem

In this work we focus on a variant of motif search problem called planted motif search problem (PMS for short), also known in the literature by (l, d)-motif search. For PMS some input instances are considered more difficult to solve. These instances are know on literature as *challenging instances*. Specifically, we present a review focused mainly in algorithmic solutions that offer guarantees of finding all motifs in challenging instances. We also use the challenging instances as benchmark to compare the algorithmic solutions presented.

PMS was introduced by Pevzner and Sze in 2000 [58]. The authors stimulated searching solutions by observing that none of the best existing algorithms, at that time, was able to find a motif of length $l = 15$, with $d = 4$ random mutations in a set of $t = 20$ sequences of DNA with $m = 600$ nucleotides each. They used an independent and identically distributed (i.i.d.) synthetic sample data where, in each sequence, the nucleotides were equally likely to occur. Then, instances of a motif with length l and d random mutations were planted at random positions in each of these sequences. Considering these variables l and d, we call this input as (l, d) instance of PMS. The specific $(15, 4)$ instance of problem proposed by Pevzner and Sze [58], became known in the literature as a *Challenge Problem*. Pevzner and Sze's strategy, using this synthetic sample data with a planted signal, created a controlled testing environment to compare the pros and cons of different algorithmic approaches for PMS.

For the sake of simplicity, in the remainder of the text, we will use the S_p notation to refer to a sample set (i.i.d.) containing 20 sequences of 600 nucleotides in length and with the l-length motif instances planted with d random mutations. Also, if there will not be an explicit mention of the input set, we will always refer to an input sample like S_p.

In 2001, through the probabilistic analysis of Buhler and Tompa [8], it was possible to have a more comprehensive notion regarding of PMS instances. According to them, for certain instances, considering the same input set, but for small values of d, the problem was quantitatively different. The probabilistic analysis allowed to classify certain instances, considered more difficult, as *challenging instances*. The basis for classifying an instance as challenging is related to the expected number of random (or spurious) patterns that can occur in the sequence set of this instances. Buhler and Tompa describe that if the values for t, m and l are fixed, then there is an upper limit value for d, for which it is unlikely that any algorithm can distinguish a motif planted from a motif found by chance (or spuriously). Thus, the concept of *challenging instance* can be described as follows (adapted from Nicolae, 2016 [54]):

Definition 1.1: **Challenging instance:** Given t m-length sequences over the Σ alphabet. An (l, d) instance is a **challenging** one if d is the largest integer for which the expected number of l-length motifs that would occur in the input randomly (or spurious) does not exceed the value of a constant.

The expected number of l-length spurious motifs, with at maximum d mismatches, which can occur, at least once, in each of the t DNA m-length sequences, can be determined by the following equation (Bulher and Tompa, 2001 [8]):

$$E(l, d) = 4^l(1 - (1 - p_d)^{m-l+1})^t, \tag{1}$$

where p_d is the probability that an l-length string occurs, with at most d mismatches, at a given position of a random sequence. The value of p_d can be calculated as (Bulher and Tompa, 2001 [8]):

$$p_d = \sum_{i=0}^{d} \binom{l}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{l-i}. \tag{2}$$

Thus, for a sample S_p, that have $t = 20$ nucleotide sequences (thus $|\Sigma| = 4$) of length $m = 600$ each, and for a constant equal to 500, some examples of challenging instances are: (9,2), (11,3), (13,4), (15,5), (17,6), (19,7), (21,8), (23,9), (25,10), (26,11), (28,12), (30,13) and (32,14). The values $t = 20$, $m = 60$ and

the constant 500 have been used by several authors in the literature allowing a basis for comparison of their algorithms.

Definition 1.2: A string $x = x[1] \ldots x[l]$ of length l, over an alphabet Σ, $(x[i] \in \Sigma,\ 1 \leq i \leq l)$, is an **l-mer**.

Definition 1.3: Given two l-mers x and y, the **Hamming distance** between x and y, denoted by $d_H(x, y)$, is the number of positions at which the symbols in x and y differ.

Definition 1.4: Given an l-mer x, we define the ***d*-neighborhood** of x, denoted by $B_d(x)$, to be $\{y \mid d_H(x, y) \leq d\}$. We refer to an individual l-mer $x' \in B_d(x)$ as ***d*-neighbor** of x. The number of d-neighbors of x can be calculated by $|B_d(x)| = \sum_{i=0}^{d} \binom{l}{i}(|\Sigma| - 1)^i$. We also write $\mathcal{V}(l, d)$ to refer to $|B_d(x)|$.

Formally the PMS can be defined as follows:

Definition 1.5: **PMS**: Given a set of sequences $S = \{S_1, ..., S_t\}$ over an alphabet Σ such that $|S_i| = m$, $1 \leq i \leq t$, and two positive integers l and d, that $0 \leq d \leq l \leq m$, the **Planted (l,d)-Motif Search** problem is to find all l-mers $\mathscr{M}$ such that exists, at least one occurrence $\mathscr{M}'$ of $\mathscr{M}$ in each S_k, $1 \leq k \leq t$, where $d_H(\mathscr{M}, \mathscr{M}') \leq d$. The string $\mathscr{M}$ is called (l, d)-motif of S and $\mathscr{M}'$ is called an instance of $\mathscr{M}$. We refer to an instance of the problem as (l, d) instance.

PMS is known to be NP-hard [24]. The most common approaches to NP-hard problems use from exact algorithms developed for small and specific instances, for which the solution can be found in a reasonable computational time, to approximate algorithms, where it is guaranteed that the solution found approaches the optimum by a determinate factor, or heuristics, which are strategies for which neither guarantees to find the optimal solution, nor its proximity with the optimal solution, but the computational time is quite reasonable and the solution is empirically acceptable.

There are many heuristic algorithms for the PMS problems. For instance, GibbsDNA by Lawrence *et al.* (1993) [47], MEME by Bailey and Elkan (1994) [2], CONSENSUS by Hetz and Storno (1999) [35], WINNOWER and SP-STAR by Pevzner and Sze (2000) [58], PROJECTION by Buhler and Tompa (2001) [8], MULTPROFILER by Keich and Pevzner (2002) [41], PatternBranching and ProfileBranching of Price *et al.* (2003) [61], VINE by Huang *et al.* (2011) [39] and CEN by Zhang *et al.* (2013) [80]. However, they do not offer guarantees that they find all motifs and had shown historically low performance with challenging instances.

Exact algorithms have exponential execution time in the worst case, due to the complexity of PMS. However the great advantage of these algorithms is that they always find all motifs. For this reason, they may be preferred by biologists, since for them, the motifs found may be much more important than the algorithm runtime. Over time, several exact algorithms have been proposed, solving challenging instances increasingly larger. However, there are still limitations in the maximum size of the instances that these algorithms are able to solve in an acceptable amount of time. Thus, for larger instances, these algorithms need to be improved or new ones must be developed.

In this paper we describe exact algorithmic solutions that were able to solve increasingly larger challenging instances. The main contribution is to summarize and present chronologically the evolution of the solutions used for PMS challenging instances. We also hold a discussion about the main methods used. At the end, we present a conclusion about this theme. The historical review is presented in Section II, the discussion and conclusion are presented in the Section III and Section IV, respectively.

II. Historical review

ACCORDING to Abbas *et al.* [1] there are two combinatorial formulations involving the problem of identifying motifs. The first is known in the literature by *Consensus Motif* and was first appeared in 1984 in the work of Waterman *et al.* [77]. The second was presented in 2000 by Pevzner and Sze [58], which deals with the problem of finding the planted (l, d)-motifs, as described in Section I.

In this section we describe, in chronological order, the main results presented to solve ever greater challenging instances in the literature. In these results, unless otherwise described, we will always refer to exact algorithms and the sample S_p as an input set.

An important data structure used in the algorithms is the tree. Two types of trees are used by some of the PMS algorithms, the Lexicographic Tree and the d-neighborhood Tree.

A **Lexicographic** Tree, denoted by $\mathcal{T}$, is a rooted tree whose nodes, except leaves, have exactly $|\Sigma|$ branches. Each branch, from the root to the leaves, is labeled with a symbol of Σ and is ordered lexicographically from the parent node. A tree, with depth p, has exactly $|\Sigma|^p$ leaves. A path, from the root to the leaf, lists a string of symbols with length p, consisting of the symbols labeled by the branches in the path. There are $|\Sigma|^p$ distinct paths that uniquely enumerate $|\Sigma|^p$ strings of length p. Figure 1 presents a lexicographic tree $\mathcal{T}$ with alphabet $\Sigma = \{0, 1\}$ and depth = 3. This tree has 2^3 distinct paths, each one enumerating a distinct string of symbols with length = 3. In the figure, the path highlighted by dashed lines lists the string '010'.

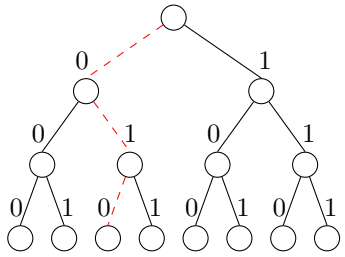

Fig. 1. Tree $\mathcal{T}$ with alphabet $\Sigma = \{0, 1\}$. The Tree have depth = 3 and 2^3 distinct paths from the root to the leaves. The path highlighted by dashed lines lists the string '010'.

A *d*-**neighborhood** Tree of an l-mer x, denoted by $\mathcal{T}_d(x)$, is a rooted tree of depth $p = d$, where each node, with depth p', $0 \leq p' \leq p$, represents an l-mer x' neighbor of x, such that $d_H(x, x') = p'$. Thus, the root and its descendants form the set $B_d(x)$. Formally, $\mathcal{T}_d(x)$ can be constructed by the following rules (adapted from Dinh *et al.*, 2011 [21]):

1. Each node in $\mathcal{T}_d(x)$ is a pair (s, i) where $s = s[1] \ldots s[l]$ is an l-mer and i is an integer, $1 \leq i \leq l$, such that $s[i] \neq x[i]$.
2. The root of $\mathcal{T}_d(x)$ is $(x, 0)$ and the depth of $\mathcal{T}_d(x)$ is d;
3. A node $(s, i) \in \mathcal{T}_d(x)$ is the parent of node $(s', i') \in \mathcal{T}_d(x)$ if and only if
   (a) $i' > i$.
   (b) $s[i] \neq s'[i']$.
   (c) $s[j] = s'[j]$ for any $j \neq i$.

Figure 2 illustrates the tree $\mathcal{T}_d(x)$, with alphabet $\Sigma = \{0, 1\}$, $x = 1010$ and $d = 2$. The value i is represented on each node by the highlighted symbol. It is easy to see that each node on the second level contains exactly two differences from the l-mer of the root, which are exactly the symbols highlighted in the path from these nodes to the root.

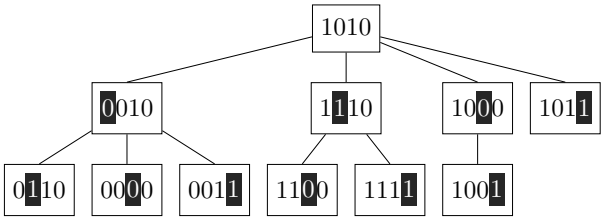

Fig. 2. Tree $\mathcal{T}_2(1010)$ with alphabet $\Sigma = \{1, 0\}$. The value i is represented on each node through the highlighted symbol. For example, $i = 2$ at node 1110, $i = 3$ at node 0000. Adapted from Dinh *et al.*, 2011 [21].

In the following subsections, a short title will highlight the size of the most challenging instance solved to date and the corresponding year or period. If the instance size is omitted in the title, it means that there has been no significant improvement over the previous period.

(6,2) Instance - Consensus sequence (1975)

The first work we highlight is the publication presented by Pribnow in 1975 [60]. Pribnow was able to visually determine, from a few sequences, a well-conserved 'TATA box' sequence centered around 10 base-pairs (bp), upstream of the transcription initiation site of *Escherichia coli* promoters. It was a six-base long consensus sequence with no more than two substitutions for each occurrence in the set, i.e., a typical (6,2) instance. Pribnow observations were only possible because there were a small number of sequences and they could be approximately aligned because the start of the transcription was known. However, more sequences became available and less information was available about them, so it became very difficult to visually determine the consensus sequence (or motifs), making computational algorithms essential [70].

Computer analysis origins (1977–1982)

There are mainly two works to be highlighted in the period from 1977 to 1982. The first one, in 1977, by Korn *et al.* [43], as far as we know, was the first paper describing a computer program that helps to find over-represented substrings with possibly local mutations (or motifs). The second, by Queen *et al.* [62], in 1982, describes a computational routine to be added to Korn *et al.* program [43]. This routine performs an algorithm that, as far we know, is the first exact algorithm to search (l, d)-motifs in a set of sequences. This algorithm tries to find motifs in a set of sequences $S = \{S_1, ..., S_t\}$ where $|S_i| = m$, $1 \leq i \leq t$, searching for l-length patterns p, such that at least q of the sequences contain an l-mer p', occurrence of p, not differing from p by more than d symbols. The algorithm finds the motifs using an array R_i of size $|\Sigma|^l$, for each sequence S_i, $1 \leq i \leq t$, to represent all possible l-mers. Then the algorithm selects every l-mer $x \in S_i$ and, in the corresponding array R_i, increases the value of all indices referring to the d-neighbors of x. The algorithm uses an additional array A of size $|\Sigma|^l$ to construct the final result list. The array A is filled simultaneously with the others arrays R_i. An index in the array A has the value incremented by 1 if the same index in another array R_i receives a value greater than 1. When an index in the array A becomes equal to q, the corresponding l-mer of this index is added to a result list. The time complexity is $O(tml\mathcal{V}(l, d))$ and space is $O((t + 1)|\Sigma|^l)$, becoming prohibitive for large values of l.

Index Based (1985)

Galas *et al.* [30], in 1985, have presented an exact algorithm to search (l, d)-motifs. This algorithm tries to find motifs searching for common words and their neighbors, which are approximate correspondences of these words over a window of possible alignments. This is an implementation and application of the work published previously [77], when the authors presented a mathematical method to solve in general the problem of detecting small unknown patterns in a set of sequences. To extract the motifs, the algorithm uses one array of integers A with size $|\Sigma|^l$ to represent all d-neighbors. Each index of A corresponds to a possible l-mer of Σ and the value at the index corresponds to the number of sequences that contain at least one d-neighbor of this l-mer. The algorithm reads each index i of A and applies a function that computes the d-neighborhood of the l-mer referenced by i and the number of sequences that contains at least one occurrence of an d-neighbor of this d-neighborhood. At the end, an scan in A reveals which indices had value equal to t, i.e., which are (l, d)-motifs of the input sample. The time complexity is $O(|\Sigma|^l tm\mathcal{V}(l, d))$ and space is $O(|\Sigma|^l)$, becoming prohibitive for large values of l.

Storno [70] describes the algorithm of Galas *et al.* to be the first algorithm for (l, d)-motifs finding, however, according to our research we believe that the first algorithm for (l, d)-motifs finding is due to Queen *et al.* (1982) [62].

Heuristic and Enumerative methods (1985–1997)

From 1985 to 1997, while more DNA sequences became available, other methods have emerged [70], most of which were heuristics employing statistical or combinatorial tools, e.g., Bailey and Elkan (1994) [2], Roytberg (1992) [64], Frech *et al.* (1993) [28], Lawrence *et al.* (1993) [47]. These methods look for subtle patterns in unaligned DNA sequences allowing, but not necessarily setting, a maximum number of mismatches present in the model. In addition, methods based on enumeration have also been proposed, e.g., Pesole *et al.* (1992) [57], Staden (1989) [69], Tompa *et al.* (1999) [74]. Algorithms based on these methods offer guarantees of finding the motifs with the highest score in the input set, but unfortunately they became impractical for long motifs or when many mutations are allowed as in challenging instances. A good review on approaches with motif search in this period can be find in Brazma *et al.* (1998) [7].

Suffix tree (1998)

In 1998, Sagot [65] have presented an exact algorithm named SPELLER that extracts motifs from a set of sequences S, with $|S| \geq 2$, and these motifs must occur, with at most d mismatches, in $1 \leq q \leq |S|$ distinct sequences of the set. The SPELLER tries to find the (l, d)-motifs by increasing lengths of a simulated traversal of a virtual lexicographic tree $\mathcal{T}$. The virtual term was used because this tree was not properly maintained in memory, but only a traversal that uses recursion. This traversal was limited to the l depth and by the validations performed to prune branches that do not list a valid motif. These validations were performed using a generalized suffix tree $\mathcal{GT}$, previously constructed from the sample sequences, and occurrences stored during the traversal. The traversal in $\mathcal{T}$, in the worst case, explores all d-neighborhood in S. In traversal, the main operations performed were the query and maintenance of $\mathcal{GT}$. In this way, the algorithm complexity time becomes $O(mt^2\mathcal{V}(l, d))$. In terms of space requirement, only very little sequence information was stored on each node of $\mathcal{GT}$. If a bit vector with the same size w of a word machine was used for this storage, then the space complexity will be bounded by $O(mt^2/w)$.

(15,4) Instance - Challenge Problem (2000)

In 2000, Pevzner and Sze [58] have introduced the Planted (l, d)-motif Problem (PMS for short), to find similar patterns in sequences which represent the promoter region of co-regulated genes, where l was the length of the pattern and d was the maximum Hamming distance around the similar patterns. This patterns in DNA sequences are know on biology field as motifs. The authors also defined the *Challenging Problem*, a specific (15,4) instance of the PMS as: "find a signal in a sample of sequences, each 600 nucleotides long and each containing an unknown signal (pattern) of 15 length with 4 mismatches". They observed that, at that time, the best available algorithms, like CONSENSUS by Hetz and Storno (1999) [35], GibbsDNA by Lawrence *et al.*(1993) [47] and MEME by Bailey and Elkan (1994) [2], have failed to find such signal planted in an i.i.d. sample even with nucleotides with the equal probabilities to occur. Pevzner and Sze have presented two heuristic algorithms that successfully solved the (15,4) instance, the WINNOWER and the SP-STAR.

Briefly, WINNOWER represents all l-mers as vertices of the input sample in a t-partite graph G. Each partite contains $m - l + 1$ representing all l-mers in an input sequence. Two vertices from different partites, and consequently from different input sequences, are linked if the Hamming distance between them is at most $2d$. The key observation is that all vertices that are motifs instances of a motif $\mathscr{M}$ form a clique. A clique in a graph is a set of vertices any two of which are connected by an edge. WINNOWER tries to retrieve the planted (l, d)-motif looking for a click of size t. The strategy is systematically to delete spurious links between the vertices to find the cliques.

The SP-STAR treats every l-mer in the sample as potential motif. In order to construct an initial model, it selects every l-mer x in the sample and finds the closest match of x from every l-mer y belonging to other input sequences, using a sum-of-pairs scoring function. Then, the initial model was refined iteratively to converge on a good motif.

(14,4) Instance - Probabilistic Analysis (2001)

In 2001, Buhler and Tompa [8] presented a heuristic algorithm called PROJECTION. This algorithm improved the scoring strategy of existing algorithms using random projections of the input to find the motif.

Random projection is an approach to partitioning and indexing the input. According with authors, this approach had been used in previous work, in vision and geometry, and in bioinformatics fields, both focused on the detection of similarity of pairs of sequences. In PROJECTION, these approach was extended to address the multiple sequence alignment problem.

Briefly, PROJECTION randomly chooses k from the l positions of the l-mers of input, to group them into 4^k buckets, according to the similarity of the projections, i.e., according to the similarity of the bases of the l-mers in these k positions. For large enough buckets, the motif is found by using an expectation maximization (EM) step, according to the idea given by Lawrence and Reilly (1990) [46]. The idea of the algorithm is that buckets with large numbers of l-mers have a high probability of containing the desired motif.

PROJECTION was able to successfully solve the instances (14,4), (16,5), and (18,6), which the algorithm

of Pevzner and Sze (2000) [58] failed to solve, and all considered more difficult than the (15,4) instance. However, they observed that PROJECTION was not able to retrieve motifs in very similar instances, such the instances (13,4), (15,5) and (17,6). In order to investigate these fails, they performed a probabilistic analysis on different instances to calculate the expected number of motifs that could occur by random chance (or spurious) $E(l, d)$ (see Equation 1). Then, they concluded that, considering the same input set, for example, for the sample S_p, and for small values of d, the problem involving planted motifs on certain instances was quantitatively different from others. For example, for the instances $(l + 1, d)$ and $(l, d + 1)$, as we can see on Table I, while the expected number of spurious motifs for the instance $(l + 1, d)$ is insignificant, the expected number of spurious motifs for the $(l, d + 1)$ instance is high, each one so well preserved as the planted motif. This means that if the expected number of spurious motifs is too small, it is likely that the planted motif stands out in the input sample sufficiently to be found with less difficulty. On the other hand, if the expected number of spurious motifs is too high, then any algorithm will probably not distinguish a spurious motif from a planted motif. In this way, the challenging instances can be considered as a limit between these two scenarios. Table I presents, for the input sample S_p, some (l, d) challenging instances (see definition in Section I) and the expected number of spurious motifs for them. For comparison purposes, the expected numbers of spurious motifs for instances $(l, d + 1)$ and $(l + 1, d)$ are also presented.

TABLE I
COMPARISON BETWEEN THE NUMBER OF EXPECTED SPURIOUS MOTIFS $E(l, d+1)$, $E(l, d)$ AND $E(l+1, d)$.

(l, d)	$E(l, d+1)$	$E(l, d)$	$E(l+1, d)$
(9,2)	$2.4\text{x}10^{5}$	1.6	$6.1\text{x}10^{-8}$
(11,3)	$3.3\text{x}10^{6}$	4.7	$3.2\text{x}10^{-7}$
(13,4)	$3.2\text{x}10^{7}$	5.2	$4.2\text{x}10^{-7}$
(15,5)	$1.8\text{x}10^{8}$	2.8	$2.3\text{x}10^{-7}$
(17,6)	$4.8\text{x}10^{8}$	0.88	$7.1\text{x}10^{-8}$

(9,2) Instance - NP-hardness of PMS (2002-2004)

With the work of Buhler and Tompa (2001) [8] it was possible to have a better understanding about the instances of PMS, in particular, regarding challenging instances. In the period from 2002 to 2004, new algorithms were proposed, some as extensions of existing algorithms and others combining different approaches. However, only the challenging instance (9,2) was solved in this period. Briefly, we report the main occurrences, including the work of Evans *et al.* (2003) [24] regarding NP-completeness of PMS.

In 2002, Keich and Pevzner [41] presented the heuristic algorithm called MULTIPROFILER. This algorithm has a hybrid approach that combines the use of pattern-driven and sample-driven techniques.

In the pattern-driven approach all 4^l patterns of size l are tested with the input sample. The problem with this approach is the cost associated with exhaustive search in 4^l patterns. To avoid this prohibitive set of 4^l patterns, other algorithms generates a smaller set of *seeds patterns* whose neighborhood are then explored by local searches.

In the sample-driven approach the motif is found directly in the sample. Algorithms, that use the sample-driven approach, select l-mers from different sample sequences, then construct a multiple alignment to evaluate the similarities among them.

In general, these algorithms use some strategy to limit the search space and avoid the test of all $(m - 1 + 1)^t$ possible alignments (considering a sample with t sequences of size m). The problem is that if the strategy used is too restrictive, it may fail to find subtle patterns, otherwise it may turn the search space as impractical as pattern-driven approaches. Then, because the pattern-driven approach uses high time consumption and the sample-driven approach often fails to find subtle patterns, they have developed a combinatorial algorithm MULTIPROFILER that extends the search capabilities of the sample-driven approach and avoids the computational complexity of the pattern-driven approach.

The authors highlighted two new ideas behind MULTIPROFILER. The first was to use the neighborhood of each l-mer in the sample with a possible dictionary that suggested how to 'spell' the motif. The second was the use of multi-positional profiles to improve the filtering between random words and possible motif occurrences in sample.

The authors have used the best algorithm at the time, the PROJECTION by Buhler and Tompa (2001) [8], to compare with MULTPROFILER. In the tests performed, MULTIPROFILER achieved better results than PROJECTION on searching for (15,4)-motifs in an input set containing 20 sequences of 2000 and 3000 bases length, finding the planted motif respectively 99% and 98% of the time. The authors also report that MULTIPROFILER was able to find the planted (9,2)-motif, 100% of the time, in a set of 20 sequences containing 600 bases length, becoming the first algorithm capable of finding (l, d)-motifs in a challenging instance. A more general analysis of MULTIPROFILER can be found in Keich and Pevzner (2002) [42].

In other work, in 2002, Eskin and Pevzner [23] have proposed a new exact algorithm called MITRA (*MIsmatch TRee Algorithm*), and two implementation versions, the MITRA-Count and MITRA-Graph. The MITRA-Count uses similar strategy to that employed by Sagot (1998) [65] for SPELLER and the MITRA-Graph uses similar techniques to those used by Pevzner and Sze (2000) [58] for WINNOWER.

In general, MITRA tries to find (l, d)-motif using simulated traversals in a virtual tree $\mathcal{T}$, where the branches are labeled with symbols of $\Sigma = \{A, C, G, T\}$ and the nodes are used to store information about occurrences in traversal.

In MITRA-Count, the idea is that all l-mers in the sample could be indexed by the arrays representing the nodes in $\mathcal{T}$. At the beginning, each node immediately linked with the root stores an array that indexes all l-mers of the input. In this array, each index is set with 0 value if the first symbol of the l-mer indexed by the index is equal to the labeled symbol in the branch, and 1 otherwise. Then, a node is selected and expanded into $|\Sigma|$ children nodes, each one linked with parent by a branch labeled by distinct symbols of Σ. Each child node stores a copy of the array (of the parent node). The indices of this copied array are updated by checking the branch symbol with the p-th symbol of the l-mers referenced by these indices, such that p was the level of the child node. If the symbols are equal then the index value is maintained, otherwise it is incremented by one. Each index that reached value v, where $v > d$, is eliminated from the array. This exploration process in depth is finalized when one of the two situations occurs: if the level l is reached, and in this case the motif is found; or if there is no l-mer of any sequence being indexed by the array.

MITRA-Graph, instead of using an array for counting mismatches as in MITRA-Count, on each node of $\mathcal{T}$ a graph G is implicitly constructed. In these graphs, it performs checks to try to remove spurious edges in a process similar of WINNOWER. The main innovation is that MITRA knows the prefix of the motif it is looking for, while WINNOWER did not. In MITRA-Graph, as the depth advances in $\mathcal{T}$, it uses p, the prefix formed in the path, to extend the pairwise filter of WINNOWER and remove larger amounts of spurious edges in G. The MITRA filter extension works as follows. Given α substitutions that occurs from position $k+1$ to position l, $k \leq l$, between two l-mers x and y, and d_x the number of substitutions between p and x, from position 1 to position k, and d_y the number of substitutions between p and y, from position 1 to position k. The extended filter consists of removing any edges between $\{x, y\}$, such that $dx + dy + \alpha \geq 2d$.

Although the complexity of time and space has not been discussed for MITRA, as well as SPELLER, the time complexity was proportional to neighborhood size, $O(tml\mathcal{V}(l, d))$, because of the exhaustive enumerative approach. The space complexity was proportional to the input set $O(tml)$, since only information about the l-mers of the input set was stored.

In tests reported by authors, both variants of MITRA were able to solve difficult instances such as (14,4), (16,5) and (18,6). In addition, the graph-based variant MITRA-Graph was able to solve larger instances such as (28,8) and (30,9). In this period, others graph-based approaches have been proposed, e.g., Liang *et al.* (2004) [48], Sze *et al.* (2004) [71]. However, only improvements in the average runtime of the same instances reported by MITRA-Graph were obtained and no solution was reported for more difficult instances.

MITRA algorithm was also able to find structured motifs[2], also known as compound motifs. Marsan and Sagot (2000) [52] were one of the first authors to address the problem of extracting structured motifs. Other approaches have been proposed by several authors, e.g., Helden *et al.* (2000) [34], Liu *et al.* (2000) [51], GuhaThakurta and Storno (2001) [31], Favorov *et al.* (2005) [26], Carvalho *et al.* (2005) [9], Pisanti *et al.* (2006) [59], Zhang and Zaki (2006) [81], Zhou *et al.* (2006) [82], Federico *et al.* (2009) [27]. The problem of searching for structured motifs will not be covered in this work, however the works referenced above constitute a good initial basis.

Evans, Smith and Wareham [24], in 2003, have used techniques from parameterized complexity to assess non-polynomial time algorithmic options and complexity for the Common Approximate Substring (CAS) Problem. Through their analyses it was possible to identify which parameters of the problem should be restricted and which non-polynomial time algorithms can be used, in addition to parameterized reductions to prove the inclusion of the problem as a member of the NP-hard class.

The CAS problem is essentially the PMS problem when $\Sigma = \{A, C, G, T\}$. The PMS is also very similar to the Closest Substring problem, also NP-hard [45, 44]. Some authors refer to Closest Substring problem to relate the PMS to the NP-hard class. The Closest Substring problem is essentially the PMS problem where the aim is to find the smallest d for which there exists at least one motif. In this way, the Closest Substring problem can be solved by a linear number of calls to PMS. Therefore, there is a polynomial time reduction from Closest Substring to PMS, which means that the PMS problem is also NP-hard.

(15,5) Instance - Voting (2005)

Chin and Leung [12], in 2005, presented the exact algorithm VOTING. We highlight this algorithm because it was the first to report the solution of the challenging instances (11,3), (13,4) and (15,5). VOTING was designed with the following observation. Assume that $N_d(S_1)$ is the set of all d-neighbors of l-mers in S_1, then note that any (l, d)-motif of S is necessarily contained in $N_d(S_1)$, as well in $N_d(S_i)$ $\forall i, 1 < i \leq t$. The idea is to explore the d-neighbors of all input sequences, then determine through votes the common elements among them.

Briefly, the algorithm iteratively explores each d-neighbor of each l-mer in each sequence to compute the votes. Two hash tables are used to control the votes. The hash table R stores, for each d-neighbor, if it has already received a vote from the $S_i, 1 \leq i \leq t$. The hash table V store, for each d-neighbor, the total number of sequences that has voted for it. Finally, a scan verifies which d-neighbor has received votes from all t sequences.

[2]Structured motifs as described by Sagot [52], are ordered collections of $p \geq 1$ "boxes" (each box corresponding to one part of the structured motif), substitution rates (one for each box) and one interval of distance (one for each pair of successive boxes in the collection).

The time complexity was $O(tm\mathcal{V}(l,d))$ since all d-neighbors of the sample were explored. The space complexity was $O(m\mathcal{V}(l,d))$ since all d-neighbors of $(m-l+1)$ l-mers of the first sequence required to be stored. Considering that each entry in V and R tables stores an integer of 2 bytes, for instance (15,5) it would require approximately 2 GB of memory, just to store the two tables. For the instance (17,6) it would require approximately 12 GB of memory, makes the memory requirement not practical.

To improve space complexity, a proposed approach was to split all l-mers of the sample into groups, where all l-mers that shared the same suffixes with size l', $l' \leq l$, were stored into the same group. Then, each group could be processed separately. With this approach, the space complexity would be reduced to $O(tm\mathcal{V}(l-l',d))$, while the time complexity would increase to $O(tm\mathcal{V}(l,d) + tm4^{l'})$. Note that, the total neighborhood generated will be the same, but the l-mers in input sequences will be read $4^{l'}$ times. The authors tested VOTING algorithm with this approach and reported that it is able to solve the challenging instances (9,2), (11,3), (13,4) and (15,5), with execution time, respectively, 0.4 seconds, 8.6 seconds, 108 seconds and 22 minutes, using a computer with 2.4 GHz processor and 512 Mb RAM.

Another approach described by the authors involved the use of projections, where instead of considering all positions of the l-mers, the algorithm considered only l' of the l positions of the l-mers to find motif (similar method as the used by Buhler and Tompa (2001) [8]). The computation of votes was modified to considering only these l' positions. Based on these votes the modified algorithm could with high probability find the motif of length l. This version of VOTING was able to solve the same instances reported for the previous version. In addition, it was reported the solution of non-challenging instances (20,7), (30,11) and (40,15) with success rate over 95% and execution time of maximum one day.

Rajasekaran, Balla and Huang [63], in 2005, presented the exact algorithms PMS1 and PMS2. The observation is that both algorithms are designed to search instances $\mathscr{M}'$ of the (l,d)-motif in the sample, such that the $d_H(\mathscr{M}',\mathscr{M})$ is *exactly* d instead *at most* d. Then we use the d^*-neighborhood instead of d-neighborhood (as described in Section I, Definition 1.4) to refer that restricted neighborhood where only neighbors that has exactly d mismatches from the l-mers in the sample are considered.

Briefly, PMS1 worked as follows. First, all the d^*-neighborhood of S_1 are generated and stored into set M, then M are lexicographically ordered (in time $O(l)$ as presented by Horowitz *et al.* (1998) [37]) and the duplicates are removed. Then, the d^*-neighborhood of S_i, $1 < i \leq t$, are iteratively generated into set C_i and, taking advantage of the ordering, it makes $M = M \cap C_i$. After $t-1$ iterations, the intersection operation maintains in M only the common (l,d)-motifs of t sequences. The time complexity of PMS1 was $O(tml\mathcal{V}(l,d))$, proportional to the size of d-neighborhood of the input. The space complexity was also proportional to common d^*-neighborhood of $(m-l+1)$ l-mers of first sequence, $O(m\mathcal{V}(l,d))$ in worst case. So PMS1 is impractical depending on the values of l and d.

The authors described an implementation that considered each l-mer as a 4-byte integer. Thus, in the worst case, for the instance $(15,5)$ the estimated use of memory would be approximately 1.9 GB just to store the d-neighborhood of the first sequence.

PMS2 explored the observations that: i) given an (l,d)-motif x of S, at least $l-k+1$ substrings of x, with size k, or k-mers, should occur in each sequence of input; ii) there should be at least one position i_j in each input sequence, such that successive k-mers of x would occur at the positions $i_j, i_j+1, ...i_j+l-k$. In this way, the k-mers could be used to find the motifs. PMS2 uses a modified version of PMS1 that explores the common d-neighborhood (instead of d^*-neighborhood) of S, to find all (k,d)-motifs $\mathscr{M}'$ of S, where $k = d + c$. Then, selects an arbitrary sequence, for example S_1, and for each k-mer y that occurs in the i-th position of S_1, such that $d_H(y,\mathscr{M}') \leq d$, for an arbitrary $\mathscr{M}'$ of S, the respective k-mer y of S_1 is stored in the i-th position of the list L. If there are $\{y_1,y_2\} \in L$, respectively at the positions L_i and L_{i+l-k}, such that the last $2k-l$ symbols of y_1 are equal to the first $2k-l$ symbols of y_2, then an l-mer l_1 could be formed by appending the last $(l-k)$ symbols of y_2 to y_1. The l-mer l_1 is stored in the list M', a set of possible motif candidates, if $d_H(l_1,l_2) = d$, where l_2 was the l-mer found at the i-th position of S_1. Finally, PMS2 checks the candidates from M' to the sequences of input in time $O(|M'|tml)$. The total time of PMS2 is proportional to the size of d-neighborhood of the k-mers of sample $O(tmk\mathcal{V}(k,d))$, plus the time of generating motif candidates based on L, $O(\sum_1^{l-k+1} |L_i||\Sigma|^{(l-k)}l)$, and the verification of candidates in the sample in time $O(|M'|tml)$.

The authors ran PMS2 using a computer Pentium IV with 2.4 GHz processor and 1 GB RAM. They reported that PMS2 was able to solve the challenging instances (9,2), (11,2) e (13,4). However, they did not perform tests with $d = 5$ and $d = 6$ due to insufficient memory in the computational environment used. The authors compared the results obtained with the results reported, using a 750 MHz processor and 1 Gb RAM, by Eskin and Pevzner (2002) [23] for MITRA, the best exact algorithm at the time according to them. For instances (11,2), (12,3) and (14,4) it was reported that PMS2 had used only a fraction of the time used by MITRA.

We show runtimes reported by PMS2 in Table II and add in this table the results reported by Chin and Leung (2005) [12] for VOTING. Note that both algorithms were presented in same year and have tested by their authors in similar computational environment, a 2.4 GHz processor with 1 GB RAM for PMS2 versus a 2.4 GHz processor with 512Mb RAM for VOTING. In this table, '-' means that the algorithm uses too much memory in the instance, took too long or its time was not reported. Time in seconds s and minutes m.

TABLE II
Comparison of Runtimes for PMS2 and VOTING

(l,d)	PMS2	VOTING
(9,2)	1.44 s	0.4 s
(11,3)	19.84 s	8.6 s
(13,4)	228.94 s	108 s
(15,5)	-	22 m

According to results presented in Table II, we can conclude that VOTING was more efficient than PMS2.

(17,6) Instance - PMSP (2006)

Davila, Balla and Rajasekaran [19], in 2006, presented the exact algorithm PMSP. This algorithm was the first to report the solution for the challenging instance (17,6).

PMSP was designed using similar ideas of PMS1 by Rajasekaran *et al.* (2005) [63]. However it added strategies that made it more efficient in both time and space. It generated all the d-neighborhood of the first sequence, then it tried to find the (l,d)-motifs directly from this d-neighborhood by checking the occurrences of them in the remaining sequences.

In short, PMSP selects each l-mer $x \in S_1$ and generates the set $N_i = \{y \in S_1 : d_H(x,y) \leq 2d\}$, $2 \leq i \leq t$. Then, for each $x' \in B_d^*(x)$, it verifies the existence of at least one occurrence of the l-mer y', in each N_i, $2 \leq i \leq t$, such that $d_H(x',y') = d$. If it exists, then x' is an (l,d)-motif of S and it is added into M, the output set.

Note that, we use the $B_d^*(x)$ instead $B_d(x)$ (as described in Section I, Definition 1.4), because the authors refer to a restricted neighborhood set, where only neighbors that has exactly d mismatches from the l-mers of interest are considered. The observation is that this algorithm, as well as the PMS1 and PMS2, is designed to search instances $\mathscr{M}'$ of the (l,d)-motif in the sample, such that the $d_H(\mathscr{M}',\mathscr{M})$ is *exactly* d instead *at most* d.

The space complexity was $O(tm^2)$, because only the set N was stored for each l-mer $x \in S_1$. The time complexity was $O(tm^2\mathcal{V}(l,d))$, proportional to the size of the $B_d^*(x)$ of l-mer $x \in S_1$ and size of N.

In experimental tests, the authors used a computer with 2.4 GHz processor and 1 GB RAM to execute PMSP. Table III summarizes the results obtained. In this table, for comparison, we also added results reported by Rajasekaran *et al.* (2005) [63] for PMS2 using a computer with 2.4 GHz processor and 1 GB RAM and results reported by Chin and Leung (2005) [12] for VOTING using a computer with 2.4 GHz processor and 512 MB RAM. In this table, '-' means that the algorithm uses too much memory in the instance, took too long or its time was not reported. Time in seconds s, minutes m and hours h.

We can see on Table III, that the PMSP was clearly more efficient than PMS2 and competitive when compared to VOTING. However, the improved memory management of the PMSP has made it able of solving the challenging instance (17,6).

TABLE III
Comparison of Runtimes for PMSP, VOTING and PMSP

(l,d)	PMS2	VOTING	PMSP
(9,2)	1.44 s	0.4 s	0.6 s
(11,3)	19.84 s	8.6 s	6.9 s
(13,4)	228.94 s	108 s	152 s
(15,5)	-	22 m	35 m
(17,6)	-	-	12 h

(19,7) Instance - PMSPrune (2007)

Davila, Balla and Rajasekaran [17], in 2007, presented the exact algorithm PMSPrune, reporting for the first time the solution of the challenging instance (19,7). The strategy used by PMSPrune was similar to that used by Davila *et al.* (2006) [19] for PMSP. It explored the d-neighborhood of all l-mers $x \in S_1$ to find the motif. In this strategy, the $B_d(x)$ is generated using a $\mathcal{T}_d(x)$ tree in a branch and bound method to prune possible branches that did not list a motif. For each l-mer x' represented by a child node of $\mathcal{T}_d(x)$ with depth p, the algorithm incrementally calculates the value $D(x') = \bar{d}_H(x',S)$, which corresponds to a minimum d_H between x' and any l-mer of S_i, $2 \leq i \leq t$. Then $D(x')$ and p are used to decide to prune each descendant node of x'. When $D(x') < d$, x is included into M, the output set. When $D(x') - d > d - p$, all descendants of x' are pruned.

PMSPrune had theoretical time complexity similar to PMSP. However, due to the pruning strategies used in the search space, it is possible to obtain better results than PMSP. For comparison, the authors ran PMSP and PMSPrune algorithms in the same machine, a computer with 2.4 GHz processor and 1 GB RAM. Table IV summarizes the results. In this table, the results reported by Chin and Leung (2005) [12] for VOTING using a computer with 2.4 GHz processor and 512 MB RAM was also added. In this table, '-' means that the algorithm uses too much memory in the instance, took too long or its time was not reported. Time in seconds, minutes and hours.

TABLE IV
Comparison of Runtimes for VOTING, PMSP and PMSPrune

(l,d)	VOTING	PMSP	PMSPrune
(11,3)	8.6 s	6.9 s	5 s
(13,4)	108 s	152 s	53 s
(15,5)	22 m	35 m	9 m
(17,6)	-	12 h	69 m
(19,7)	-	-	9.2 h

According to the results in Table IV, we can see that PMSPrune was more efficient than PMSP and VOTING. We also highlight the solution reported for the challenging instance (19,7). Other comparative results by Sharma *et al.* (2011) [68] indicate that PMSPrune was more time efficient than other algorithms. However, years later of publication of PMSPrune, in 2011, Z. Chen *et al.* (2011) [11] found that PMSPrune contained bugs and failed to find the correct solution for certain instances. They pointed out that once the bug is fixed, PMSPrune will run significantly slower than the bugged version.

Davila, Balla, and Rajasekaran [18], in 2007, presented the exact algorithm Pampa. This algorithm extended and improved the ideas used by Davila *et al.* (2007) [17] for PMSPrune by exploring the search space generation more efficiently. The key idea was to use extended l-mers in the form of wildcards symbols representing any symbol of the alphabet Σ. For example, the 5-mer {A****} abbreviated all 5-mers started with the symbol 'A'. In this way, an adaptation of the computation of original $D(x')$ value of PMSPrune was performed, and the generation of $B_d(x)$ was made more efficiently. The theoretical time complexities was similar to the PMSPrune, however the authors reported that Pampa outperforms the PMSPrune by a factor close to 2 in instances such as (15,5), (17,6) and (19,7).

For comparison, the authors ran Pampa, PMSP by Davila *et al.* (2006) [19], PMSPrune by Davila *et al.* (2007) [17] in the same machine, a computer with 2.4 GHz processor and 1 GB RAM. Table V summarizes the results obtained. In this table, were also added the results reported by Chin and Leung (2005) [12] for VOTING using a computer with 2.4 GHz processor and 512 MB RAM. In this table, '-' means that the algorithm uses too much memory in the instance, took too long or its time was not reported. Time in seconds s, minutes m and hours h.

TABLE V
Comparison of Runtimes for VOTING, PMSP, PMSPrune and Pampa

(l,d)	VOTING	PMSP	PMSPrune	Pampa
(11,3)	8.6 s	6.9 s	5 s	4 s
(13,4)	108 s	152 s	53 s	35 s
(15,5)	22 m	35 m	9 m	5 m
(17,6)	-	12 h	69 m	40 m
(19,7)	-	-	9.2 h	5.75 h

According to the results in Table V, we can see that Pampa outperforms all algorithms for all tested instances.

MEME SUITE (2009)

Bailey *et al.* [4], in 2009, presented the web portal known as MEME SUITE[3], an online or standalone environment that provides tools for discovery and analysis of sequence motifs representing features such as DNA binding sites and protein interaction domains. Although it is not directly related to PMS and challenging instances, we highlight this work since MEME SUITE contains tools that are often the starting point for research involving motifs. One of these tools, used to discover multiple (ungapped) motifs in a set of sequences, is the MEME by Bayley *et al.* (1994) [2]. Another tool, for (gapped) motif search is GLAN2 by Frith *et al.* (2008) [29]. Also included in the portal are tools such as TOMTOM by Gupta *et al.* (2007) [32] to search for motifs in a database of known motifs, e.g., JASPAR by Sandelin *et al.* (2004) [66]. FIMO by Cuellar *et al.* (2011) [13], GLAM2SCAN by Frith *et al* (2008) [29] and MAST by Bailey and Gribskov (1998) [3] to search for occurrences of motifs in a sequence database and GOMO by Boden and Bailey (2008) [6] which provides associations between motifs and genes linked with one or more Genome Ontology (GO) terms.

(21,8) Instance - BitBased (2010)

Dasari, Desh and Zubair [16], in 2010, presented the exact parallel algorithm BitBased. This algorithm have used Open Multi-Processing[4] (OpenMP) directives to parallelize the code, reporting, for the first time, the solution of the challenging instance (21,8). This was not the first parallel approach to PMS. Other authors, e.g., Faheem (2010) [25] and Ho *et al.* (2009) [36], have investigated the use of parallelism, but without expressive results in challenging instances such as BitBased.

The algorithm used similar ideas to those proposed by Watterman *et al.* [77]. Briefly, the main idea of BitBased was to divide the problem solution into two phases. In the first phase, it constructed a vector of bits B_i of size $|\Sigma|^l$, for each sequence S_i, $1 \leq i \leq t$, which represented every possible d-neighbor of the l-mers of S_i. To fill this vector, each l-mer $x \in S_i$ is selected and its d-neighborhood generated, then each d-neighbor generated has its value filled with 1 in the respective position in the vector B_i. In the second phase, it performed a logical *AND* at the vector positions, in order to find the motifs in the common d-neighborhood.

With this approach the time complexity for the first and second phases was $O(tml\mathcal{V}(l,d) + t|\Sigma|^l)$. The space complexity was $O(t|\Sigma|^l)$, considering the t vectors used, and there is one of the biggest problems of this approach, even using binary coding. For example, in this approach, for instances (15,5), (17,6) and (19,7), it would be required 2.5 GB, 40 GB and 640 GB of memory, respectively, only for the vectors.

To reduce memory requirements, the authors described three mechanisms for implementing BitBased. The first, called *Incremental support*, was to first find the (l',d)-motifs, $l' \leq l$, then extend the (l',d)-motifs to find the (l,d)-motifs of S, in a similar idea to that described by Rajasekaran *et al.* (2005) [63]. The second, called *Iterative approach*, was similar to the idea used by Chin and Leung (2005) [12], which consisted in splitting up the l-mers in d-neighborhood into groups and processing them iteratively. Each group would store only l-mers that shared the same prefix of length p. With this approach, the bit vector could be reduced to $|\Sigma|^{l-p}$, adding $|\Sigma|^p$ processing iterations, one for each group. The third, called *Filtering*, was to consider initially only t' of t sequences, $t' \leq t$, then each motif found was considered a candidate that would be validated in the $t - t'$ remaining sequences.

BitBased combined the iterative approach, filtering, and incremental support as follows. In the beginning, it computes the value of l', $l' \leq l$, and the optimal value of t', $t' \leq t$, for instance (l',d). So, if memory was available

[3]More details about MEME-SUITE and others on line tools are available at http://meme-suite.org/.

[4]More information about OpenMP is available at https://www.openmp.org/.

for $t'|\Sigma|^{l'}$ vectors, it used a direct approach to generate the (l', d)-motifs candidates, apply the filter mechanism and include them in the temporary set M'. Otherwise, it used the iterative approach, performing the same process, but one group at a time, to generate the (l', d)-motifs candidates and include them in the temporary set M'. Then it used the set M' as input for the incremental support to get the (l, d)-motifs. The time and space complexity were respectively $O(tml\mathcal{V}(l', d) + t|\Sigma|^l)$ and $O(t|\Sigma|^{l'})$ for direct approach, and $O(tml\mathcal{V}(l', d)|\Sigma|^p + t|\Sigma|^l)$ and $O(t|\Sigma|^{l'-p})$ for the iterative approach.

The parallelization consisted in splitting up the input sequences to the available processors and processing them independently. In the tests performed, an X5550 Xeon computer with four quad-core processors of 2.67 GHz and a maximum of 1 GB RAM was used. For comparison, the authors used the results reported by Davila *et al.* (2007) [17] for PMSPrune using a computer with a 2.4 GHz processor and 1 GB RAM. Table VI summarizes the results obtained, where BitBased-n represents execution with n cores. In table, '-' means that the algorithm did not return the response after 10 hours execution. Time in seconds s, minutes m and hours h.

TABLE VI
Comparison of Runtimes for BitBased-n, using n cores, and PMSPrune

	(13,4)	(15,5)	(17,6)	(19,7)	(21,8)
BitBased-16	2 s	11 s	2.4 m	30.6 m	6.9 h
BitBased-8	2 s	16 s	3.5 m	42.3 m	-
BitBased-4	4 s	29 s	6.5 m	1.3 h	-
BitBased-1	9 s	1.8 m	2.6 m	4.7 h	-
PMSPrune	53 s	9 m	69 m	9.2 h	-

As we can see in Table VI, although they have used different computational environments, we can verify that BitBased, with a single processor, obtained better results than those reported for PMSPrune. We also highlight the solution of the instance (21,8) reported for the first time with the execution of BitBased-16.

Dasari, Desh and Zubair [16], in 2011, after the success obtained by BitBased, presented a similar version of the algorithm based on parallel programming using GPU devices. We will refer to this version of algorithm as gBitBased. According to the authors, although GPU utilization had already been employed for general motif search in heuristics such as Gibbs sampling (e.g., Yu and Xu, 2009 [79]) and MEME (e.g., Chen *et al.*, 2008 [10]), they reported that at the time there was no knowledge about a specific approach for the PMS problem.

The authors performed experimental tests using an NVidia Tesla S1070 computing system with 4 GPU devices containing 240 cores of 1.5 GHz each. The results reported with 1 and 4 GPU devices were compared with results reported by BitBased with 1 and 16 cores. Table VII summarizes this results and presents the speedup rates obtained in comparison to the results reported for BitBased using processors with 1 and 16 cores. In this table, '-' means that the algorithm did not return the response after 10 hours execution. Time in seconds s, minutes m and hours h.

TABLE VII
Comparison of Runtimes (gBitBased time) and Speedups for gBitBased related to Runtimes of BitBased (using 1 and 16 cores, respectively)

(l, d)	GPUs	gBitBased time	BitBased-1 speedup	BitBased-16 speedup
(15,5)	1	8.0 s	13.5	1.4
(15,5)	2	4.4 s	24.5	2.5
(15,5)	3	3.2 s	33.6	3.4
(15,5)	4	2.7 s	40.0	4.1
(17,6)	1	91.2 s	13.6	1.6
(17,6)	2	46.1 s	26.8	3.1
(17,6)	3	31.1 s	39.7	4.6
(17,6)	4	23.9 s	51.7	6.0
(19,7)	1	19.7 m	14.3	1.6
(19,7)	2	9.9 m	28.5	3.1
(19,7)	3	6.62 m	42.6	4.6
(19,7)	4	5.0 m	56.4	6.1
(21,8)	1	4.5 h	-	1.5
(21,8)	2	2.3 h	-	3.0
(21,8)	3	1.5 h	-	4.6
(21,8)	4	1.1 h	-	6.3

We can see in Table VII that gBitBased achieved better results than the previous BitBased for all instances, even though using a single GPU device. As reported by the authors, a single GPU device was 13 to 14 times faster than a single core. Four GPUs were 40-60 times faster than a single core, and 4 to 6 times faster than 16 cores.

(23,9) Instance - ILP (2011-2012)

Dinh, Rajasekaran and Kundeti [21], in 2011, presented the exact algorithm PMS5, reporting for the first time the solution of the challenging instance (23,9). PMS5 worked similarly to PMSPrune by Davila *et al.* (2007) [17]. It explored the d-neighborhood of l-mers $x \in S_1$ using $\mathcal{T}_d(x)$ to try to find the motifs. However, it innovated by considering properties of the tuple of l-mers $\{x, y, z\}$ from distinct sequences of the input sample to prune branches in $\mathcal{T}_d(x)$.

Briefly, PMS5 works as follows. For each l-mer $x \in S_1$, the l-mers $y \in S_{2k}$ and $z \in S_{2k+1}$, $1 \leq k \leq (t-1)/2$ are iteratively selected. Then, $B_d(x, y, z)$ is computed, by verifying that $x' \in B_d(x)$ was contained in $B_d(y) \cap B_d(z)$. If positive, $x' \in B_d(x, y, z)$ and is added to the temporary set Q. At the end of processing of the sequences S_{2k} and S_{2k+1}, the intersection $M' = M' \cap Q$, kept in M', only the common d-neighborhood between x and the l-mers of t sequences of the entry, and Q can be initialized. PMS5 still included a mechanism to interrupt the k iterations if the size of the set M' (potentially large) reached a threshold value, small enough to be directly verified in the set S. After processing all l-mers of S_1, M contains all the (l, d)-motifs of S. To compute $B_d(x, y, z)$ PMS5 uses integer linear programming (ILP) to express constraints on properties of the tuple $\{x, y, z\}$ using ten variables[5].

[5]For more details about variables and constraints of PMS5, we recommend reading Section 2.2.2 of Dinh *et al.*, 2011 [21].

PMS5 has space complexity $O(max|M'|)$ and time complexity, in the worst case, $O(tm^3d\mathcal{V}(l,d))$, that occurs in the extreme case if $x = y = z$, then $|B_d(x,y,z)| = |B_d(x)| = \mathcal{V}(l,d)$, and $|B_d(x,y,z)|$ would be computed at most $t/2(m-l+1)^3$ times. Although the time complexity of PMSPrune is better than that of PMS5, experimental tests performed by the authors demonstrated that PMS5 is more efficient.

The authors performed experimental tests with PMS5 using a Dual Core Pentium 2.4 GHz CPU with 3 GB RAM and Windows XP operating system. For comparison, the authors executed the algorithms Pampa by Davila *et al.* (2007) [18] and PMSPrune by Davila *et al.* (2007) [17] in the same machine. Table VIII summarizes the results obtained. In addition, we also added in this table the sequential runtimes reported by Dasari *et al.* (2010) [16] for BitBased-1 using only one core. In table, '-' means that the algorithm did not run for lack of computational resource or it took too long and was aborted. Time in seconds s, minutes m and hours h.

TABLE VIII
Comparison of Runtimes for PMS5, Pampa, PMSPrune and BitBased

	(13,4)	(15,5)	(17,6)	(19,7)	(21,8)	(23,9)
PMSPrune	45 s	10.2 m	78.7 m	15.2 h	-	-
Pampa	35 s	6 m	40 m	4.8 h	-	-
PMS5	117 s	4.8 m	21.7 m	1.7 h	9.7 h	54 h
BitBased-1	9 s	1.8 m	2.6 m	4.7 h	-	-

We can see from the results presented in Table VIII that the PMS5 outperforms the others sequential algorithms for all instances, except for (13,4). According to the authors, PMS5 took extra time to load the tables that store the ILP results, impacting on the time results of the instance (13,4). However, this time became negligible, as the instance size increased. We can also see that the PMS5 becomes more efficient than the execution of BitBased-1 from the instance (19,7) onwards. We also point out PMS5 because up to that moment it was the only one to report the solution of the challenging instance (23,9).

Bandyopadhyay, Sahni and Rajasekaran [5], in 2012, described the PMS6, an exact algorithm that extended the PMS5 by Dinh *et al.* (2011) [21] and improved, by more than twice, the runtime for the instance (23,9). The PMS6 differed only from PMS5 in the way it determined $B_d(x,y,z)$. While PMS5 computed $B_d(x,y,z)$, by independently calculating each pair (y,z), $y \in S_{2k}$, and $z \in S_{2k+1}$, the PMS6 grouped the l-mers $\{x,y,z\}$ into classes, gathering those whose d-neighbors shared similar computation process.

The PMS6 determined the motifs corresponding to tuple (x,y,z) using a two-step process. In the first step, all tuples (x,y,z) were partitioned into classes $C(n_1,...,n_5)$. To classify x,y,z, it first identified five types of situations that occurred between indices i, $1 \leq i \leq l$. For example, the type 1 was $x[i] = y[i] = z[i]$ and the type 2, $x[i] = y[i] \neq z[i]$. Then, according to the number of occurrences of each type, represented by n_1, n_2, n_3, n_4 and n_5, it decomposed into classes in the form $C(n_1,n_2,n_3,n_4,n_5)$. In the second step, it computed $B_d(C(n_1,...,n_5))$. This was done most efficiently because it simultaneously processed tuples(x,y,z) of the same class by sharing the required computational processing.

The theoretical times of PMS6 were similar to those of PMS5, but the authors reported that PMS6 was more efficient for all challenging instances solved. The same machine was used to perform these tests, a Dual Core Pentium 2.4 GHz CPU and 3 GB RAM with the Windows XP operating system. The results obtained are showed in Table IX. Time in seconds s, minutes m and hours h.

TABLE IX
Comparison of Runtimes for PMS6 and PMS5

	(13,4)	(15,5)	(17,6)	(19,7)	(21,8)
PMS5	117 s	4.8 m	21.7 m	1.7 h	9.7 h
PMS6	67 s	3.2 m	14 m	1.16 h	5.8 h

Dinh, Rajasekaran and Davila [20], in 2012, presented the exact algorithm qPMS7. This algorithm extended the ideas of d-neighborhood exploration used by Davila *et al.* (2007) [17] for PMSPrune, combined with the central idea of PMS5 by Dinh *et al.* (2011) [21]. qPMS7 considered two l-mers, $x \in S_i$ and $y \in S_j$, $1 \leq i \leq j \leq t$, instead of just one as in PMSPrune. Thus, the exploration method of d-neighborhood $B_d(x,y)$ has modified to use the tree $\mathcal{T}_d(x,y)$, where for each l-mer z represented by a node $n \in \mathcal{T}_d(x,y)$ with depth p, $p \leq d$, both $d_H(z,x) \leq p$ and $d_H(z,y) \leq p$.

Briefly, to find the motifs, for every pair $\{x,y\}$, the exploring process of $\mathcal{T}_d(x,y)$ is made as follows. In depth-first manner, starting from the node $n = (x',0)$, compute the value $D(x') = \bar{d}_H(x',S_k)$, which corresponded to minimum d_H between the l-mer represented by x' and any l-mer w of S_k, $1 \leq k \leq t$, $k \neq i$ and $k \neq j$. During computation, an l-mer w of an arbitrary sequence is eliminated if $d_H(x',w) + p > 2d$, n stores the number of survivors l-mers for each sequence. If the number of sequences in n with no surviving l-mers are greater than $t-2$, then n is discarded. If the number of sequences in n, whose $\bar{d}_H(x',S_k) \leq d$ is equal to $t-2$, then add the l-mer represented by n in M, the output set.

The qPMS7 had space complexity $O(tm^2)$ and time complexity $O(t^3m^2\mathcal{V}(l,d))$. The authors ran the qPMS7 using a Dual Core Pentium 2.4 GHz processor and 3 GB RAM with Windows XP operating system. For comparison the authors ran, using the same machine, the algorithms PMSPrune by Davila *et al.* (2007) [17], Pampa by Davila *et al.* (2007) [18], PMS5 by Dinh *et al.* (2011) [21] and PMS6 by Bandyopadhyay *et al.* (2012) [5]. Table X summarizes the results. In this table we also added the sequential results reported by Dasari *et al.* (2010) [16] for BitBased-1 using a single core. The value '-' means that the algorithm did not run for lack of computational resource or it took too long and was aborted. Time in seconds s, minutes m and hours h.

TABLE X
Comparison of Runtimes for qPMS7, PMSPrune, Pampa, PMS5 and PMS6

	(13,4)	(15,5)	(17,6)	(19,7)	(21,8)	(23,9)
PMSPrune	45 s	10.2 m	78.7 m	15.2 h	-	-
Pampa	35 s	6 m	40 m	4.8 h	-	-
PMS5	117 s	4.8 m	21.7 m	1.7 h	9.7 h	54 h
PMS6	67 s	3.2 m	14 m	1.1 h	5.8 h	-
qPMS7	47 s	2.6 m	11 m	0.9 h	4.3 h	24 h
BitBased-1	9 s	1.8 m	2.6 m	4.7 h	-	-

We can see in Table X that qPMS7 is more time efficient than other sequential algorithms. It also obtained better results than those reported for BitBased-1 from instance (19,7) onwards. We also highlight the solution of the instance (23,9) with runtime of 24 hours.

(26,11) Instance - The state of the art (2014)

In January 2014, Nicolae and Rajasekaran [55] presented the exact algorithm PMS8. This algorithm, in essence, represents the state of art for solution of challenging instances of PMS. It was the first to sequentially solve the challenging instance (25,10) with 15.45 hours of runtime and, with 48 cores, the challenging instance (26,11) in 46.9 hours of runtime.

The main innovation of PMS8 was the balanced use of phases using sample-driven and pattern-driven techniques and a new filter mechanism based on the total consensus distance (Cd), described later, which attempted to limit the search space in the sample in the sample-driven approach, and reduce the number of patterns generated in the pattern-driven approach.

Basically PMS8 was divided into three phases: sample-driven, pattern-driven and validation. In the sample-driven phase, tuples T_k containing k l-mers of distinct sequences were generated, $1 \leq k \leq t' \leq t$, where both k and t' were defined heuristically with $k = max(2, \sqrt{2(d+1)\log|\Sigma| - \log m})$ and $t' = min(t, k + t/4 - \log k)$. In the pattern-driven phase, (l,d)-motifs for the l-mers $\in T_k$ were enumerated, that we call as (l,d)-motifs candidates. In the validation phase, each (l,d)-motif candidate generated in the pattern-driven phase, was checked with the $t-k$ remaining sequences of the input to find the (l,d)-motifs. We will now address each of these phases in a little more detail.

In the sample-driven phase, PMS8 maintains a R table with all l-mers of the sequences, where each line R_i of R contains the l-mers of S_i, $1 \leq i \leq t$. Initially, the algorithm chooses the first l-mer $x \in R_i$, with $i = 1$, and puts it into T_k, then it filters all l-mers $y \in R_j$, where $i < j \leq t$, such that $d_H(x,y) > 2d$. Then it sorts the R_j lines of the table, in ascending order, according to the number of l-mers survivors in each row. If any line is empty then the l-mer x is replaced by the following one in R_i.

The process is repeated including in T_k an l-mer $x \in R_j$, $i \leq j \leq k \leq t'$, and applying the filter on each l-mer $y \in R_{j'}$, and by sorting the lines $R_{j'}$, where $j < j' \leq k \leq t'$, with the same criterion used previously. However, this time the filter process includes an additional criterion that also filters all y whose value of the total consensus distance $Cd(T_{k'}) > d|T_{k'}|$, where $T_{k'} = T_k \cup \{y\}$, $k' = k+1$, and $Cd(T_{k'}) = \sum_{i'=1}^{l} k' - MaxFreq(T_{k'}[i'])$. The idea of this additional filter was that if $MaxFreq(T_{k'}[i']) = v$, that is, if the maximum frequency of i'-th column of $T_{k'}$ was v, then the cumulative sum of $k'-v$ can be used to eliminate any y whose the cumulative sum exceeds $k' * d$ mismatches. Note that, considering that every l-mer can accumulate at most d mismatches, then the limit of mismatches between $T_{k'}$ and an arbitrary motif $\mathscr{M}$ is $k' * d$.

The pattern-driven phase starts when a tuple reaches size k. In this phase, simulated traversals in a virtual lexicographic tree $\mathcal{T}$ were used to enumerate the motifs of T_k. A motif was generated by adding the symbols of the branches in the traversal to a partial motif p, until p reached the length l. In the traversal, a branch labeled with symbol s could be pruned, considering $p' = p' \cup \{s\}$, if one of the situations occurred:

i) case $d_H(x', p') > d$, where $x' = x[1]...x[|p'|]$ for any $x \in T_k$;
ii) case $d_H(x', p') + d_H(y', p') + \alpha > 2d$, as in Eskin and Pevzner (2002) [23], where α was equal to the number of substitutions that occurred from position $|p'|+1$ until l, between any two l-mers x and y, such that $x \neq y$ and both $\{x, y\} \in T_k$, and $x' = x[1]...x[|p'|]$ and $y' = y[1]...y[|p'|]$;
iii) if $\sum_{i'=1}^{|p'|} Freq(p'[i']) + \sum_{i'=|p'|+1}^{l} |T_k| - MaxFreq(T_k[i']) > d|T_k|$, where $Freq(p'[i'])$ returns the frequency of i'-th symbol of p' and $MaxFreq(T_k[i'])$ returns the maximum frequency of the symbols of the i'-th column of T_k.

The validation phase begins when a traversal reached length l in the previous phase, thus building a candidate motif c. At this phase, it is checked whether there is at least one l-mer u in each S_i, $k < i \leq t$, such that $d_H(u,c) \leq d$. If so, c is (l,d)-motif of S and it is added in the output set M. After the end of the validation phase, the algorithm returns to pattern-driven phase and tries to find another path of length l. When there are no more traversals to be explored the pattern-driven phase would end and return to the sample-driven phase. When returning, the sample-driven phase attempts to replace the last l-mer added in T_k by the other, not already chosen, l-mer of the same sequence. If there are no more l-mers in this sequence, then it will remove the penultimate l-mer added (from the previous sequence) and attempts to replace by another one, using this approach successively until all options are explored and the algorithm is finalized.

The authors also reported other techniques used in PMS8, like the compression of l-mers, to work with integers instead of symbols, the pre-processing of the distance between the pair of l-mers of the input sample, the exploration of the cache locality in the form like the table R was allocated and maintained, and the use of parallel computing with Message Passing Interface (MPI) library[6].

[6]More information about MPI is available at https://www.mpi-forum.org/.

The space complexity was $O(tm)$, since only the R table must be stored in memory. The complexity of time was $O(tm^{2k}\mathcal{V}(l,d))$, in the worst case, considering the time complexity of the first phase $O(m^k)$, proportional to the maximum number of tuples of size k, the time complexity of the second phase $O(\mathcal{V}(l,d))$ proportional to common d-neighborhood of T_k, and the complexity $O(m(t-k))$ of the last phase where, in the worst case, each motif candidate was validated with each l-mer of the remaining $t-k$ sequences.

In experimental tests the authors ran PMS8 on a maximum of 48 cores in 4 nodes, from a cluster consisting of 64 nodes, each one equipped with a 12-core Intel Xeon X5650 2.66 GHz and 48 GB RAM. Table XI summarizes the results obtained by the PMS8-p with p cores. For comparison, we also added in this table, results reported by authors for qPMS7 by Dinh *et al.* (2012) [20] using a single core execution in the same machine. We also added the results reported by Dasari *et al.* (2010) [16] for BitBased-16 with 16 cores and by Dasari *et al.* (2010) [15] for gBitBased-4 using 4 GPUs. The value '-' means that the algorithm did not run for lack of computational resource or it took too long and was aborted. Time in seconds s, minutes m and hours h.

TABLE XI
Comparison of Runtimes Obtained by PMS8-p, with p Cores, qPMS7 [20] with One Core, and BitBased-16 with 16 Cores and gBitBased-4 Using 4 GPUs

	(13,4)	(15,5)	(17,6)	(19,7)
qPMS7	29 s	2.1 m	10.3 m	54.6 m
PMS8-1	7 s	48 s	5.2 m	26.6 m
PMS8-16	3 s	5 s	22 s	1.7 m
PMS8-32	2 s	4 s	12 s	52 s
PMS8-48	2 s	3 s	9 s	37 s
BitBased-16	2 s	11 s	2.4 m	30.6 m
gBitBased-4	2.7 s	23.9 s	5 m	1.1 h
	(21,8)	(23,9)	(25,10)	(26,11)
qPMS7	4.87 h	27.09 h	-	-
PMS8-1	1.64 h	5.48 h	15.45 h	-
PMS8-16	6.5 m	21.1 m	1.01 h	-
PMS8-32	3.3 m	10.7 m	30.4 m	-
PMS8-48	2.2 m	7.4 m	20.7 m	46.9h
BitBased-16	6.9 h	-	-	-
gBitBased-4	-	-	-	-

We can see on Table XI that the single core execution of PMS8 obtained better results than qPMS7 for all tested instances. Although, using different computational environment, PMS8 obtained significantly better runtimes than BitBased-16 and gBitBased-4 in most of the tests, both with 1 or 16 cores. The PMS8 still stood out since it was the only one that solve the instances (25,10) and (26,11).

In February 2014, Tanaka presented TraverStringRef [72], an exact algorithm that extended and added enhancements to the qPMS7 algorithm by Dinh *et al.* (2012) [20]. The author highlighted TraverStringRef as the first algorithm to solve the instance (25,10) with a single computer (without using multiple threads) using 15 hours or less. However, in January of the same year, Nicolae and Rajasekaran presented the PMS8 [55] which also solved the instance (25,10) with a single core and in a similar runtime. Apparently, Tanaka's work was written at the same time as the work of Nicolae and Rajasekaran, and by the fact that it did not contain references about PMS8, we believe that the author was not aware of it.

The key ideas behind TraverStringRef were based on the following observations of authors: i) the algorithms became more efficient if the size of the search trees are reduced; ii) it was necessary to reduce processing time to check whether a subtree could be pruned or not; and iii) if two root nodes (l-length substrings) were similar, then the search path in the tree would also be.

Based on these observations, improvements were included in the common d-neighborhood verification. One of these improvements was the incremental calculation of some information, which was previously fully preprocessed and stored. Unnecessary combinations of operations were also eliminated to avoid redundant processing, and elements in the lists of $\mathcal{T}_d$ were sorted to make the process of checking and pruning possible ramifications more efficient.

The time and space complexities with the changes were respectively $O(t^3m^2(m+\log t)\mathcal{V}(l,d))$ and $O(tm^2)$. To verify the effectiveness of the algorithm, the author ran experimental tests using a notebook with an Intel Core i7-3610QM 2.3 GHz processor and 16 GB RAM. Table XII summarizes the average times obtained in the experiments. For comparison, we added the average times reported by author for qPMS7 by Dinh *et al.* (2012) [20] using the same computational environment. The value '-' means that the algorithm took too long and was aborted. Time in seconds s, minutes m and hours h. Based on the results presented, the TraverStringRef has been shown to be 3 to 4 times faster than qPMS7. We also highlight the solution reported by TraverStringRef for the instance (25,10).

TABLE XII
Comparison of Runtimes for TraverStringRef and qPMS7

(l,d)	TraverStringRef	qPMS7
(13,4)	10.9 s	39.4 s
(15,5)	46.5 s	2.21 m
(17,6)	2.99 m	5.49 m
(19,7)	12.4 m	37.34 m
(21,8)	51.64 m	3.05 h
(23,9)	3.61 h	15.59 h
(25,10)	14.93 h	-

(30,13) Instance - State of art (2015)

Nicolae and Rajasekaran [56], in 2015, presented the exact algorithm qPMS9. We highlight this algorithm for having improved the runtime of all instances solved by PMS8 (by the same authors [55]) and for having reported, for the first time in the literature, the solution of the instances (28,12) and (30,13), using 48 cores for both instances.

The qPMS9 extended PMS8 algorithm including modifications to support the qPMS problem and two changes that made it in general more time efficient. The first change was in the heuristic formula used to calculate the value k, corresponding to the number of l-mers contained in the tuple. In qPMS9, $k = max(3, \sqrt{2(d)\log|\Sigma| - \log m + 4})$. The effects of this change are shown in Table XIII where the occurrences of (k, t') whose different values between qPMS9 and PMS8 are highlighted.

TABLE XIII
Comparison of the Heuristics Values Defined by PMS8 and qPMS9 to (k, t')

(l, d)	PMS8 (k, t')	qPMS9 (k, t')
(15,5)	(3,6)	**(4,7)**
(17,6)	(4,7)	(4,7)
(19,7)	(4,7)	**(5,7)**
(21,8)	(5,7)	(5,7)
(23,9)	(5,8)	**(5,7)**
(25,10)	(5,8)	**(6,8)**
(26,11)	(6,8)	(6,8)
(28,12)	(6,8)	(6,8)
(30,13)	(6,9)	**(6,8)**

The second change was in the sample-driven phase, in the way the lines in R table were ordered after the filtering process. While in PMS8, the R table was decreasingly ordered according to the number of surviving l-mers in each row, qPMS9, instead of considering only the number of surviving l-mers, calculated a value relative to the line, which would consider the weight of each surviving l-mer u. This weight was calculated by computing the total additional consensus distance (Cd) by including u in the tuple Cd($T \cup \{u\}$) - Cd(T). The tuples were ordered in decreasing order by the minimum additional Cd. If there were a draw in the minimum additional Cd, then the line containing the fewest surviving l-mers is prioritized.

The authors tested the qPMS9, using 48 cores in 4 nodes, from a cluster consisting of 104 nodes, each one equipped with 12 Intel Xeon X5650 2.66 GHz cores and 48 GB RAM. Table XIV summarizes the results obtained using 48 cores for the instances (26,11), (28,12) and (30,13) and only one core for the others. In this table we also added results reported by the authors for PMS8 by Nicolae and Rajasekaran (2014) [55] and TraverStringRef by Tanaka *et al.* (2014) [72]. The "-" value in table means that there is no information on the execution of these instances reported by the authors. Time in seconds s, minutes m and hours h.

As we can see in Table XIV, the PMS8 achieved a better result than the TraverStringRef for the most challenging instance (25,10) and competitive results for the others. The qPMS9 obtained better results than both in the same instances. We also highlight the solutions reported by qPMS9 for instances (28,12) and (30,13), both with 48 cores.

TABLE XIV
Comparison of Runtimes for qPMS9, PMS8 and TraverStringRef

(l, d)	TraverStringRef	PMS8	qPMS9
(13,4)	14 s	7 s	6 s
(15,5)	55 s	48 s	34 s
(17,6)	3.5 m	5.2 m	2.7 m
(19,7)	14.5 m	26.6 m	13.4 m
(21,8)	59.8 m	1.64 h	45.4 m
(23,9)	4.08 h	5.48 h	2.26 h
(25,10)	17.55 h	15.45 h	6.3 h
(26,11)	-	46.9 h	12.12 h
(28,12)	-	-	27.58 h
(30,13)	-	-	51.02 h

Recent improvements (2016-2018)

Since the qPMS9 was presented in 2015, there has still been no publication reporting the solution of larger challenging instances such as instance (32,14). There are also no reports of algorithms capable of sequentially solving the instance (26,11) and larger. In this period, from 2016 to 2018, we only found publications that report extensions of qPMS9 making it more efficient in terms of execution time for the already reported instances. These publications are presented below.

Kazemian, Fazlali, Katanforoush and Rezvani [40], in 2016, presented qPMS9-OMP a parallelized version of qPMS9 by Nicolae and Rajasekaran (2015) [56] using the OpenMP library. The main contributions were the creation of a dynamic method of planning threads and the parallelization of iterative loops in the source code that, together, outperformed the original algorithm.

TABLE XV
Performance Comparison Between the qPMS9 and qPMS9-OMP

(l, d)	qPMS9-1	qPMS9-12	qPMS9-OMP-12
(13,4)	6.9 s	2.2 s (3.1)	2 s (3.4)
(15,5)	39.8 s	5.5 s (7.2)	5.1 s (7.8)
(17,6)	3.24 m	34.63 s (6.0)	18.8 s (10.5)
(19,7)	15.71 m	2.42 m (6.5)	1.35 m (11.5)
(21,8)	54.19 m	9.53 m (5.7)	4.56 m (11.8)
(23,9)	2.86 h	58.96 m (3.0)	14.49 m (11.9)
(25,10)	8.06 h	2.74 h (2.9)	40.63 m (11.9)

The results reported by authors are shown in Table XV. The qPMS9-1 column displays the results obtained with the sequential execution of qPMS9 and the column qPMS9-MPI-12 displays the results obtained with 12 cores. The qPMS9-OMP-12 column displays the results obtained with the execution of the OpenMP version of qPMS9, qPMS9-OMP, also using 12 cores. The speedup obtained in relation to sequential runtime of qPMS9 is shown alongside the runtimes of the algorithms qPMS9-12 and qPMS9-OMP-12. All executions were performed in the same machine, a computer with 12 Intel 2.8 GHz cores and 32 GB RAM. The time is in seconds s, minutes m and hours h. As we can see on Table XV, the qPMS9-OMP-12 obtained better speedups than qPMS9-12 for all instances.

Xiao, Pal and Rajasekaran [78], in 2016, presented the randomized algorithm qPMS10. The idea of the

algorithm was firstly to solve a PMS subproblem with a random sample of the input with t' sequences, where $t' = t * \epsilon_1$, $\epsilon_1 \in (0, 1]$, and a known deterministic algorithm was used as subroutine. Then, the output of the subroutine, which reported (l, d)-motifs of subproblem with t', requires was filtered to find the solution of the original problem. The authors used the qPMS9 by Nicolae and Rajasekaran (2015) [56] as a subroutine of qPMS10.

In tests carried out, they reported better results of qPMS10 then those obtained by qPMS9, both algorithms were tested in same machine, a MacBook Air computer core with Intel Core i5 1.6 GHz processor and 8 GB RAM. Table XVI summarizes the tests results. All experiments were performed using a MacBook Air computer core with Intel Core i5 1.6 GHz processor and 8 GB RAM. The time is in minutes m and hours h.

TABLE XVI
Speedups obtained by qPMS10 in relation of qPMS9

(l, d)	qPMS9	qPMS10 (speedup)
(17,6)	2.9 m	t'=13 (1.13)
(19,7)	14.4 m	t'=11 (1.18)
(21,8)	50.2 m	t'=12 (1.20)
(23,9)	2.49 h	t'=11 (1.15)

As we can see on Table XVI, the qPMS10 obtained speedups in relation to qPMS9 varying between 1.13 and 1.20, using values between 11 and 13 for t'.

Final comments

In this section, we presented the main results found in the literature for PMS. We focus primarily on the exact algorithms that have achieved success with challenging instances. We try, as far as possible, to provide the reader with a good level of detail of these algorithms and in the results obtained by them.

According to our knowledge (disregarding algorithms designed in specific architectures or hardwares), until the moment of writing this material, we believe that qPMS9 is the most efficient exact sequential and parallel algorithm for PMS. In our point of view, the state of the art is essentially represented by the algorithms PMS8 and its extension qPMS9.

In Table XVII, we present the main results summary for the PMS. Additional information about occurrences in table has been described in the previous subsections. For specific details, the original work, indexed in the last column, could be consulted.

TABLE XVII
Main Results for the PMS

(l, d)	Year	Description	Reference
(6,2)	1975	Visual observation.	Pribnow [60]
-	1982	First exact algorithm. Queen Algorithm.	Queen *et al.* [62]
-	1985	Waterman Algorithm.	Waterman *et al.* [77]
-	1998	SPELLER	Sagot [65]
(15,4)	2000	Origin of PMS. Challenge Problem. WINNOWER. SP-STAR.	Pevzner and Sze [58]
(14,4)	2001	Probabilistic analysis. Challenging Instances. PROJECTION	Buhler and Tompa [8]
-	2003	NP-completeness class.	Evans *et al.* [24]
(9,2)	2003	MULTIPROFILER.	Keich *et al.* [41]
(15,5)	2005	VOTING.	Chin and Leung [12]
(17,6)	2006	PMSP.	Davila *et al.* [19]
(19,7)	2007	PMSPrune.	Davila *et al.* [17]
(21,8)	2010	BitBased	Dasari *et al.* [16]
(23,9)	2011	PMS5	Dinh *et al.* [21]
(25,10) (26,11)	2014	PMS8	Nicolae and Rajasekaran [55]
(28,12) (30,13)	2015	qPMS9	Nicolae and Rajasekaran [56]
(32,14)		Not reported yet.	

III. Discussion

ALTHOUGH the planted (l, d)-motifs search problem (PMS) is NP-complete and so there is no exact efficient algorithm to solve it, several (exact and heuristics) algorithms have been proposed to solve it in acceptable time for practical-sized instances. We have focused the discussion, as in historical review, on the exact algorithms for PMS and on a particular set of instances that are probabilistically more difficult to solve. In the literature these instances are called *challenging instances*. Table XVIII presents some of these instances, considering the alphabet $\Sigma = \{A, C, G, T\}$ and a input sample S_p. To compare the instances with each other, table shows for each instance, the total number of l-length patterns (or l-mers), and the size of d-neighborhood, $|Bd|$ of a single l-mer.

TABLE XVIII
Comparison Between Challenging Instances for PMS

(l, d)	4^l	$\|Bd\|$
(15,5)	1.073.741.824	853.570
(17,6)	17.179.869.184	10.738.204
(19,7)	274.877.906.944	133.145.104
(21,8)	4.398.046.511.104	1.634.428.162
(23,9)	70.368.744.177.664	19.920.393.772
(25,10)	1.125.899.906.842.624	241.519.496.656
(26,11)	4.503.599.627.370.496	1.755.693.024.424
(28,12)	72.057.594.037.927.900	20.905.591.546.804
(30,13)	1.152.921.504.606.850.000	248.678.050.515.949
(32,14)	18.446.744.073.709.600.000	2.955.978.004.693.310

Several algorithms and techniques have been employed in attempts to solve challenging instances. These techniques varied according to the size of the instance and the computational resources available at the time.

Algorithms based on index strategy, e.g, Queen *et al.* (1982) [62], Waterman *et al.* (1984) [77], Chin and Leung (2005) [12], Dasari *et al.* (2010) [16], construct arrays to index a set of d-neighbors of a selected subset of the input, then explore the d-neighborhood of this subset of the input and store in these arrays the number of occurrences of these d-neighbors. If a d-neighbor occurs in all sequences of the subset, then this d-neighbor is a motif of this subset. If this subset contains all sequences of the input, then the motif has been found,

otherwise it is a motif candidate that should be checked with the remaining input sequences. This technique was successfully used for small instances, however as the size of the instance increased, the amount of memory required to store these arrays became insufficient, even with the increase in memory supply that occurred over time. Trying to attenuate this problem, techniques such as subdivision of search space, e.g., Chin and Leung (1984) [12], and binary encoding, e.g., Dasari *et al.* (2010) [16], were used. These techniques were successful once they enabled the processing of instances that were previously not possible, but quickly proved to be insufficient with larger instances.

Other strategies were parallelized versions of known sequential algorithms. These strategies were immediately successfully for PMS, by they made the original algorithms more efficient, e.g., Dasari *et al.* (2010) [16], Faheem (2010) [25], Dasari *et al.* (2010) [15], Bandyopadhyay *et al.* (2012) [5]. However, for larger instances, the main idea of the sequential algorithms needed to be reformulated, and then again it was necessary to design new ones.

Historically, two approaches have been widely employed by PMS algorithms, the pattern-driven and sample-driven approaches. In pattern-driven all l-length patterns were considered as potential motifs, then each pattern should be generated and checked in the input sample to verify whether was a motif, e.g., Pesole *et al.* (1992) [57], Staden (1989) [69], Tompa (1999) [74]. In sample-driven, motifs were searched directly in the sample, then all possible alignments of l-length substrings (the l-mers) of different sequences were considered as potential motif instances, then each alignment should be generated and checked whether it shared a motif. Some algorithms used both approaches together, selecting one or more l-mers of the sample, and generating only the patterns with maximum Hamming distance d (which we call of d-neighborhood) of these l-mers. The strategies employed for the selection of l-mers and the generation of patterns differed with the passage of time.

In the beginning, some algorithms generated the d-neighborhood of all t input sequences, considering l-mers of one sequence at a time, e.g, Queen *et al.* (1982) [62], Pesole *et al.* (1985) [30], Rajasekaran *et al.* (2005) [63], while others generated the d-neighborhood of $t/2$ sequences, considering l-mers of two sequences at a time, e.g., Davila *et al.* (2006) [19]. Strategies were also designed to generates only the patterns (d-neighbors) of the l-mers of the first sequence, e.g., Davila *et al.* (2007) [17], Davila *et al.* (2007) [18]. While others generated only common patterns of l-mers of two sequences at at time, e.g, Dinh *et al.* (2012) [20] and three sequences at a time, e.g., Dinh *et al.* (2011) [21], Bandyopadhyay *et al.* (2012) [5].

The state of the art algorithms, PMS8 by Nicolae and Rajasekaran [55] and qPMS9 by Nicolae and Rajasekaran [56] have innovated by not using a fixed number of sequences. Instead, they analyzed the size of the input instance and then, heuristically, decided the number of sequences. In addition, they used strategies to sort and choose sequences that could be processed faster. In these strategies, sorting criteria were used to carefully prioritize the sequences that could be processed faster. Another innovation was the creation of a filtering mechanism that analyzes the symbols frequency of the selected l-mers of these sequences to eliminate possible combinations of tuples of l-mers that do not share a motif.

Recent publications, e.g., as Kazemian *et al.* (2012) [40] and Xiao *et al.* (2012) [78], demonstrated that it is possible to improve the current algorithms so that new challenging instances for PMS can be solved. It is difficult to answer how much these algorithms must be improved to be able to locate a pattern out of eighteen quintillion of patterns, as presented in Table XVIII for the instance (32,14).

The objective of this work is to serve as a starting point for the study of the PMS problem and the existing solutions, aiming to help the research of new solutions for challenging instances not yet reported in the literature.

IV. Conclusion

In this work, we presented the PMS problem and instances considered more difficult to solve, the challenging instances. Then, we gathered and presented the main results found in the literature. The focus of the research were the publications, which at the time, presented successfully algorithms for the challenging instances. In this case, we focus mainly on exact algorithms and the techniques used by them. Finally, we discussed the evolution of these algorithms until the state of the art algorithm. In this context, we discussed about two approaches that have historically stood out. One was an index based approach and its main problem was the high memory requirement. It was continually improved with the addition of strategies that attempted to reduce memory usage. Another approach, tried to have a balanced use of sample-driven and pattern-driven approaches, and used filtering mechanisms to eliminate part of the search space and pattern generation. This approach has been successfully used by many algorithms that reported solutions to challenging instances, including the state of the art algorithms.

Nowadays, there are still limitations to solve larger instances and new enhancements are needed to solve them. We hope this work could be used as an initial support to help future research in this field.

Acknowledgments

We are thankful to Dr. Ricardo Linden for the precious suggestions on the final version of the manuscript.

Funding

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. NFA also has grants from CNPq (306624/2019-2) and Fundect-MS (TOs 141/2016, 007/2015).

References

[1] Mostafa M Abbas, Mohamed Abouelhoda, and Hazem M Bahig. "A hybrid method for the exact planted (l, d)-motif finding problem and its parallelization". In: *BMC bioinformatics* 13.17 (2012). Article number: S10.

[2] Timothy L Bailey and Charles Elkan. "Fitting a mixture model by expectation maximization to discover motifs in bipolymers". In: *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*. Menlo Park, California: AAAI Press, 1994, pp. 28–36.

[3] Timothy L. Bailey and Michael Gribskov. "Combining evidence using p-values: application to sequence homology searches." In: *Bioinformatics (Oxford, England)* 14.1 (1998), pp. 48–54.

[4] Timothy L. Bailey et al. "MEME Suite: tools for motif discovery and searching". In: *Nucleic Acids Research* 37.suppl_2 (May 2009), W202–W208. ISSN: 0305-1048. DOI: 10.1093/nar/gkp335.

[5] Shibdas Bandyopadhyay, Sartaj Sahni, and Sanguthevar Rajasekaran. "PMS6: A Fast Algorithm for Motif Discovery". In: *Proceedings of the 2012 IEEE 2Nd International Conference on Computational Advances in Bio and Medical Sciences*. ICCABS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–6. ISBN: 978-1-4673-1320-9. DOI: 10.1109/ICCABS.2012.6182627. URL: https://doi.org/10.1109/ICCABS.2012.6182627.

[6] Mikael Boden and Timothy L Bailey. "Associating transcription factor-binding site motifs with target GO terms and target genes". In: *Nucleic acids research* 36.12 (2008), pp. 4108–4117.

[7] Alvis Brazma et al. "Approaches to the automatic discovery of patterns in biosequences". In: *Journal of computational biology* 5.2 (1998), pp. 279–305.

[8] Jeremy Buhler and Martin Tompa. "Finding motifs using random projections". In: *proceedings of the 5th Intel Conference on Computational Molecular Biology*. 2001, pp. 22–25.

[9] Alexandra M Carvalho et al. "A highly scalable algorithm for the extraction of cis-regulatory regions". In: *Proceedings Of The 3rd Asia-Pacific Bioinformatics Conference*. World Scientific. 2005, pp. 273–282.

[10] Chen Chen et al. "GPU-MEME: Using graphics hardware to accelerate motif finding in DNA sequences". In: *IAPR International Conference on Pattern Recognition in Bioinformatics*. Springer. 2008, pp. 448–459.

[11] Zhi-Zhong Chen and Lusheng Wang. "Fast exact algorithms for the closest string and substring problems with application to the planted (l, d)-motif model". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8.5 (2011), pp. 1400–1410.

[12] Francis YL Chin and Henry CM Leung. "Voting algorithms for discovering long motifs." In: *APBC*. 2005, pp. 261–271.

[13] Gabriel Cuellar-Partida et al. "Epigenetic priors for identifying active transcription factor binding sites". In: *Bioinformatics* 28.1 (2011), pp. 56–62.

[14] Modan K Das and Ho-Kwok Dai. "A survey of DNA motif finding algorithms". In: *BMC bioinformatics* 8.7 (2007), p. 1.

[15] N Dasari, Ranjan Desh, and M Zubair. "Solving planted motif problem on GPU". In: *International Workshop on GPUs and Scientific Applications, GPUScA 2010, Vienna, Austria, September 11*. 2010.

[16] Naga Shailaja Dasari, Ranjan Desh, and Mohammad Zubair. "An efficient multicore implementation of planted motif problem". In: *High Performance Computing and Simulation (HPCS), 2010 International Conference on*. IEEE. 2010, pp. 9–15.

[17] Jaime Davila, Sudha Balla, and Sanguthevar Rajasekaran. "Fast and practical algorithms for planted (l, d) motif search". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 4.4 (2007), pp. 544–552.

[18] Jaime Davila, Sudha Balla, and Sanguthevar Rajasekaran. *Pampa: An improved branch and bound algorithm for planted (l, d) motif search*. Tech. rep. Department of Computer Science and Engineering, University of Connecticut, Storrs, CT, 2007.

[19] Jaime Davila, Sudha Balla, and Sanguthevar Rajasekaran. "Space and time efficient algorithms for planted motif search". In: *International Conference on Computational Science*. Springer. 2006, pp. 822–829.

[20] Hieu Dinh, Sanguthevar Rajasekaran, and Jaime Davila. "qPMS7: A Fast Algorithm for Finding (l,d)-Motifs in DNA and Protein Sequences". In: *PLOS ONE* 7.7 (July 2012), pp. 1–8. DOI: 10.1371/journal.pone.0041425.

[21] Hieu Dinh, Sanguthevar Rajasekaran, and Vamsi K Kundeti. "PMS5: an efficient exact algorithm for the (l, d)-motif finding problem". In: *BMC bioinformatics* 12.1 (2011), p. 410.

[22] Laurent Duret and Philipp Bucher. "Searching for regulatory elements in human noncoding sequences". In: *Current opinion in structural biology* 7.3 (1997), pp. 399–406.

[23] Eleazar Eskin and Pavel A Pevzner. "Finding composite regulatory patterns in DNA sequences". In: *Bioinformatics* 18.suppl 1 (2002), S354–S363.

[24] Patricia A Evans, Andrew D Smith, and H Todd Wareham. "On the complexity of finding common approximate substrings". In: *Theoretical Computer Science* 306.1 (2003), pp. 407–430.

[25] HM Faheem. "Accelerating motif finding problem using grid computing with enhanced brute force". In: *Advanced Communication Technology (ICACT),*

2010 The 12th International Conference on. Vol. 1. IEEE. 2010, pp. 197–202.

[26] Alexander V Favorov et al. "A Gibbs sampler for identification of symmetrically structured, spaced DNA motifs with improved estimation of the signal length". In: *Bioinformatics* 21.10 (2005), pp. 2240–2245.

[27] Maria Federico et al. "An efficient algorithm for planted structured motif extraction". In: *Proceedings of the 1st ACM workshop on Breaking frontiers of computational biology*. ACM. 2009, pp. 1–6.

[28] Kornelie Frech, Günter Herrmann, and Thomas Werner. "Computer-assisted prediction, classification, and delimitation of protein binding sites in nucleic acids". In: *Nucleic acids research* 21.7 (1993), pp. 1655–1664.

[29] Martin C. Frith et al. "Discovering Sequence Motifs with Arbitrary Insertions and Deletions". In: *PLOS Computational Biology* 4.5 (May 2008), pp. 1–12. DOI: 10.1371/journal.pcbi.1000071.

[30] David J Galas, Mark Eggert, and Michael S Waterman. "Rigorous pattern-recognition methods for DNA sequences: Analysis of promoter sequences from Escherichia coli". In: *Journal of molecular biology* 186.1 (1985), pp. 117–128.

[31] Debraj GuhaThakurta and Gary D Stormo. "Identifying target sites for cooperatively binding factors". In: *Bioinformatics* 17.7 (2001), pp. 608–621.

[32] Shobhit Gupta et al. "Quantifying similarity between motifs". In: *Genome biology* 8.2 (2007). Article number: R24.

[33] Fatma A Hashim, Mai S Mabrouk, and Walid Al-Atabany. "Review of different sequence motif finding algorithms". In: *Avicenna journal of medical biotechnology* 11.2 (2019), pp. 130–148.

[34] Jacques van Helden, Alma F Rios, and Julio Collado-Vides. "Discovering regulatory elements in non-coding sequences by analysis of spaced dyads". In: *Nucleic Acids Research* 28.8 (2000), pp. 1808–18.

[35] Gerald Z Hertz and Gary D. Stormo. "Identifying DNA and protein patterns with statistically significant alignments of multiple sequences." In: *Bioinformatics* 15.7 (1999), pp. 563–577.

[36] Eric S Ho, Christopher D Jakubowski, and Samuel I Gunderson. "iTriplet, a rule-based nucleic acid sequence motif finder". In: *Algorithms for Molecular Biology: AMB* 4.14 (2009).

[37] Ellis Horowitz, Sartaj Sahni, and Sanguthevar Rajasekaran. *Computer algorithms*. New York, NY, USA: W.H.Freeman Press, 1998.

[38] Jianjun Hu, Bin Li, and Daisuke Kihara. "Limitations and potentials of current motif discovery algorithms". In: *Nucleic acids research* 33.15 (2005), pp. 4899–4913.

[39] Chao-Wen Huang, Wun-Shiun Lee, and Sun-Yuan Hsieh. "An improved heuristic algorithm for finding motif signals in DNA sequences". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 8.4 (2011), pp. 959–975.

[40] Fazeleh Sadat Kazemian et al. "Parallel implementation of quorum planted (l, d) motif search on multi-core/many-core platforms". In: *Microprocessors and Microsystems* (2016).

[41] Uri Keich and Pavel A Pevzner. "Finding motifs in the twilight zone". In: *Proceedings of the sixth annual international conference on Computational biology*. ACM. 2002, pp. 195–204.

[42] Uri Keich and Pavel A. Pevzner. "Subtle motifs: defining the limits of motif finding algorithms". In: *Bioinformatics* 18.10 (2002), pp. 1382–1390.

[43] Lawrence J Korn, Cary L Queen, and Mark N Wegman. "Computer analysis of nucleic acid regulatory sequences". In: *Proceedings of the National Academy of Sciences* 74.10 (1977), pp. 4401–4405.

[44] J Kevin Lanctot et al. "Distinguishing string selection problems". In: *Information and Computation* 185.1 (2003), pp. 41–55. ISSN: 0890-5401.

[45] J. Kevin Lanctot et al. "Distinguishing String Selection Problems". In: *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '99. Baltimore, Maryland, USA: Society for Industrial and Applied Mathematics, 1999, pp. 633–642. ISBN: 0-89871-434-6.

[46] Charles E Lawrence and Andrew A Reilly. "An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences". In: *PROTEINS: Structure, Function and Genetics* 7.1 (1990), pp. 41–51.

[47] Charles E Lawrence et al. "Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment". In: *SCIENCE-NEW YORK THEN WASHINGTON*- 262 (1993), pp. 208–214.

[48] Shoudan Liang, Manoj Pratim Samanta, and BA Biegel. "cWINNOWER algorithm for finding fuzzy DNA motifs". In: *Journal of bioinformatics and computational biology* 2.01 (2004), pp. 47–60.

[49] Andrei Lihu and Ştefan Holban. "A review of ensemble methods for de novo motif discovery in ChIP-Seq data". In: *Briefings in bioinformatics* 16.6 (2015), pp. 964–973.

[50] Bingqiang Liu et al. "An algorithmic perspective of de novo cis-regulatory motif finding based on ChIP-seq data". In: *Briefings in Bioinformatics* 19.5 (2018), pp. 1069–1081. DOI: 10.1093/bib/bbx026. URL: http://dx.doi.org/10.1093/bib/bbx026.

[51] Xiaole Liu, Douglas L Brutlag, and Jun S Liu. "BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes". In: *Biocomputing 2001*. New Jersey, NJ, USA: World Scientific, 2000, pp. 127–138.

[52] Laurent Marsan and Marie-France Sagot. "Extracting structured motifs using a suffix tree - Algorithms and application to promoter consensus identification". In: *Proceedings of the fourth annual international conference on Computational molecular biology*. ACM. 2000, pp. 210–219.

[53] Satarupa Mohanty and Suneeta Mohanty. "Genetic Algorithm-Based Motif Search Problem: A Review". In: *Smart Intelligent Computing and Applications*. Springer, 2020, pp. 719–731.

[54] Marius Nicolae. "Data Structures and Algorithms for the Identification of Biological Patterns". PhD thesis. University of Connecticut - Storrs, 2016.

[55] Marius Nicolae and Sanguthevar Rajasekaran. "Efficient sequential and parallel algorithms for planted motif search". In: *BMC bioinformatics* 15.1 (2014). Article number: 34.

[56] Marius Nicolae and Sanguthevar Rajasekaran. "qPMS9: An efficient algorithm for quorum planted motif search". In: *Scientific reports* 5 (2015).

[57] Graziano Pesole et al. "WORDUP: an efficient algorithm for discovering statistically significant patterns in DNA sequences". In: *Nucleic Acids Research* 20.11 (1992), pp. 2871–2875.

[58] Pavel A Pevzner, Sing-Hoi Sze, et al. "Combinatorial approaches to finding subtle signals in DNA sequences". In: *International Conference on Intelligent Systems for Molecular Biology, ISMB*. Vol. 8. 2000, pp. 269–278.

[59] Nadia Pisanti et al. "RISOTTO: Fast extraction of motifs with mismatches". In: *Latin American Symposium on Theoretical Informatics*. Springer. 2006, pp. 757–768.

[60] David Pribnow. "Nucleotide sequence of an RNA polymerase binding site at an early T7 promoter". In: *Proceedings of the National Academy of Sciences* 72.3 (1975), pp. 784–788.

[61] Alkes Price, Sriram Ramabhadran, and Pavel A Pevzner. "Finding subtle motifs by branching from sample strings". In: *Bioinformatics* 19.suppl_2 (2003), pp. ii149–ii155.

[62] Cary Queen, Mark N Wegman, and Laurence Jay Korn. "Improvements to a program for DNA analysin: a procedure to find homologies among many sequences". In: *Nucleic acids research* 10.1 (1982), pp. 449–456.

[63] Sanguthevar Rajasekaran, Sudha Balla, and C-H Huang. "Exact algorithms for planted motif problems". In: *Journal of Computational Biology* 12.8 (2005), pp. 1117–1128.

[64] Mikhail A. Roytberg. "A search for common patterns in many sequences". In: *Computer applications in the biosciences: CABIOS* 8.1 (1992), pp. 57–64.

[65] Marie-France Sagot. "Spelling approximate repeated or common motifs using a suffix tree". In: *Latin American Symposium on Theoretical Informatics*. Springer. 1998, pp. 374–390.

[66] Albin Sandelin et al. "JASPAR: an open-access database for eukaryotic transcription factor binding profiles". In: *Nucleic acids research* 32.suppl_1 (2004), pp. D91–D94.

[67] Geir Kjetil Sandve and Finn Drabløs. "A survey of motif discovery methods in an integrated framework". In: *Biology direct* 1.1 (2006), p. 11.

[68] Dolly Sharma, Sanguthevar Rajasekaran, and Hieu Dinh. "An Experimental Comparison of PMSPrune and Other Algorithms for Motif Search". In: *arXiv preprint arXiv:1108.5217* (2011).

[69] Rodger Staden. "Methods for discovering novel motifs in nucleic acid sequences". In: *Bioinformatics* 5.4 (1989), pp. 293–298.

[70] Gary D Stormo. "DNA binding sites: representation and discovery". In: *Bioinformatics* 16.1 (2000), pp. 16–23.

[71] Sing-Hoi Sze, Songjian Lu, and Jianer Chen. "Integrating sample-driven and pattern-driven approaches in motif finding". In: *International Workshop on Algorithms in Bioinformatics*. Springer. 2004, pp. 438–449.

[72] Shunji Tanaka. "Improved Exact Enumerative Algorithms for the Planted (l, d)-Motif Search Problem". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 11.2 (2014), pp. 361–374.

[73] Rie Terada et al. "A type I element composed of the hexamer (ACGTCA) and octamer (CGCGGATC) motifs plays a role(s) in meristematic expression of a wheat histone H3 gene in transgenic rice plants". In: *Plant Molecular Biology* 27.1 (1995), pp. 17–26.

[74] Martin Tompa. "An exact method for finding short motifs in sequences, with application to the ribosome binding site problem." In: *International Conference on Intelligent Systems for Molecular Biology, ISMB*. Vol. 99. 1999, pp. 262–271.

[75] Martin Tompa et al. "Assessing computational tools for the discovery of transcription factor binding sites". In: *Nature biotechnology* 23.1 (2005), p. 137.

[76] Ngoc Tam L Tran and Chun-Hsi Huang. "A survey of motif finding Web tools for detecting binding site motifs in ChIP-Seq data". In: *Biology direct* 9.1 (2014), p. 4.

[77] MS Waterman, R Arratia, and DJ Galas. "Pattern recognition in several sequences: consensus and alignment". In: *Bulletin of mathematical biology* 46.4 (1984), pp. 515–527.

[78] Peng Xiao, Soumitra Pal, and Sanguthevar Rajasekaran. "qPMS10: A randomized algorithm for efficiently solving quorum Planted Motif Search problem". In: *Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on*. IEEE. 2016, pp. 670–675.

[79] Linbin Yu and Yun Xu. "A parallel gibbs sampling algorithm for motif finding on GPU". In: *2009 IEEE International Symposium on Parallel and*

Distributed Processing with Applications. IEEE. 2009, pp. 555–558.

[80] Yipu Zhang, Hongwei Huo, and Qiang Yu. "A heuristic cluster-based EM algorithm for the planted (l, d) problem". In: *Journal of bioinformatics and computational biology* 11.04 (2013). Articule ID 1350009.

[81] Yongqiang Zhang and Mohammed J Zaki. "EXMOTIF: efficient structured motif extraction". In: *Algorithms for Molecular Biology* 1.1 (2006). Article number: 21.

[82] Jianjun Zhou, Jorg Sander, and Guohui Lin. "Efficient composite pattern finding from monad patterns". In: *International journal of bioinformatics research and applications* 3.1 (2006), pp. 86–99.