11-29-2009

Motif Recognition

Sushain Pandit
Iowa State University

Susan R. VanderPlas
Iowa State University, skoons@iastate.edu

Chaoliang Zhang
Iowa State University

Motif Recognition

Sushain Pandit, Susan VanderPlas, Chaoliang Zhang

November 29, 2009

Abstract

The problem of recognizing motifs from biological data has been well-studied and numerous algorithms, both exact and approximate, have been proposed to address the underlying issue. We strongly believe that open availability and ease of accessibility of quality implementations for such algorithms are critical to the research community, in order to directly reproduce and utilize the results from other studies, so as not to reinvent the wheel. Moreover, it is also important for the implementation to be as generic as possible so that any researcher can to extend it with minimal effort to test a newly implemented algorithmic extension or heuristic. With this motivation, we choose to focus an existing algorithm, `PatternBranching` and, to a lesser degree, `Yang2004`. We analyze these approaches for minor heuristical changes & speed-ups by adjusting certain thresholds, and finally, implement the variants in high-level languages (Java and C) using thought through programming practices and generic, extensible interfaces. We also analyze the performance of `PatternBranching` using a synthetically generated test-suite for a variety of sequence lengths and report the results. Code from this project will be made freely available online to the research community.

1 Introduction

The Motif Recognition problem is critically important to biologists, and, consequently, to bioinformaticists. Many biological molecules bind only to certain short sequences found in DNA. Thus, if biologists know that certain genes are all activated by the same molecule, then it is reasonable to suspect that they share similar motifs. With the biological problem stated, it is possible to formulate the computational problem.

In its original formulation, the challenge problem for identifying weak motifs was stated:

> "Find a signal in a sample of sequences, each 600 nucleotides long and each containing an unknown signal (pattern) of length 15 with [up to] 4 mismatches."[2]

In 2000, when the problem was first formally posed, there was no algorithm that could reliably solve the challenge problem [4]. Algorithms existed that theoretically might have worked, but for one problem that they could not seem to overcome: Any two motifs l and k could have up to 8 differences between them. As l and k are 15 nucleotides long, a difference

of 8 between them is significant. As a result, typical motif finding algorithms do not perform well on the challenge problem. The brute force approach does not work particularly well either: 4^l, with $l > 10$ is too computationally intensive, and as there is no guarantee that the motif actually appears in the sample sequences (given the rate of biological errors), it is difficult to reduce the problem to a reasonable number of computations.

2 Definitions and Notation

2.1 Challenge Problem

Input: n sequences of length m, length of the motif l, and maximum hamming distance d from the uncorrupted signal
Output: All locations of the signal
The challenge problem is also known as the *planted (l,d) motif problem.* Pevzner & Sze (2000) originally posed the planted (15, 4) problem, where each string provided contains a motif of length 15, with at most 4 errors.

2.2 Notation

The following notation will be used throughout the paper.

m : length of the input sequences
S : set of sequences of length m each
n : number of input sequences
N : the total amount of text input,$(N = m * n)$
l : length of the motif to be found
C : set of all l-mers in the input sequences
M : motif, a pattern of length l
d: maximum hamming dist. between a mutated motif and the target motif M

Many approaches to the challenge problem use graph-based algorithms. Let $G(E, V)$ be a graph of C, such that u and v are connected if $dist(u, v) \leq 2d$ and u and v are from different input sequences. G is multipartite and "almost random."

A clique, $Q = v_1, ..., v_n$, is a subset of $G(E, V)$ such that for each $v_i, v_j \in Q$, there is an edge between v_i and v_j. Q is then a fully connected subgraph. Furthermore, Q is extendable if Q has a neighbor in each part of G.

For pattern-based approaches, especially Pattern-Branching, we utilize the following notations in addition to those above:

$D_{=k}(A_j)$: set of patterns at a hamming distance of exactly k from A_j.
$\mathbf{d}(A, S_i) : min\{\mathbf{d}(A, P) \mid P \in S_i\}$, where P is an pattern of lenth l
$\mathbf{d}(A, S)$: total distance of a pattern A from the sample (S).
Note: $\mathbf{d}(A, S) = \sum_{S_i \in S} \mathbf{d}(A, S_i)$
$BestNeighbors(A, t)$: t patterns from $\{D_{=1}(A)\}$ with the lowest total distance

3 Related Work

Due to its importance and vast impact, the problem has been very well-studied and consequently, many algorithms and approaches, both exact and approximate, have been proposed. One of the primary criteria for categorization for such algorithms is whether they are pattern or profile based. *Pattern-based* algorithms predict the actual motif by searching in the pattern space (optionally followed by determining it's position of occurrence in each of the given sequences), while *profile-based* algorithms emphasize predicting the starting positions of occurrence of the motif by searching the profile space.

Each of the above algorithm classes may be exact or approximate. Exactness often comes with the price of exhaustive enumeration. This leads to a combinatorial explosion that is often associated with such problems due to the large number of expected (possibly mutated) motifs. We emphasize this in the analysis below and follow it up by describing some of the popular algorithms that have been proposed in literature for solving the problem.

3.1 Initial Probability Analysis

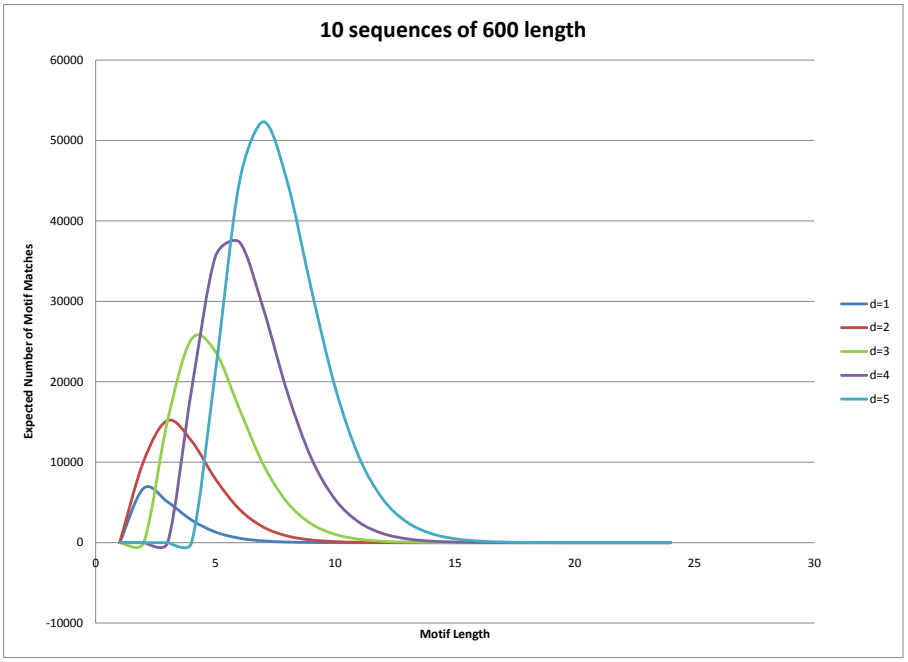


Figure 1: Expected Number of Motif Matches

There are 4^l possible target sequences, and given one target sequence, there are $\binom{l}{d}3^d$ sequences within Hamming distance d of the target (Figure 1). In the given n sequences,

there are $n(m-l+1)$ positions for possible mutated motifs, and the probability that a mutated motif occurs at any given position is $\binom{l}{d}\left(\frac{3}{4}\right)^d\left(\frac{1}{4}\right)^{l-d}$. Then, the expected number of mutated motifs in a sequence is $\left(\sum_{i=0}^{l-1}\lfloor\frac{m+i}{l}\rfloor\right)\binom{l}{d}\left(\frac{3}{4}\right)^d\left(\frac{1}{4}\right)^{l-d}$, which would correspond to the expected number of "spurious hits" of a random "target" on a given sequence (Figure 2).

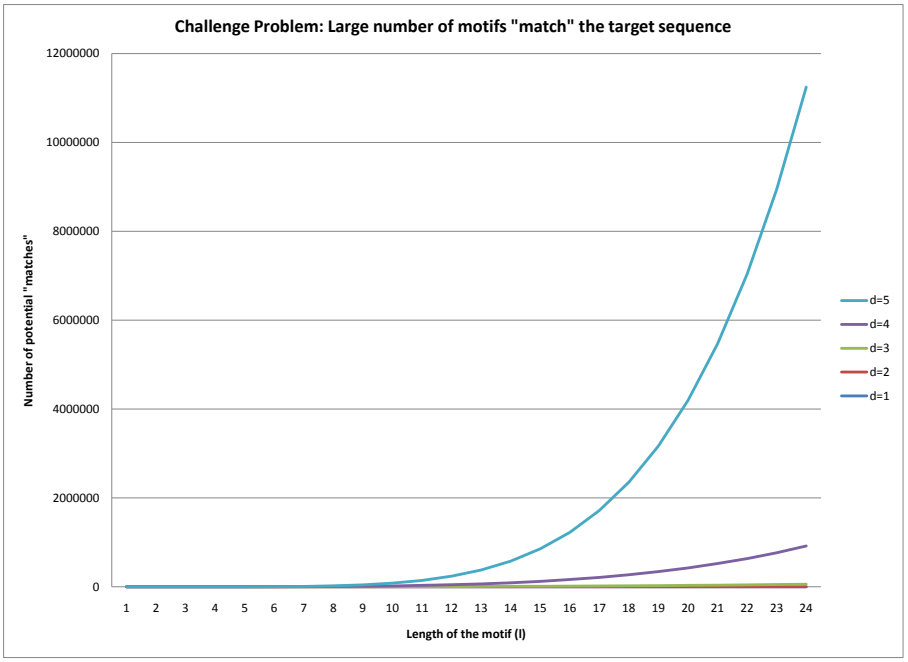


Figure 2: Number of potential matches for a target sequence of length l

3.2 Probability Algorithms

Using a library of all l-mers in the input sequences (C in our notation), a reasonable approach would be to calculate a "significance value" of each l-mer in C, based on the number of repetitions of that l-mer. l-mers with high significance values would then be potential motifs. This approach fails when there are mutated motifs, because the target sequence may not appear at all, and if it did appear, there might be scattered mutations, dividing the significance value of each mutated motif and making it difficult to recognize. Enumeration methods used on other motif problems do not generally work on the challenge problem because of the mutations in the motifs and the length of the target sequence.

3.3 Projection Algorithm

In a manner similar to the probabilistic method proposed above, `Projection` chooses k of the l positions at random, and then uses those k positions to hash each l-mer in the input sequences. The hash function $h(x)$, where x is the l-mer input to the hash function, results from concatenating the k residues in x. x is a l-dimensional Hamming space, and $h(x)$ is the projection of x onto a k-dimensional space.[1] k is chosen such that $k < l - d$, but k should not be too large because sequences must match the planted motif at all k positions. Once hashing is complete, any bucket in $h(x)$ with more than s l-mers is refined further to obtain possible motifs. Refinement occurs via the EM algorithm in [1] in a manner similar to the refinement in `SP-STAR`. `Projection` performs better on average than `WINNOWER`, `SP-STAR`, and a traditional motif finding algorithm, `Gibbs`, on variations of the planted (l, d) motif problem with $10 \leq l \leq 19$ and $2 \leq d \leq 6$[1].

3.4 WINNOWER and SP-STAR Algorithms

Pevzner & Sze created the `WINNOWER` algorithm to solve the weak motif problem. The `WINNOWER` algorithm uses a graph approach, with nodes of subsequences and edges connecting similar subsequences. `WINNOWER` then has to determine which edges are spurious relations, and which edges are actual instances of the motif. According to Pevzner & Sze, there are 20,000 spurious edges for every signal edge. `WINNOWER` uses clique-finding techniques to delete spurious edges while preserving signal edges, and generally reduces the graph so that the motif is computable. It does so by recognizing that every edge in a maximal n-clique belongs to $\binom{n-2}{k-2}$ extendable cliques of size k. `WINNOWER` can detect multiple motifs[2]. `WINNOWER` has a running time of $O(N^{2d+1})$, but typically runs much faster.

Another algorithm created by Pevzner & Sze, `SP-STAR`, improves `WINNOWER`'s computational demands by making an observation: `WINNOWER` does not account for the edit distance between two nodes in the graph. Two strings who have a hamming distance of 1 should be weighted more than two strings with the maximum hamming distance of $2d$. `SP-STAR` minimizes the hamming distance between a median string and motifs in the sequences. From that initial attempt, `SP-STAR` refines the initial signal by computing the *majority string*- the string that has the most common character in each position. Modifications of `SP-STAR` allow for the use of refining techniques based on different algorithms[2].

3.5 Extension of WINNOWER and SP-STAR

Yang & Rajapakse (2004) proposed another algorithm for clique-based weak motif detection. They allow for n sequences x_i of length m_i, where m_i is allowed to vary. In this paper, the problem was restricted so that each x_i contained only one instance of the mutated motif. Then, a reference sequence $x_r \in x$ is connected to each other sequence such that the motifs are linked in the graph if $d_H < 2d$. Once the graph has been created, dynamic programming techniques are used to search each graph for cliques of size n. This algorithm was partially implemented as part of this project, and is detailed below [5].

3.6 Pattern Based Algorithms

Pattern branching algorithms use a series of strings A_i of length l to determine a motif signal. Each A_i is considered a "neighbor" of any other A_j if it is within hamming distance d. For each $l - mer$ in the input, the neighbors are scored, and the highest scoring "neighbor" is then the motif signal. The `PatternBranching` algorithm presented in Price et al. (2003) applies a `BestNeighbor` function sequentially to each occurence A_i of the motif M within distance $k < d$, finally scoring A_n against M, reducing the number of potential motifs to check. The pseudocode for the algorithm, as given in [3]:

S, the set of all sequences of length l in the sample
l, the length of the sequence
k, the number of mutations between two sequences

`PatternBranching`(S, l, k)
    $M \leftarrow$ *an arbitrary sequence of length* l
    for each l-mer A_0 in S
        for $j \leftarrow 0$ to k
            if(d(A_j, S) < d(M, S)
                $M \leftarrow A_j$
            $A_{j+1} \leftarrow$ BestNeighbor(A_j)
    return M

`PatternBranching` has been shown [3] to perform well on the problem. Improvements can be made by storing more than one `BestNeighbor`.

Our implementation of `PatternBranching` is detailed in Section 5.1.

4 Project Scope and Motivation

With numerous existing approaches and theroetically-proven efficiency bounds, there are many different approaches one could take to this problem. Trying to come up with an entirely novel algorithm or marginally improving the existing efficieny bounds is certainly one approach, however, we feel that open availability and ease of accessibility of quality implementations for such algorithms is equally critical to research community, because it provides easy modification and implementation, as well as consistency between papers. Moreover, it's also important to have the said implementations as generic as possible so that any researcher is able to extend it at will in order to test a newly implemented algorithmic extension or heuristic.

With this motivation and to the effect of enabling accessibility of quality implementation, we chose to focus on two of the existing algorithms `PatternBranching` and `Yang2004`. We will analyze these approaches for potential (minor) heuristical changes & speed-up techniques. Finally, we implement these variants in high-level languages (C and Java) using

thoughtful programming practices and generic, extensible interfaces. We also analyze the performance of `PatternBranching` using a synthetically generated test-suite for a variety of sequence lengths. Code for this project is now available at http://sushain.com/cs567/project/, and will later be uploaded to a more well-known online repository such as SourceForge.

5 Implementation Details

5.1 System Design for Pattern Branching Variant

We retain the core idea of `PatternBranching` as explained in section 2.6 of [3]. However, instead of having a single "BestNeighbor", we make $BestNeighbors(.,.)$ return top t best neighbors and we use this set (stored in A_j, $j > 0$) to perform a more thorough search of the neighbor space. The underlying steps are as described below.

(1) Starting with an arbitrary motif pattern, loop over the entire sequence space, S and peform the following steps.

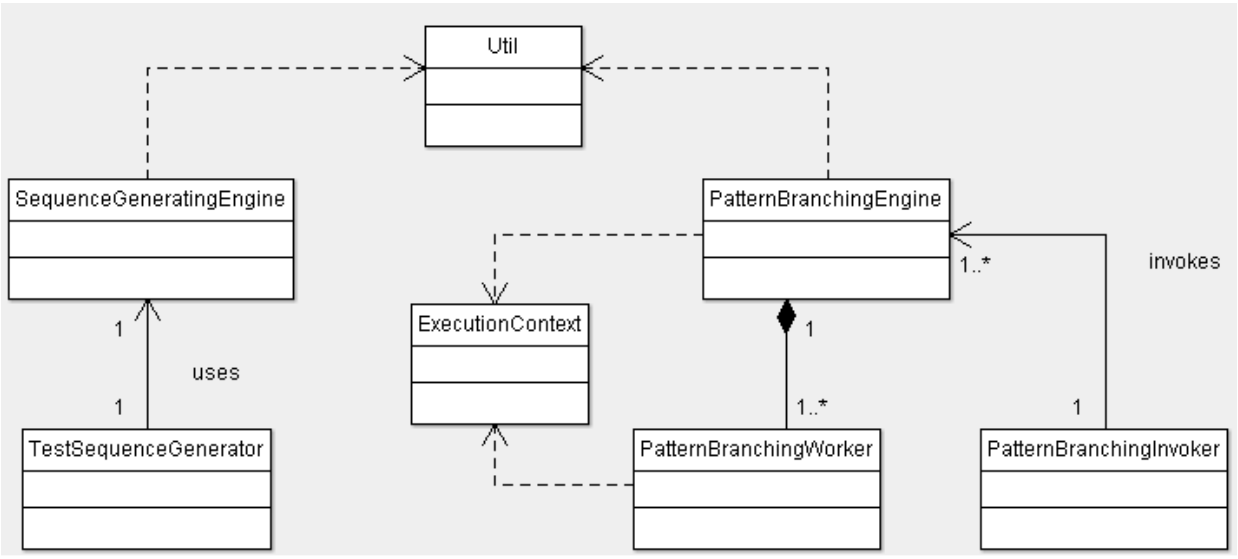
   (a) For each j from 0 to d, perform step 3.

   (b) Check if $d(a_k, S) < d(M, S), \forall a_k \in A_j$. Set motif, M, to the pattern corresponding to the minimum of these distances.

   (c) Set A_{j+1} to the output of $BestNeighbors(A_j, t)$

(2) Output M as the recognized motif pattern.

The above algorithm is implemented in Java and the resulting system design is (briefly) shown below.



`SequenceGeneratingEngine` is utilized to generate synthetic sequences, which are in turn consumed by the `PatternBranchingEngine` to find the motif pattern. It, in turn, delegates the work of finding the BestNeighbors and distances to it's worker. The motif discovery process is initiated by `PatternBranchingInvoker`, which launches the engine with the appropriate parameters.

5.2 Implementation of Yang & Rajapakse (2004)

Implementation of Yang & Rajapakse (2004)- Method (as printed in [5]:
Let the dataset of DNA sequences be denoted by $x = x_i : i = 1, ..., m$, where m is the number of sequences and each sequence $x_i = \{\omega_{i1}, \omega_{i2}, ...\omega_{in_i}\}$, where $\omega_{ij} \in \Omega_{DNA}$, n_i is the length of the sequence x_i and Ω_{DNA} is the alphabet containing the four nucleotides [5].

5.2.1 Step 1: Graph Construction

Randomly select a sequence x_r $(1 \leq r \leq m)$ from the dataset and refer it as a *reference sequence*. As each sequence in the text contains a motif instance, let that instance be at v_{rp}, where p is the starting position. For each position $p = 1, ..., n_r - l + 1$, build a graph G_p as follows:

(a) For any other sequence in the dataset 1, $x_i \in x$, find a subsequence represented by the vertices v_{ij} where $i = 1, 2, ..., m = r$ and $j = 1, 2, ..., n_i - l + 1$

(b) Connect two vertices v_{rp} and v_{ij} with edge $e_{v_{rp},v_{ij}}$ if $d_H(v_{rp}, v_{ij}) \leq 2d$ [5]

5.2.2 Step 2: Clique Finding

This is a dynamic programming (DP) approach to search cliques of vertices representing motif instances.

(a) Let the set $S = s_r, s_2, ..., s_m$ represent the subsets of vertices in the graph G_p such that
$s_r = \{v_{rp}\}$, $s_2 = \{v_{21'}, v_{22'}, ..., v_{2c_2}\}$,...,
$s_i = \{v_{i1'}, v_{i2'}, ..., v_{ic_i}\}$,..., $s_m = \{v_{m1'}, v_{m2'}, ..., v_{mc_m}\}$.

(b) Lists $L_{ij'}$, $j = 1, ..., c_i$, are created as follows:[5]

(i) Set $L_{rp} = \{\}$

(ii) $L_{2j'} = \{v_{rp}\}$ for vertex $v_{2j'} \in s_2$.

(iii) for $i = 3, ..., m$

Set $L_{ij'} = \{\}$ for $v_{ij'} \in s_i$
for each vertex $v_{i-1,k'} \in s_{i-1}$, $k = 1, ..., c_{i-1}$
if $d_H(v_{ij'}, v_{i-1,k'}) \leq 2d$
$L_{ij'} = L_{ij'} \bigcup v_{i-1,k'} \bigcup L_{i-1,k'}$

Repeat
Repeat

(c) By maintaining the lists for each vertex $v_{ij'} \in s_i$, if a clique of size m is in G_p, there must be a list $L_{mj'}$ for vertices $v_{mj'} \in s_m$ that contains vertices from each set s_i. The list is then processed for each $v_{ij'}$ to make sure that all vertices in $L_{ij'}$ are within $2d$ of $v_{ij'}$.[5]

5.2.3 Step 3: Rescanning

Remove the influence of spurious motif instances by rescanning the dataset with the motif consensus sequences derived from the cliques and saving those instances satisfying the inequality $d_H(\psi, \psi_i) \leq d$.

6 Evaluation and Results

For purposes of evaluation, we chose to generate synthetic sequence data since we wanted to have complete comtrol over the characteristics and variations in the test samples. To this effect, we generated three different test-sets for 20 sequences ($n = 20$) of length 800, 1000 and 2000 (m). For each of these cases, we generate a random motif of varying length ($l = 15, 20, 25, 30, 35$). Further, we randomly insert this motif into the generated sequence and then mutate it in exactly (d) places into any of the three remaining nucleotides. These motifs follow the Fixed-number of Mutations (FM) model. There is another variant, called Variable-number of Mutations (VM), in which each nucleotide of the pattern is mutated into one of the remaining three nucleotides with a fixed probability, however, we don't employ it in our evaluation.

m	l	d	Implanted Motif(M)	Discovered Motif	Time (s)
800	15	4	ACAACCCCTTAAGCT	ACAACCCCTTAAGCT	27.157
800	20	6	**T CAACATATAATAGCGTATG**	**CAACATATAATAGCGTATG C**	28.125
800	25	7	AGTTTAGTCACATGGGGAACAGTAC	AGTTTAGTCACATGGGGAACAGTAC	27.750
800	30	9	CTAGAGCATTGTCCTGACCAACTCTGGAGT	CTAGAGCATTGTCCTGACCAACTCTGGAGT	41.734
800	35	10	GCCCTAACTTGCATGCAACAATCTCCCCTAGTGGT	GCCCTAACTTGCATGCAACAATCTCCCCTAGTGGT	27.375
1000	15	4	AGTAAGCAGCGCCGT	AGTAAGCAGCGCCGT	41.641
1000	20	6	**ATGCCGCTAGCGACCGGAC C**	**T ATGCCGCTAGCGACCGGAC**	43.469
1000	25	7	ACCTTCGGCTAAGTCGAGCCCGCCC	ACCTTCGGCTAAGTCGAGCCCGCCC	42.813
1000	30	9	CTTACTCTATATCCTGACTCCAGCAGTAGC	CTTACTCTATATCCTGACTCCAGCAGTAGC	64.594
1000	35	10	GCCGCGCCAACAACGGGACCGTCTGCTAACTCATG	GCCGCGCCAACAACGGGACCGTCTGCTAACTCATG	42.015
2000	15	4	CTTCTTTCTATTAAG	CTTCTTTCTATTAAG	164.563
2000	20	6	**A GTGGTTAGCTGGCGATTGT**	**GTGGATAGCTGGCGATTGT A**	172.156
2000	25	7	AATGTCTTTAGTTACTAACTTGGAC	AATGTCTTTAGTTACTAACTTGGAC	169.406
2000	30	9	GGTCTTGAATACCCGTGCGTATGAACTTTG	GGTCTTGAATACCCGTGCGTATGAACTTTG	250.078
2000	35	10	GAGAAAGGGATTAACTGGGTGGTTATGGATGCGCG	GAGAAAGGGATTAACTGGGTGGTTATGGATGCGCG	163.500

Table 1: Algorithm - Pattern Branching Variant [$fixed\, n = 20$]

In our analysis, we report the discovered motifs for each test instance and overall success-rate of `PatternBranchingVariant` We also include the very-hard problem instances for the case of subtle motifs $(15, 4)$ with $m = 2000$. On these very-hard problem instance, the performance of other algorithms (like `Projection`) has been reported around 80% in around

2 minutes (on an average) [1]. We see below that `PatternBranchingVariant` performs close to this class of algorithms in terms of the success-rate as well as the time taken in finding the correct motif (as per our test simulation).

Table 1 shows the performance of `PatternBranchingVariant` on different test sequence sets. We've also reported the running time to indicate the efficiency in terms of time taken to arrive at the desired motif.

7 Discussion and Conclusion

We described two algorithm implementations and corresponding system design for the motif recognition problem. The first implementation is based on a variant of original `PatternBranching` algorithm, with a minor heuristic modification for defining the *Best-Neighbors* as a set instead of a single pattern. Evaluations run on our algorithm implementation show that restricting pattern search to small neighborhoods generally leads to increased efficiency in terms of the time it takes to determine the correct motif, while not sacrificing accuracy. As suggested in the original paper on `PatternBranching`, it would be interested to see how this algorithm implementation scales to the problem instances where input sequences consist of more than 4 nucleotide types (A, C, G, T). Future explorations could test this idea by generating synthetic sequences using a bigger alphabet of nucleotides. Since this is mainly a system implementation, it would be interesting to explore the extensibility of this framework to perform small heuristic changes by simply overriding the interface specifications and quickly testing the results.

References

[1] J. Buhler and M. Tompa. Finding motifs using random projections. *Journal of Computational Biology*, 9(2):225–242, 2002.

[2] P. A. Pevzner and S. H. Sze. Combinatorial approaches to finding subtle signals in DNA sequences. *Proc Int Conf Intell Syst Mol Biol*, 8:268–278, 2000.

[3] A. Price, S. Ramabhadran, and P. A. Pevzner. Finding subtle motifs by branching from sample strings. *Bioinformatics (Oxford, England)*, 19 Suppl 2, October 2003.

[4] S. Rajasekaran. *Handbook of Computational Molecular Biology*, chapter 37. Chapman and Hall/CRC, 2006.

[5] X. Yang and J. C. Rajapakse. Graphical approach to weak motif recognition. *Genome Informatics*, 15(2), 2004.

8 Division of Responsibility & Labor

(1) Abstract: Sushain Pandit, edited by Susan VanderPlas

(2) Introduction: Susan VanderPlas, edited by Sushain Pandit

(3) Definitions & Notation: Susan VanderPlas, edited by Sushain Pandit

(4) Related work: Susan VanderPlas, with intro paragraph by Sushain Pandit

(5) Motivation: Sushain Pandit, edited by Susan VanderPlas

(6) Algorithm I (PatternBranching): Sushain Pandit, coding by Sushain Pandit, edited by Susan VanderPlas

(7) Algorithm II (Yang 2004): Chaoliang Zhang, edited and expanded by Susan VanderPlas

(8) Evaluation & Results: Sushain Pandit, coding by Sushain Pandit, edited by Susan VanderPlas

(9) Discussion & Conclusion: Sushain Pandit, edited by Susan VanderPlas

Latex, Formatting, Editing, etc.: Susan VanderPlas