1 Introduction

In this class, we have seen many algorithms for regulatory motif finding. In general, these algorithms fall into two categories – probabilistic and combinatorial. Probabilistic solutions, such as expectation-maximization and Gibbs sampling have had much empirical success, but they are not well understood from a theoretical standpoint. For combinatorial solutions, either the algorithms use exhaustive search and find the “best” motif, or they have more practical running times, but, again, cannot be proven to find the “best” motif.

I began this project as a critical review of the following paper:

Toward Optimal Motif Enumeration
Patricia A. Evans and Andrew D. Smith

Upon a careful reading of this paper, I found the results to not be so compelling. The paper gives exponential-time algorithms for enumerating all $l$-mers within Hamming distance $d$ of some substring of every given string $S_1, \ldots, S_m$. The running time of their algorithm improves upon prior results. However, their algorithm is designed to minimize the worst-case running time, and is not dependent on the structure of the input; when a motif is “obvious”, the running time does not improve.

Instead of reviewing this paper, I investigated the regulatory motif finding problem from the combinatorial perspective a bit more. My results are:

1. NP-hardness of the Motif-finding problem. This had been known before, but I was unable to find a proof of it, so I proved it myself.

2. Trying all $l$-mers of the input strings $S_1, S_2, \ldots, S_m$ yields a 2-approximation to the motif-finding problem. In particular, there is always one $l$-mer among the input strings $S_1, \ldots, S_m$ that is at Hamming distance at most $2d$ from some substring in $S_1, \ldots, S_m$, where $d$ is the Hamming distance associated with the optimal motif. While this algorithm was mentioned in class, the fact that it is a 2-approximation was not mentioned. I also suggest heuristics that might improve this algorithm.

3. An implementation of the above algorithm. I tried it on the four datasets given for Problem set 3. In three of the four datasets, it found the few, true motifs. On the fourth dataset (data4), the few, true motifs were not found – instead, many, spurious motifs were found.

4. An “average-case” analysis of the 2-approximation algorithm. The empirical results suggest that, in some circumstances, the algorithm does find a few, good motifs. I attempted to
answer the question: when can we apply the algorithm to find few, good motifs? To this end, I considered a planted-motif mode. In this model, an \( l \)-mer with hamming distance less than \( d \) to a fixed motif is placed in \( m \) random strings (i.e. each character is independently and uniformly one of A,G,T,C). I show that when the distance \( d \) is not too large (in terms of \( l \), \( m \), and \( n \)), the 2-approximation algorithm will find the true, planted motif.

\section{NP-hardness of Motif-finding}

Here, we prove the following Motif-finding problem is NP-hard.

**Given:** Strings \( S_1, \ldots, S_m \) of length at most \( n \) over an alphabet \( \Sigma \), integers, \( l \leq d \leq n \), and a scoring function \( f : \Sigma \to \mathbb{Z} \).

**Output:** Does there exist an \( l \)-mer \( w \) such that \( d_f(w, x_i) \leq d \), for \( i = 1, \ldots, m \), where \( x_i \) is some \( l \)-mer appearing in \( S_i \).

The function \( d_f \) is defined as follows: \( d_f(x, y) = \sum_i f(x_i, y_i) \).

**Proof.** We give a reduction from 3-SAT to this problem, i.e. we show how, given a 3-SAT formula \( F \) with \( m \) clauses and \( n \) variables, we can construct an instance of the motif-finding problem in polynomial time.

Let \( F = C_1 \land C_2 \land \ldots C_m \). We construct \( m \) strings \( S_1, \ldots, S_m \), each of length \( 7n + 6 \) such that:

\[ F \text{ is satisfiable } \iff \exists \text{ } n\text{-mer } w \text{ of with } d_f(w, x_i) \leq n - 3 \text{ for some } x_i \text{ in each string } S_i. \tag{1} \]

We map each \( C_i \) to a string \( S_i \). For each clause, there are 7 truth assignments to the variable in \( C_i \) that satisfy \( C_i \), and 1 truth assignment that does not satisfy \( C_i \). \( S_i \) is the concatenation of the 7 truth assignments (where we use the character a for a variable that does not appear in \( C_i \)) with 6 delimiting characters. For instance:

\[(x_1 \lor x_2 \lor \neg x_3) \rightarrow 100a..aX101a..aX110a..aX111a..aX010a..aX011a..aX000a..a. \]

We set \( f(\cdot, \cdot) \) as follows:

\[
\begin{array}{c|c|c|c|c}
0 & 1 & a & X \\
\hline
0 & 0 & 1 & 1 & \infty \\
1 & 1 & 0 & 1 & \infty \\
a & \infty & \infty & \infty & \infty \\
X & \infty & \infty & \infty & \infty \\
\end{array}
\]

We now prove (??). If \( F \) is satisfiable, then there exists a satisfying assignment \( A \) satisfying each clause. Write out \( A \) as a bit string \( w \), where the \( i \)th character denotes the assignment to variable \( x_i \). The bit-string \( w \) is an \( n \)-mer. Furthermore, since every clause is satisfied, it must be that \( d(w, x_i) = n - 3 \) for the substring \( x_i \) in \( S_i \) that agrees with \( A \) on the variables present in \( S_i \). This proves the forward direction of (??).

\footnote{I came up with this idea on my own, only to find out later that it was published in “Combinatorial approaches to finding subtle signals in DNA sequences, Pevzner and Sze, 2000.”}
Now, we prove the other direction of (\ref{eq:sat}). Suppose \( F \) is unsatisfiable. Then, there exists no satisfying assignment satisfying every clause \( C \). By the structure of \( f \), it is clear that the motif must be a binary string. Furthermore, by the presence of the delimiting symbol \( X \), a motif with \( d(w, x_i) \leq \infty \) cannot overlap \( X \). Thus, the only candidate substrings in each \( S_i \) are the satisfying assignments. However, since every satisfying assignment does not satisfy at least one clause, it must be the case that any bit-string representation of a satisfying assignment has \( d_f(\cdot, \cdot) \) at least \( n - 2 \) with any \( x_i \) in \( S_i \). This completes the proof. 

3 A 2-approximation

Consider the following algorithm for motif-finding.

**Sample-driven algorithm:** Consider each \( l \)-mer appearing in the strings \( S_1, \ldots, S_m \) as a possible motif. Take the \( l \)-mer \( w \) minimizing:

\[
\max_{i=1}^m \min_{x_j \in S_i} d(w, x_j).
\]

This combinatorial algorithm was covered in class and was described as a faster alternative to exhaustive search, i.e trying all \( 4^l \) possible \( l \)-mers. The disadvantage was stated as:

If the true motif is weak and does not occur in the data, then a random motif may score better than any instance of the true motif.

One interesting property of this algorithm, though, is that it is a 2-approximation – if the optimal motif \( w \) has \( d = \max_i \min_{x_j \in S_i} d(w, x_j) \), then the motif \( w' \) found by the algorithm has \( \max_i \min_{x_j \in S_i} d(w, x_j) \leq 2 * d \).

**Lemma 1.** The Sample-driven algorithm is a 2-approximation.

**Proof.** Let \( w \) be the optimal motif with cost \( d \), and let \( x_j = \arg\min_{x_j \in S_i} d(w, x_j) \) for an arbitrary \( S_i \). Then \( d(w, x_j) \leq d \) by the optimality of \( w \). Therefore, for every other \( S_i \), we have by triangle inequality:

\[
d(x_j, S_i) \leq d(x_j, w) + d(w, S_i) \leq 2d.
\]

The lemma suggests one notable heuristic. Consider the substrings \( x_1, \ldots, x_n \) in \( S_1, \ldots, S_n \) such that \( d(w, x_i) \) is minimal among substrings in \( S_i \). Then each such \( x_i \) is a 2-approximation. Since so many substrings "signify" the true motif, we can use this information in a new algorithm to try to find the true motif:

**Clique-algorithm:** Let \( d \) be the cost of the motif found by the sample-driven algorithm. Let \( V \) be the set of all substrings in \( S_1, \ldots, S_m \) with cost at most \( 2d \). Connect an edge \((u, v)\) between two substrings if:

1. \( u, v \) lie in different strings.
2. \( d(u, v) \leq 2d \).

Report the motif associated with the largest clique in \( G = (V, E) \).
Note that above, the $x_i$ form a clique, since they are each at most $2d$ from each other, and all lie in different strings. Of course, finding cliques is NP-complete, but the underlying graph is relatively small, because of the preprocessing done by the sample-driven-algorithm. This algorithm was also suggested in “Combinatorial approaches to finding subtle signals in DNA sequences, Pevzner and Sze, 2000.”

4  Implementation of the sample-driven algorithm

I implemented the sample-driven algorithm in C++ and tested it on the datasets data1, data2, data3, and data4. For the first three datasets, the algorithm both quickly and accurately found the true underlying motifs:

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>motifs found</th>
</tr>
</thead>
<tbody>
<tr>
<td>data1</td>
<td>13 sec</td>
<td>AATTCGAATT, ATTCGAATTC, TTCGAATTCC</td>
</tr>
<tr>
<td>data2</td>
<td>3 sec</td>
<td>AATCTGTCTA, ATCTGTCTAC, CGGTCTACTA, GTCTACTAAG, TCTGTCTACT, TGTCTACTAG</td>
</tr>
<tr>
<td>data3</td>
<td>1 min, 45 sec</td>
<td>AAAAATAGAA, AAAGAATAAA</td>
</tr>
</tbody>
</table>

However, for data4, the algorithm failed to find a few, good motifs. Indeed, the algorithm found roughly 3,000 motifs, all having cost 3. The algorithm was still efficient, however – it only took roughly 7 minutes.

The implementation of the sample-driven algorithm is efficient because of easy optimizations. For example, once we have scored one motif, we can always preempt computation of the score of another motif when we know that its score is greater than the minimum so far. This can greatly reduce the running time.

Clearly, in some circumstances, one can apply the sample-driven algorithm and achieve good results. In other circumstances, however, the algorithm is ineffectual. What distinguishes these two circumstances? What is the relationship between the optimal motif score $d$, and the input parameters $l, m$, and $n$? I attempted to answer this question through the planted-motif model.

5  Planted-motif model

In this model, we have $m$ strings $S_1, \ldots, S_m$. In each $S_i$ is a $l$-mer $x_i$ such that $d(w, x_i) \leq d$ for a fixed $l$-mer $w$. The rest of the characters in the substrings $S_i$ are chosen uniformly and independently from A,G,T,C.

We know that the sample-driven algorithm finds an $l$-mer of cost at most $2d$ when the optimal motif cost is $d$. When $S_1, \ldots, S_m$ are random strings, what is the chance that the sample-driven algorithm finds an $l$-mer in $S_1, \ldots, S_m$ with cost at most $2d$? If this probability is low-enough, then the sample-driven algorithm would actually find the planted $l$-mers. We derive an upper bound on this probability in the next lemma.

**Lemma 2.** Let $S_1, \ldots, S_m$ be random strings of length $n$, and let $w$ be the best $l$-mer motif found by the sample-driven algorithm on $S_1, \ldots, S_m$. Then we have that:

$$P\left(\max_{x_j \in S_i} \min_{x_j} d(w, x_j) \leq d\right) \leq m(n-l)^2 \left(\frac{n(3l)\frac{d+1}{4}^d}{4^l}\right)^{m-1}.$$
Proof. We first evaluate the probability that $S_i$ has some substring within hamming distance $d$ of some fixed string $y$:

$$P(S_i \text{ has a substring } x_j \text{ with } d_H(x_j, y) \leq d) \leq (n - l) \cdot P(d_H(x_j, y) \leq d) \leq n \sum_{i=1}^{d} \frac{(l)^3}{4^i} \leq \frac{n(3l)^{d+1}}{4^l}.$$  

Now, the sample-driven algorithm finds a motif of cost $d$ if some $l$-mer $y_i$ occurring in some $S_i$ has the following property:

$$\forall j \in \{1, \ldots, m\} \setminus \{i\}, S_j \text{ has some substring } x_j \text{ with } d_H(x_j, y_i) \leq d.$$  

We can bound this with by the calculation done above:

$$P(\forall j \in [m] \setminus \{i\}, S_j \text{ has some substring } x_j \text{ with } d_H(x_j, y_i) \leq d) \leq \left(\frac{n(3l)^{d+1}}{4^l}\right)^{m-1}.$$

Here, we used the independence of the $S_j$'s. Now, we union bound over all candidate motifs that the sample-driven algorithm considers to complete the proof:

$$P(\exists y_i \forall j \in [m] \setminus \{i\}, S_j \text{ has some substring } x_j \text{ with } d_H(x_j, y_i) \leq d) \leq m(n - l)^2 \left(\frac{n(3l)^{d+1}}{4^l}\right)^{m-1}.$$  

Unfortunately, this analysis doesn’t give any insight into the motif challenge problem: find a planted 15-mer with hamming distance 4 to some substring in 20 (otherwise) random sequences of length 1000. Indeed, plugging in these parameter values gives an absurd bound. However, the bound does tell us when the sample-driven algorithm almost surely performs well in this model. If instead the planted $l$-mer is of length 40, the sample-driven algorithm will almost surely find it, since the above probability is minuscule ($1.74 \times 10^{-38}$).

To determine if the bound we derived was meaningful, we tested our algorithm on a planted model. When we used the motif challenge parameters ($m = 20$, $n = 1000$, $l = 15$, $d = 4$), the sample driven method found roughly 500 motifs, all with scores of 4. Note that any of the planted motifs would have a score of roughly 8, so it is clear that the sample driven method would fail. However, when we increase the $l$-mer length to 30, the sample-driven algorithm was able to find the planted motifs.

6 Conclusion

We have presented some minor results on the combinatorial motif-finding problem and the sample-driven algorithm. The simplicity of the sample-driven algorithm is appealing, and I believe that further analysis of the sample-driven algorithm could lead to better heuristics – perhaps eventually solving the (15, 4) challenge problem.