# An Efficient Combinatorial Approach for Solving the DNA Motif Finding Problem

Filippo Geraci
filippo.geraci@iit.cnr.it

Marco Pellegrini
marco.pellegrini@iit.cnr.it

M. Elena Renda
elena.renda@iit.cnr.it

Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Via G. Moruzzi 1
56124, Pisa – ITALY

## Abstract

*The detection of an over-represented sub-sequence in a set of (carefully chosen) DNA sequences is often the main clue leading to the investigation of a possible functional role for such a subsequence. Over-represented substrings (with possibly local mutations) in a biological string are termed motifs. A typical functional unit that can be modeled by a motif is a Transcription Factor Binding Site (TFBS), a portion of the DNA sequence apt to the binding of a protein that participates in complex transcriptomic biochemical reactions. In the literature it has been proposed a simplified combinatorial problem called the* planted (l-d)-motif problem *(known also as the (l-d) Challenge Problem) that captures the essential combinatorial nature of the motif finding problem. In this paper we propose a novel graph-based algorithm for solving a refinement of the (l-d) Challenge Problem. Experimental results show that instances of the (l-d) Challenge Problem considered difficult for competing state of the art methods in literature can be solved efficiently in our framework.*

## 1 Introduction

The study of the insurgence and the evolution of genetic diseases has recently turned to the analysis of the transcriptional regulation modules in an effort to deepen our understanding [8, 1]. Transcription factors (TF), that are proteins mediating the complex machinery of gene expression, and transcription factor binding sites (TFBS), the specific loci onto the DNA strings onto which TF bind, are among the most basic concepts in transcriptomic research. In order to abstract from the specific biochemical properties it is often used the more generic idea of "motif" as an over-represented substring with the potential of being a candidate TFBS. In this paper we deal with the "ab initio" motif-finding problem that takes as input just a set of DNA strings and produces the set of over-represented motifs.

Designing algorithms for "ab initio" detection of motifs in biological sequences is a popular activity; indeed recent surveys estimate that more than 100 methods (and variants) have been proposed so far (see, for instance, [5, 7, 13, 17]), assessing the performance of such tools on a common platform and a well designed benchmark data sets (mixing both synthetic and natural) is a sub-field that just started to be developed [10, 16, 19]. The problem is complex both from the biological point of view and from the algorithmic one since many different characterizations of what constitute a "motif" are possible and the biochemical machinery of transcription is still being investigated.

**The *(l-d) Challenge Problem*.** In a seminal paper of 2000, Pevzner and Sze [14] proposed the use of the so called planted *(l-d)*-motif problem as benchmark for evaluating the relative performance of motif finding algorithms.

Given $t$ DNA sequences of length $n$, suppose there is a fixed but unknown nucleotide sequence $M$ (the motif consensus) of length $l$, and that each sequence contains variants (or instances) of $M$, derived from $M$ with at most $d$ points substitutions. The *(l-d)*-motif problem consists in determining $M$ and the location of the motif instances in each sequence.

This model simplifies most of the biochemical issues to concentrate on a fairly simple to state combinatorial problem that, however, captures the hardness of the problem. It is quite easy to generate synthetic data set in this model, allowing algorithmic choices to be validated (or more often discarded) in an early stage.

Among the combinatorial algorithms, a number of them use as key focus graph theoretic notions and reduce the motif finding problem to the problem of detecting dense subgraphs (cliques) in appropriate graphs. In this category:

WINNOWER and SP-STAR [14], MULTIPROFILER [9], cWINNOWER [11], MCL-WMW [3]. Some approaches mix graph model with a deeper use of string properties ([6], [15]).
Other approached are based on randomized projections (e.g. PROJECTION [4]), or on suffix trees (e.g. [12]).
In this paper we present a new combinatorial approach for solving refinements of the *(l-d) Challenge Problem*, a graph-based algorithm which, despite of its simplicity, is able to efficiently perform on classes of planted-motif instances considered hard to solve. It is well know that in mammals the TFBS for a given gene can be at a distance of many kilobytes from the start site of the regulated gene (see e.g. [8] and references therein), which implies that longer sequences must be processed. Motif finding in the DNA of less complex organisms (e.g. *Saccharomyces cerevisiae*) is easier since the distance of the TFBS to the start site is much shorter (a few hundred bases). When searching for motifs in a set of DNA strings the fraction of the strings in the set in which the motif is represented is also a critical parameter. Most method are efficient only when this fraction is 1, and suffer a combinatorial explosion as soon as this value decreases (in a typical motif finding experiment it is very hard to ensure that the faction is 1 or close to 1). Finally the length of the motif and the number of allowed degeneracies (i.e. base substitutions) are critical parameters of the problem. Most methods are scalable only in one (or two) of the critical parameters of the problem, while our method shows good empirical scalability properties in all of them. This positive property is attained by relaxing the classical definition of an *(l-d) Challenge Problem* aiming at good performance on an average problem instance.
The paper is organized as follows. Section 2 introduces the *(l-d) Challenge Problem* and Section 3 discusses our proposed modification leading to the key algorithmic idea; Section 4 reports a detailed description of the proposed algorithm; Section 5 gives the experimental methodology and evaluation of the experimental results; Section 6 concludes and outlines the guidelines for further developing this work.

## 2 The *(l-d) Challenge Problem*

**Definition 1** *Given:* $(i)$ *a set of strings* $\mathcal{S} = \{s_i\}_{i=1}^t$ *over an alphabet* $\Sigma$, *such that* $\forall i\ |s_i| = n$, $(ii)$ *a distance threshold* $d \geq 0$, *and* $(iii)$ *a length* $l$, $0 \leq d \leq l$, *the (l-d)-motif problem consists in finding a motif* $M \in \Sigma^l$ *such that in* $q$ *strings* $s_i$, $0 \leq q \leq t$, *exists a contiguous substring* $m_i$, *with* $|m_i| = l$ *and Hamming distance* $H(M, m_i) \leq d$.

This problem is solved by exhibiting the motif $M$ and the set of representative motifs $\{m_i\}_{i=1}^q$ (also called *instances* or *variants* of $M$) and the corresponding positions within the sequences.

In the *planted (l-d)*-motif problem, each $s_i \in \mathcal{S}, i \in [1, t]$, is generated by randomly choosing $n$ characters in $\Sigma$, and $M$ is chosen at random in $\Sigma^l$; finally, each $m_i, i \in [1, q]$, is obtained by mutating up to $d$ uniformly chosen at random characters in $M$. Each mutated motif $m_i$ is then embedded in 1 of the $q$ randomly selected strings of $S$, in a position uniformly chosen at random in $[1, n-l]$. Let us call the obtained planted set $\mathcal{S}^*_M$. The algorithms eligible for finding the planted motifs receive as input the set $\mathcal{S}^*_M$ and the parameters $l$, $d$, and $q$. Note that, depending on the choice of parameters $n, t, l, d, q$, there is the chance of having a motif $y \in \mathcal{S}$, $y \neq M$, satisfying the $(l, d, q)$-property. The *(l-d) Challenge Problem* was first introduced in [14] with $l = 15$, $d = 4$, $t = q = 20$ in sequences with length $n = 600$. Since year 2000, other more difficult instances with parameters set to $(17-6), (19-7), (21-7)$ [6] have been tackled.

Denoting with $E(l, d)$ the *expected number of distinct motifs in* $\mathcal{S}$ *satisfying the* $(l, d)$*-property* [4], it is possible to find an explicit analytic formula for computing the $E(l, d)$, as well as several typical values in tabular form [2]. In general, for fixed values of $n$, $l$, $q$ and $t$ there is a value $d'$ such that $E(l, d'-1)$ is negligible but $E(l, d')$ is not, thus, for the value $d = d'$, the probability of having spurious solutions in $\mathcal{S}$ is high [2]. As the number $t$ of sequences increases and the length $n$ grows, the number of noisy $l$ length spurious (or nearby) motifs is more likely to grow, increasing the difficulty of detecting the planted motif instances.

Most combinatorial algorithms in literature that aim at finding the solution $(M, m_1, \ldots, m_q)$ to the *(l-d) Challenge Problem* (as in definition 1) take the worst case approach and assume implicitly that for each $i \in [1..q]$, $H(M, m_i) = d$. Thus they assume that $d$ represent both the minimum and the maximum distance of some $m_i$ to $M$.

## 3 A refinement of the *(l-d) Challenge Problem*

In this section we propose a refinement in the definition of the *(l-d) Challenge Problem* in which we make explicit a new parameter $\delta$.

**Definition 2** *Let* $S = \{s_1, \ldots, s_t\}$ *be the set of* $t$ *DNA strings,* $l$ *the length of the consensus motif* $M$ *and* $d$ *the maximum number of mutations allowed in the planted instances of* $M$. *The* Motif Graph $G_{\leq h} = \{V, E_h\}$ *is a graph in which each substring (signal)* $x$ *of length* $l$ *in a given string* $s_k \in S$ *is represented by a vertex* $v_x$, *labeled with* $x$, *and the edge between* $v_x$ *and* $v_y$ *exists if the Hamming distance* $H(v_x, v_y)$ *between the corresponding signals* $x$ *and* $y$ *is not greater than* $h$[1]

[1]Given two vertex $v_x$ and $v_y$ labeled with the strings $x$ and $y$ respectively, we denote with $H(v_x, v_y)$ the Hamming distance between strings $x$ and $y$. More in general, with an abuse of notation, we will indicate with

For the sake of the explanation we denote with $G$ without subscripts the motif graph $G_{\leq 2d}$.
Let $M$ be a motif consensus and $(m_1, \ldots, m_k)$ all its variants in $V$, let $\delta$ be the minimum distance of $M$ to any string in $V$, and let $y = m_i$ be a string at distance $\delta$ to $M$ in $V$. By the triangular inequality $y$ is also at distance at most $d + \delta$ from any string in $(m_1, \ldots, m_q)$. Thus the set of strings $(y, m_1, ..m_q)$ will induce a clique in $G$ and a star centered in $y$ in $G_{\leq d+\delta}$.

We refine the standard definition of the *(l-d) Challenge Problem* by introducing a new parameter $\delta$ that allows us to have a spread of problem instances. When $\delta$ is relatively small compared to $d$ our algorithm takes advantage of this fact and it can solve instances that are harder in terms of the parameters $l$, $d$, $t$ and $n$. Because we refine the definition of an *(l-d) Challenge Problem* by introducing a new parameter $\delta$ direct comparisons between published results of competing methods and the one we present would be inappropriate. Our initial tests is section 5 show that (for $\delta = 0$) indeed our method does work over ranges of values for $l$, $d$, $q$, and $n$ for which other methods do not perform well. Future work includes a more extensive testing and comparisons among different methods on a common set of instances with varying values of the parameter $\delta$.

## 4 Our Approach

Our algorithm takes advantage of the fact that $0 \leq \delta \leq d$, and when $\delta \ll d$ we are dealing with an easier instance of the *(l-d) Challenge Problem*, for which we can gain in performance.

In a nutshell our algorithm computes a covering of the vertices of $G$ and $G_{\leq d+\delta}$ so to make sure that each cover set does contain the vertices of a $q$-clique in $G$ and at the same time the same cover set contains a matching star in $G_{\leq d+\delta}$. Finally given the candidate sets $m_1..., m_q$ we compute a candidate consensus string $M$ by majority vote on each character position (as in [18]).

According to the above model, the *(l-d) Challenge Problem* can be reduced to that of finding subgraphs of $G$ containing a clique. Note that, each string $s_i \in S$ could contains both a planted instance of the consensus motif $M$ and one or more spurious solutions; for this reason, in order to reach the quorum $q$, it is not sufficient that the clique has at least $q$ nodes.

### 4.1 The Algorithm

The algorithm takes as input the set $S$ of $t$ DNA strings of length $n$, and three parameters specifying $(i)$ the length $l$ of the motif to be searched, $(ii)$ the minimum number $q$ (the quorum) of sequences that contain the instances of $M$, and $(iii)$ the maximum number of mutations $d$ allowed in each instance of $M$. The algorithm returns, for each identified motif, $(i)$ the consensus string, and $(ii)$ the coordinates in the corresponding input string of each signal identifying an instance of the motif. The algorithm implicitly considers the motif graph $G$. For the sake of explanation, we assume that the edge between $v_x$ and $v_y$ in $G$ is blue if $H(v_x, v_y) \leq d$, while it is red if $H(v_x, v_y) \leq 2d$. Obviously, with this assumption, a blue edge is also red.

The algorithm consists of two main phases: $(i)$ the *motif graph clustering*, and $(ii)$ the *motif identification*.

In the first phase, the algorithm divides the nodes of $G$ into overlapping clusters, each of which is the subgraph of $G$ containing the set of nodes connected to the cluster seed. In the second phase, for each cluster the algorithm checks if the seed of the cluster -or a node linked to it with a blue edge- is the center of a star whose size is at least equal to the quorum. If such a star exists it represents a "valid" motif, *i.e.*, a consensus string with at least $q$ instances of it, thus the elements it contains are returned as output.

These two phases are described in detail in what follow.

**Motif Graph Clustering.** The algorithm maintains a list $L$ of all those nodes in $G$ not linked to a cluster seed by a blue edge. Thus, initially the list $L$ contains all the nodes in $G$. The algorithm iteratively:

1. initializes a new cluster $C_k$ by selecting a random node $v_i \in L$ as seed, and removes $v_i$ from $L$;
2. adds all the nodes directly connected to $v_i$ by a blue or red edge to $C_k$, and updates $L$ by removing the nodes connected to $v_i$ by a blue edge.

The procedure ends when $L$ is empty and the overlapping clustering of the motif graph, $C_G = \{C_1, \ldots, C_k\}$, is returned to the second phase. By exploiting the distribution of the edges in the motif graph, we can assert that if the cluster seed is an instance of the motif, then all the other instances of the motif will fall in the same cluster.

The complexity of this phase is $O(nk)$, where $n$ is the number of nodes in $G$, and $k$ is the -a priori not known- number of iterations. In the worst case, *i.e.*, when the motif graph has no edges, the computational cost of this procedure is $O(n^2)$, but typically it is much smaller.

**Motif Identification.** In the clustering of the motif graph, $C_G = \{C_1, \ldots, C_k\}$, each $C_i$ is a small subgraph of $G$ that could contain a motif. The algorithm analyzes all the clusters of $C_G$ to verify whether they contain a valid motif or not, and filters out those not containing a motif. Clearly, if a minimum quorum is required, all those clusters whose cardinality is less than the quorum $q$ can be safely discarded.

the same symbol the node of the graph and the corresponding string, when it is clear from the context.

Denote with $C_G^*$ the subset of $C_G$ such that a cluster $C_i \in C_G^* \iff |C_i| \geq q$. Informally, for each cluster in $C_G^*$, the algorithm extracts a candidate consensus string $M$, and removes all the signals with Hamming distance greater than $d'$ from $M$, $d' = d + \delta, \delta \geq 0$; if each resulting cluster still has cardinality greater than or equal to the quorum $q$, then it is eligible to identify a valid motif, otherwise it is discarded.

As we already mentioned, we know that if a certain cluster seed is an instance of the motif all the other instances are present in the cluster. Thus, according with the definition of *(l-d)*-motif, we can exploit this fact by observing that the center of the star that identify the motif must be either the cluster seed or a node at distance at most $d'$ from it. Let $\hat{C} = \{\hat{c}_1, \ldots, \hat{c}_k\}$ be the subset of a cluster $C$ containing all the elements at distance at most $d'$ from the seed. For each $\hat{c}_i$, let $M(\hat{c}_i)$ the set of nodes of the clusters at distance at most $d'$ from $\hat{c}_i$. If $M(\hat{c}_i)$ reaches the quorum, it represents all the instances of a valid motif. This step requires $|\hat{C}||C|$ distance computations. Even if this is $O(|C|^2)$ in the worst case, in practice we observed that $\hat{C}$ is quite small.

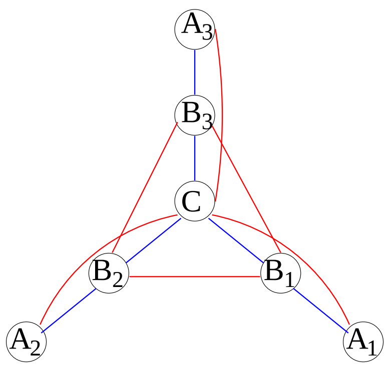

**Figure 1. Instance of a motif graph.**

As an example, consider the motif graph represented in Figure 1, where the consensus $C$ has been planted together with three variants, $B_1, B_2$, and $B_3$; the required quorum is 4, while $\delta = 0$, *i.e.*, $d = d'$. At the beginning, the set $L = \{C, A_1, A_2, A_3, B_1, B_2, B_3\}$ and every node can be selected as seed in the clustering phase. Suppose the algorithm chooses $A_1$, creates the cluster $c_1$ with center $A_1$, and removes $A_1$ from $L$. Then, the algorithm inserts $B_1$ and $C$ in $c_1$, being these two nodes at Hamming distance $\leq 2d$ from $A_1$, and removes $B_1$ from $L$ (Note that $C$ is not removed from $L$ since its distance from $A_1$ is greater than $d$). At the beginning of the second iteration the set $L = \{C, A_2, A_3, B_2, B_3\}$. Again, all the nodes in $L$ are possible candidates to become the seed of the new cluster $c_2$. Now, suppose the algorithm chooses $A_2$, removes it from $L$, inserts the nodes $B_2$ and $C$ in $c_2$, and removes $B_2$ from $L$. In the next iteration of the clustering procedure, the algorithm can choose between $C$, $A_3$ and $B_3$. Suppose it selects $A_3$ as seed of the cluster $c_3$, inserts $B_3$ and $C$ in $c_3$, and removes $B_3$ from $L$. In the next step, the only possible candidate node as seed is $C$. The new cluster $c_4$ has $C$ as seed, and contains all the nodes of $G$. Once removed $C$ from the set $L$, the clustering procedure terminates and all the nodes of the graph fall in at least one cluster. In the second phase, the algorithm discards $c_1$, $c_2$ and $c_3$, being smaller than the quorum, identifies $C$ as the consensus, and removes from $c_3$ the nodes $A_1, A_2, A_3$, being at distance greater than $d$ from $C$.

It is worth nothing that, for any other choice of the cluster centers, the consensus motif would have been also found.

## 4.2 Improve clustering speed

The most time consuming part of the presented algorithm is the clustering procedure, and in particular the comparisons to determine if a node is part or not of a new created cluster. In fact, once a signal (corresponding to a vertex $v_i$ of the motif graph) is selected to be the seed of a new cluster, it has to be compared with all the other signals in $G$, so to find out the adjacent nodes. According to different graph topologies, this procedure could be quadratic in the worst case. In order to drastically reduce the number of distance computations, we reformulate the problem as a "range query search" one, *i.e.*, given a query string $s$ and a set of signals, return all the signals at distance at most $2d$ from $s$.

We create a hash table where each node $v_x \in G$, labeled with the nucleotide sequence $x$, has hash key $h(v_x) = \{|A \in x|, |C \in x|, |G \in x|, |T \in x|\}$. All the nodes with the same hash key fall in the same bucket. We observe that:

$$\frac{1}{2} L_1(h(v_i), h(v_j)) \leq H(v_i, v_j) \tag{1}$$

where $L_1$ is the 1-norm distance of vectors. Given the above lower bound, the search procedure is trivial. The hash key $k$ of the query signal is compared, using the $L_1$ norm, with all the keys of the hash table and all the elements in those buckets for which the distance with $k$ is greater than $4d$ can be safely ignored, thus reducing the number of distance computations. The effectiveness of this optimization depends on the value of distance $d$ and the length $l$ of the signals. In fact, both the Hamming distance and the $L_1$ norm are bounded in the range $[0, l]$. Thus, if $d \geq l/4$, this procedure does not save distance computations. On the other hand, the larger is the value of $d$ the smaller is the number of clusters (and searches) to be done.

## 5 Experiments

In order to evaluate the proposed algorithm, we run three different sets of experiments, trying to stress the *(l-d) Challenge Problem* from different points of view. Note that our method is always able to find the implanted solution, therefore the only parameter of performance to consider is the running time.

The results reported in Tables 1–3 are in terms of the elapsed time for finding the solution, highlighting the time needed for the clustering phase. The clustering and total time have been obtained on an 3.2 GHz Intel Pentium D dual-core workstation with 3.2 GB RAM running Linux kernel 2.6.18.2-34. The code has been implemented in Python 2.4 We also report the $E(l, d)$, *i.e.*, the expected number of spurious solutions of that class in the background sequence.

In the experiments performed, each of the $t$ sequences of length $n$ has been generated independently from the others using a uniform random distribution. The $l$-length motif consensus $M$ (corresponding to $\delta = 0$) and $q - 1$ mutations of $M$, generated by mutating $M$ in exactly $d$ random positions, are implanted in $q$ randomly selected sequences.

The first set of experiments (Table 1) aims at assessing the performance of the algorithm for "critical" assignments of the parameters $l$ and $d$. For fixed values of $n$, $t$ and $q$, we say that the assignment of $l$ and $d$ is critical when, given a certain value of $l$, the parameter $d$ is the largest value for which $E(l, d) < 10$. Critical pairs are important since for larger number of allowed errors, the number of spurious solutions is such that the search time is dominated by the time for identifying and reporting all the spurious solutions.

In particular, we fixed $t = 20, n = 600, q = 20$, and run our algorithm with the *(l-d)* values proposed by Davila et al. in [6], *i.e.*, (13-4), (13-5), (17-5), (17-6), (19-6), (19-7), (21-7), and (21-8). Furthermore, we tested our algorithm with more challenging values of *(l-d)*, considering quite longer motifs and beyond the classical threshold of a number of mutation $d < 1/3l$: (15-5), (23-9) and (25-10). The results show that our method has a very slow increase, since it weakly depends on the values of $l$ and $d$. Even harder cases, such that (25,10), (23,9) and (21,8), are solved in just about twenty minutes.

The second set of experiments (Table 2) analyzes the algorithm behavior when the quorum threshold is lowered. In particular, similarly to the experiments proposed in [11], we fixed $l = 15$, $d = 4$, $t = 20$, $n = 600$, and reduced the quorum $q$ from 20 to 13. Actually, we further stress the quorum, lowering it till 11. The results reported in Table 2 show that our algorithm is able to find the solution without any time penalty even with only 11 embedded instances.

The third set of experiment (Table 3) explores the influence of the parameter $n$, the length of the $t$ input sequences. In particular, we fixed $l = 15$, $d = 4$, $t = 20$, $q = 20$, and increased the sequence length $n$ from 600 to 4000 in steps of 200. The algorithm is a bit slower than some existing methods, since it solves the case $n = 2000$ within 1 hour and 31 minutes, but it exhibits a quite slow rate of increase. It is worth noting that it is able to tackle strings up to $n = 4000$ in a very reasonable time (5 hours and 40 minutes), for such a large instance.

Overall, the results reported here show that the proposed algorithm is robust and efficient when the main parameters, $l, d, q, n,$ are scaled up, even to very challenging values.

| *(l-d)* | Clustering | Total Time | E(l,d) |
|---|---|---|---|
| (13-4) | 218.565s | 10m 54.993s | 5.2 |
| (13-5) | 66.360s | 23m 24.288s | - |
| (15-5) | 246.836s | 13m 8.953s | 2.84 |
| (17-5) | 582.865s | 12m 37.531s | 2 x $10^{-15}$ |
| (17-6) | 271.120s | 15m 33.498s | 0.88 |
| (19-6) | 666.16s | 15m 6.781s | 9 x $10^{-16}$ |
| (19-7) | 356.264s | 17m 59.123s | 0.17 |
| (21-7) | 704.394s | 16m 59.336s | 2.5 x $10^{-16}$ |
| (21-8) | 377.834s | 19m 28.869s | 0.02 |
| (23-9) | 450.470s | 21m 41.841s | 0.002 |
| (25-10) | 532.093s | 22m 47.773s | 0.0002 |

**Table 1. Results in terms of elapsed seconds, for t=20, n=600, q=20, varying (l-d). E(l,d) is the expected number of spurious solutions of that class in the background sequence.**

| q | Clustering | Total Time | E(l,d,q) |
|---|---|---|---|
| 20 | 513.224s | 10m 23.691s | 2.17 x $10^{-15}$ |
| 19 | 496.896s | 10m 10.422s | 6.65 x $10^{-13}$ |
| 18 | 515.843s | 10m 22.107s | 9.65 x $10^{-11}$ |
| 17 | 513.126s | 10m 20.099s | 8.86 x $10^{-9}$ |
| 16 | 541.970s | 10m 45.284s | 5.76 x $10^{-7}$ |
| 15 | 515.600s | 10m 24.787s | 2.82 x $10^{-5}$ |
| 14 | 528.670s | 10m 33.624s | 1.07 x $10^{-3}$ |
| 13 | 494.069s | 10m 7.126s | 0.033 |
| 12 | 533.175s | 10m 39.192s | 0.82 |
| 11 | 534.171s | 10m 40.492s | 16.7 |

**Table 2. Results in terms of elapsed seconds, for t=20, n=600, l=15, d=4, varying q. E(l,d,q) is the expected number of spurious solutions of that class in the background sequence.**

| n | Clustering | Total Time | n | Clustering | Total Time |
|---|---|---|---|---|---|
| **600** | 538.481s | 11m 37.985s | **2400** | 4141.475s | 129m 24.437s |
| **800** | 788.146s | 18m 1.316s | **2600** | 4744.862s | 151m 47.749s |
| **1000** | 1128.770s | 26m 39.276s | **2800** | 5339.982s | 165m 11.531s |
| **1200** | 1439.996s | 35m 41.334s | **3000** | 6296.115s | 203m 23.263s |
| **1400** | 1846.050s | 48m 15.665s | **3200** | 6992.294 | 228m 10.964s |
| **1600** | 2246.456s | 60m 21.614s | **3400** | 7785.070 | 255m 7.502s |
| **1800** | 2847.454s | 77m 9.381s | **3600** | 8330.551 | 277m 48.584s |
| **2000** | 3200.224s | 91m 51.200s | **3800** | 9317.619 | 314m 53.355s |
| **2200** | 3778.337s | 110m 23.630s | **4000** | 9909.280 | 342m 43.407s |

**Table 3. Results in terms of elapsed seconds, for t=20, l=15, d=4, q=20, varying n.**

## 6 Conclusions

The planted *(l-d) Challenge Problem* is a purely combinatorial problem that, while abstracting from many details of the motif finding problem for biological sequences, captures the core combinatorial hardness of motif finding. What constitutes a challenging size of the problem has evolved with time. However most algorithms are able to scale well only when one of the relevant parameters of the problem is varied. We have shown a promising new algorithm that performs well when all of the relevant parameters of the problems are varied. Application and testing of the proposed algorithm to biological data is work in progress.

## Funding

Funding for this research has been partially supported by Italian Registry of ".it" ccTLD and by the EU funded 7FP Virtual Physiological Human Network of Excellence (VPH NoE) (contract number 223920).

## References

[1] Malin C Andersen, Pr G Engstrm, Stuart Lithwick, David Arenillas, Per Eriksson, Boris Lenhard, Wyeth W Wasserman, and Jacob Odeberg. In silico detection of sequence variations modifying transcriptional regulation. *PLoS Comput Biol*, 4(1):e5, 01 2008.

[2] Sudha Balla, Jaime Davila, and Sanguthevar Rajasekaran. On the challenging instances of the planted motif problem. Technical Report BECAT/CSE-TR-07-2, Booth Engineering Center for Advanced Technology (BECAT.), 2007.

[3] Christina Boucher, Daniel G. Brown, and Paul Church. A graph clustering approach to weak motif recognition. In Raffaele Giancarlo and Sridhar Hannenhalli, editors, *WABI*, volume 4645 of *Lecture Notes in Computer Science*, pages 149–160. Springer, 2007.

[4] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. In *Journal of Computational Biology*, pages 69–76, 2001.

[5] Modan Das and Ho K. Dai. A survey of dna motif finding algorithms. *BMC Bioinformatics*, 8(Suppl 7), 2007.

[6] Jaime Davila, Sudha Balla, and Sanguthevar Rajasekaran. Fast and practical algorithms for planted (l, d) motif search. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 4(4):544–552, 2007.

[7] M. Haussler and J. Nicolas. Motif discovery on promoter sequences. Technical Report 5714, INRIA, October 2005.

[8] Dustin Holloway, Mark Kon, and Charles DeLisi. In silico regulatory analysis for exploring human disease progression. *Biology Direct*, 3(1):24, 2008.

[9] U Keich and PA Pevzner. Finding motifs in the twilight zone. *Bioinformatics*, 18(10):1374–81, 2002.

[10] Kjetil Klepper, Geir Sandve, Osman Abul, Jostein Johansen, and Finn Drablos. Assessment of composite motif discovery methods. *BMC Bioinformatics*, 9(1), 2008.

[11] Shoudan Liang. cwinnower algorithm for finding fuzzy dna motifs. In *CSB '03: Proceedings of the IEEE Computer Society Conference on Bioinformatics*, page 260, Washington, DC, USA, 2003. IEEE Computer Society.

[12] Laurent Marsan and Marie-France Sagot. Extracting structured motifs using a suffix tree—algorithms and application to promoter consensus identification. In *RECOMB '00: Proceedings of the fourth annual international conference on Computational molecular biology*, pages 210–219, New York, NY, USA, 2000. ACM.

[13] Giulio Pavesi, Giancarlo Mauri, and Graziano Pesole. In silico representation and discovery of transcription factor binding sites. *Briefings in Bioinformatics*, 5(3):217–236, 2004.

[14] PA Pevzner and SH Sze. Combinatorial approaches to finding subtle signals in dna sequences. *Proc Int Conf Intell Syst Mol Biol*, 8:269–78, 2000.

[15] S. Rajasekaran, S. Balla, and C.-H. Huang. Exact algorithms for planted motif problems. *Journal of Computational Biology*, 12(8):1117–1128, 2005.

[16] Geir Sandve, Osman Abul, Vegard Walseng, and Finn Drablos. Improved benchmarks for computational motif discovery. *BMC Bioinformatics*, 8(1):193, 2007.

[17] Geir Sandve and Finn Drablos. A survey of motif discovery methods in an integrated framework. *Biology Direct*, 1:11, 2006.

[18] Sing-Hoi Sze, Songjian Lu, and Jianer Chen. Integrating sample-driven and pattern-driven approaches in motif finding. In *Algorithms in Bioinformatics, 4th International Workshop, WABI*, pages 438–449, 2004.

[19] Martin Tompa et Al. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23(1):137–144, January 2005.