

PRUNER: Algorithms for Finding Monad Patterns in DNA Sequences

Ravi VijayaSatya and Amar Mukherjee

School of Computer Science, University of Central Florida, Orlando, FL 32816-2362

(rvijaya, amar)@cs.ucf.edu

Abstract

We present new algorithms for discovering monad patterns in DNA sequences. Monad patterns are of the form $(l,d)-k$, where l is the length of the pattern, d is the maximum number of mismatches allowed, and k is the minimum number of times the pattern is repeated in the given sample. The time-complexity of some of the best known algorithms to date is $O(nt^2l^d|\Sigma|^d)$, where t is the number of input sequences, and n is the length of each input sequence. The first algorithm that we present takes $O(n^2t^2l^{d/2}|\Sigma|^{d/2})$ and space $O(ntl^{d/2}|\Sigma|^{d/2})$, and the second algorithm takes $O(n^3t^3l^{d/2}|\Sigma|^{d/2})$ time using $O(l^{d/2}|\Sigma|^{d/2})$ space. In practice, our algorithms have much better performance provided the d/l ratio is small.

Keywords: Pattern discovery, regulatory patterns, k -mismatch patterns

1. Introduction

Discovering regulatory patterns in DNA sequences is a well known problem in computational biology. Due to mutations and other errors, the actual occurrences of these regulatory patterns allow for a certain degree of error. The general approach to this problem is to take a set of t DNA sequences each of length n , and look for patterns of a certain length l that occur in at least k out of the t sequences with at most d mismatches at each occurrence. The values of l , d and k can be determined either from prior knowledge about the binding site, or by trial and error, trying different values of l and d . These single contiguous blocks of patterns are called *monad* patterns.

Pevzner and Sze [7] have put forward a challenge problem: to find the signal in a sample of sequences, each 600 nucleotides long, each containing an unknown pattern of length 15 with at most 4 mismatches. They presented the WINNOWER and SP-STAR algorithms that could solve this problem, which was not solvable by many of the earlier

techniques. Among other approaches proposed to solve this problem [2,4,5], time-complexity of the best known algorithms [2,5] is $O(nt^2l^d|\Sigma|^d)$. Some statistical approaches have also been proposed [1,3,6,8], which are not always guaranteed to arrive at a correct solution.

Most of these algorithms search the d -mismatch neighborhood of each l -gram in the sample. The size of the d -mismatch neighborhood of an l -gram is $O(l^d|\Sigma|^d)$. The main contribution of our approach is to limit the search to a small portion of the d -mismatch neighborhood, which results in improving the theoretical worst case time complexity to $O(n^2t^2l^{d/2}|\Sigma|^{d/2})$.

2. Previous approaches

The WINNOWER algorithm [7] and the cWINNOWER algorithm [4] are based on graph theory. In these algorithms, a graph is constructed in which each vertex is an l -gram in the input sequence. Two l -grams are connected by an edge if they mismatch in at most $2d$ positions. Now, the problem is mapped to the problem of finding k -cliques in the graph. The problem of finding k -cliques in graph, when $k > 3$ is an NP-complete problem. Therefore, WINNOWER and cWINNOWER try to apply some heuristics to arrive at a solution. In the first step, all the nodes that have a degree less than $k-1$ are removed. After that, different techniques are applied to try to remove the spurious edges in the graph that can not be part of a solution. The complexity of WINNOWER and cWINNOWER for the most sensitive versions of the algorithms are given by $O(t^3n^{2.66})$ and $O(t^4n^4)$, respectively. However, it is important to note even though most sensitive versions of these algorithms solve almost all practical problems, they are not guaranteed to solve a given problem.

3. The PRUNER Algorithm

3.1 Our contributions

Our approach is based on the WINNOWER algorithm [7]. As in WINNOWER, we build a graph based on pair-wise similarity information, and prune the graph eliminating vertices that can not be part of a solution. However, after this point, we employ a different approach. The algorithms try to successively remove edges from the graph, after checking all the patterns that mismatch in at most d positions from both the l -grams that are connected by the edge. We categorize the edges into two groups. Group1 consists of edges that connect l -grams that differ in *more* than d positions, and Group2 consists of edges that connect l -grams that differ in *less than or equal to* d positions. In the following sections, we will show that there will be relatively few patterns that mismatch in at most d positions from both the l -grams that are connected by a Group-1 edge. Precisely, we will show that there will be at most $O(l^{d/2}/|\Sigma|^{d/2})$ such patterns for every Group-1 edge. We present a technique which enumerates all the patterns corresponding to each Group1-edge, checks each one of them to see if they satisfy the search criteria, and removes the Group1-edge. We show that if at any vertex, after all the Group-1 edges are removed, if the degree of the vertex is less than $k-1$, then the vertex can be safely removed from the graph, without affecting any patterns that are not yet examined and reported. After all Group1-edges are removed, this leaves us with a graph in which each vertex has a degree of at least $k-1$, and all the edges that are incident on the vertex are Group2-edges. *Therefore, if the graph has any vertices left, there will be at least k vertices left.* Since the degree of each vertex is at least $k-1$ and each edge is a Group-2 edge, the l -gram corresponding to each vertex has at least $k-1$ other l -grams that mismatch with it in at most d positions. Therefore the l -gram corresponding to each vertex in the graph is itself a solution, and can be reported. Beyond this, there might be other patterns in the graph that meet the search criteria, but in a general case, we assume that there are fewer than k distinct monad patterns in the given sample. In the almost impractical scenario that there are more than k distinct monad patterns, the algorithms we present report at least k of them. Unlike WINNOWER and cWINNOWER[4], our algorithm is guaranteed to find a solution in $O(n^2 l^{d/2}/|\Sigma|^{d/2})$ time using $O(nl^{d/2}/|\Sigma|^{d/2})$ space.

3.2 Problem statement

In the discussion that follows, for convenience in illustration, we treat the input sample as a single sequence of size n . The time and space complexities

are not affected by this simplification. Therefore, the problem can be stated as follows: given a string S of length n over the alphabet $\Sigma = \{A, C, G, T\}$, the problem is to find a pattern P of length l that occurs at least k times in S with at most d mismatches in each occurrence.

3.3 Terms and Definitions

We denote a length- l substring (an l -gram) of S starting at position i in S by L_i . A score $h = D(L_i, L_j)$ indicates the number of positions in which the two l -grams L_i, L_j mismatch. We denote the set of patterns that mismatch with both L_i and L_j in at most d positions by $\rho(L_i, L_j)$. We refer to the set $\rho(L_i, L_j)$ also as the set of patterns that are *consistent* with L_i and L_j . We now describe, briefly, how to compute the size of the set $\rho(L_i, L_j)$. Let P be any pattern such that $P \in \rho(L_i, L_j)$. Now, it is important to note that $\rho(L_i, L_j) = \{\phi\}$ if $h > 2d$, as both $D(L_i, P)$ and $D(L_j, P)$ have to be less than or equal to d . Enumerating all the different possibilities for P $h \leq 2d$, we arrive at the following expression:

$$|\rho(L_i, L_j)| = \sum_{d_c=0}^{\lfloor \frac{2d-h}{2} \rfloor} \left[\binom{l-h}{d_c} 3^{d_c} \sum_{h_1=h-d+d_c}^{d-d_c} \sum_{h_2=h-h}^{d-d_c} \binom{h}{h_1} \binom{h_1}{h_1+h_2-h} 2^{h_1+h_2-h} \right]$$

In the above expression, $|\rho(L_i, L_j)|$ increases when h decreases. When $d < h \leq 2d$, the maximum value of $|\rho(L_i, L_j)|$ occurs when $h = d+1$. When $h = d+1$, the maximum value that d_c can take is given by $d_c = (d-1)/2$ which is equal to $d/2$ when d is odd, and $(d/2)-1$

when d is even. Now, $\binom{l-h}{d_c} 3^{d_c}$ is in $O(l^{d_c} 3^{d_c})$.

Therefore, on the whole, $|\rho(L_i, L_j)|$ is in $O(l^{d/2} 4^{d/2})$.

3.4 The PRUNER-I and PRUNER-II algorithms

In both the algorithms, we construct a graph $G(L, E)$ where each vertex is an l -gram in the input sample, and there is an edge $e(L_i, L_j, D(L_i, L_j))$ connecting two l -grams L_i and L_j if $D(L_i, L_j)$ is less than or equal to $2d$. We then successively remove vertices representing l -grams from the graph $G(L, E)$ that have a degree less than $k-1$, and remove the edges that are incident on these vertices. Until this point, our algorithms are no different from WINNOWER. However, they differ from WINNOWER in the following steps.

Both the PRUNER-I and the PRUNER-II algorithms process each vertex successively. The PRUNER-I algorithm enumerates the consistent patterns for every group1-edge. It then computes how

many times each pattern repeats. It does this by adding all the consistent patterns for each edge to a list, sorting and scanning the list. Each time a pattern appears in the list, it means that the pattern is within d mismatches from another l -gram. Hence, if a pattern repeats $k-1$ times, it means that the pattern is within d mismatches from $k-1$ other l -grams. However, since we have not yet processed the Group2-edges (i.e., edges connecting l -grams that mismatch in d or fewer positions), we can not yet discard the patterns that repeat less than $k-1$ times. We do not want to evaluate all the consistent patterns for the Group2-edges, as there are too many ($O(l^d 4^d)$) such patterns. Therefore, we will have to take each pattern in the list, and compare it with each l -gram that is connected to the current vertex through a Group2-edge. Only then will we know how many times each one of those patterns has repeated. An efficient way of doing all this is presented below.

At each node L_i , we enumerate the consistent patterns $\rho(L_i, L_j)$ for all the Group1-edges, i.e., edges $(L_i, L_j, D(L_i, L_j))$, such that $d < D(L_i, L_j) \leq 2d$. We add these patterns to a list $\eta(i)$, and remove the edge $(L_i, L_j, D(L_i, L_j))$. Lemma 3.1 states that we can safely remove the edge $(L_i, L_j, D(L_i, L_j))$ after enumerating and adding $\rho(L_i, L_j)$ to $\eta(i)$.

Lemma 3.1: After a vertex L_i in $(L_i, L_j, D(L_i, L_j))$ is processed, there can be no new patterns in $\rho(L_i, L_j)$ that were not reported while processing L_i , but will be reported while processing the vertex L_j .

Now, we need to find out how many times each pattern is repeated in $\eta(i)$. An easy way of doing this will be to sort $\eta(i)$, and scan $\eta(i)$. As each pattern in $\eta(i)$ is a length- l string of a fixed alphabet, $\eta(i)$ can be sorted in linear time using radix sort. Let a pattern P repeat m times in $\eta(i)$. Let R be the degree of node L_i after processing and removing all Group1-edges. As explained in section 1, R is expected to be very small. We do the following.

- If $m < k-1 - R$, we discard P . The number of times P repeats can increase by at most R , by comparing P with each one of the Group2-edges. If $m < k-1 - R$, there is no way that P can repeat $k-1$ times. So we can discard P .
- If $m \geq k-1$, report P , since it is clear that P has already occurred at least $k-1$ times.
- If $k-1 - R \leq m < k-1$, we compare P with all l -grams that are still connected to L_i . For each such l -gram that mismatches P in at most d locations, we increment the repeat count of P . If the repeat count reaches $k-1$, we report P . Other wise, we discard P .

Before we leave L_i and proceed to process the next vertex, we can do one more thing – we can

remove the vertex L_i from the graph if $R < k-1$, without ever enumerating the consistent patterns for these edges. Lemma 3.2 proves this.

Lemma 3.2: If the residual degree R of vertex L_i is less than $k-1$ after processing and removing all Group1-edges of L_i , there can be no new patterns that will be reported by processing the Group2-edges.

We are now left with a graph in which the score of each edge is at most d , and degree of each remaining vertex is at least $k-1$. In practice, we do not expect any vertices to remain at this stage, as our assumption is that there are not too many patterns that meet the search criteria. All the l -grams that do remain until this stage are valid solutions, since they mismatch in at most d positions with at least $k-1$ other l -grams. Hence, we report all the remaining l -grams.

The PRUNER-II algorithm is very similar to the PRUNER-I algorithm in concept. However, the PRUNER-II algorithm attempts to eliminate the potentially huge memory requirements of the PRUNER-I algorithm. While processing each node L_i , the PRUNER-I algorithm maintains a list $\eta(i)$ that contains all the patterns that are consistent with each one of the Group1-edges. When the number of such edges is huge, the amount of memory required for $\eta(i)$ may be too big. Especially, this might be the case when d is large and the d/l ratio is large, in which case the graph $G(L, E)$ will be highly connected.

At each vertex L_i , the PRUNER-II algorithm processes edges one by one. For each edge $(L_i, L_j, D(L_i, L_j))$, it enumerates the set of consistent patterns $\rho(L_i, L_j)$. For each consistent pattern $P \in \rho(L_i, L_j)$, if we compare P with all the l -grams that are directly connected with vertex L_i , we can determine if P mismatches in at most d positions with at least $k-1$ of them. However, a deeper analysis reveals that it not necessary to compare P with all the l -grams that share an edge with L_i . For any l -gram L_q , if $D(L_q, P) \leq d$, then $D(L_q, L_j)$ will be less than or equal to $2d$. This means that the l -grams L_q and L_j will also be connected. Therefore, we only need to compare P with all vertices L_q such that the edge $(L_q, L_j, D(L_q, L_j)) \in E$. If at least $k-2$ of them mismatch with P in fewer than d positions, it reports P . Otherwise, P is discarded. As in the PRUNER-I algorithm, it removes the edge $(L_i, L_j, D(L_i, L_j))$ after checking all the patterns in $\rho(L_i, L_j)$.

3.5 Complexity analysis

Building the graph involves calculating the mismatch count for each l -gram pair (L_i, L_j) such that L_i and L_j are derived from different input sequences. There are $(n-l+1)$ l -grams for each input sequence, and $n(l-1)$ other l -grams for each l -gram in the input sequence. Therefore, building the graph takes $O(n^2 t^2)$.

Pruning the graph involves removing all the edges incident on each vertex whose degree is less than $k-1$. In the worst case, we might have to delete all the nodes, so the maximum number of edges that need to be removed is $((k-1)*nt - 1)$, which is $O(ntk)$. This time is common for both PRUNER-I and PRUNER-II.

In the PRUNER-I algorithm, each l -gram can have up to $n(t-1)2d$ -mismatch neighbors. Therefore, at each l -gram, we might have to enumerate the consistent patterns with $n(t-1)$ other l -grams. The maximum number of these consistent patterns as discussed in section 3.3, is $O(l^{d/2}4^{d/2})$. Hence the worst-case time complexity at each node is given by $O(nl^{d/2}4^{d/2})$. We need to store all these patterns in a list, so we need $O(nl^{d/2}4^{d/2})$ space. In the worst case, we will have to process $(t-k+1)n$ l -grams, since no new patterns can be discovered after removing all l -grams from $(k-1)$ l -grams. Therefore, the overall complexity is given by $O(n^2t(t-k+1)l^{d/2}4^{d/2})$. If k is small w.r.t. t , this will be $O(n^2t^2l^{d/2}4^{d/2})$. When $k = t$, the complexity of the PRUNER-I algorithm is $O(n^2tl^{d/2}4^{d/2})$.

In case of the PRUNER-II algorithm, each edge is processed separately. All the patterns consistent with each edge (L_i, L_j) have to be compared with all the l -grams that are connected to both L_i and L_j . In the worst case, there can be $n(t-2)$ vertices that are connected to both L_i and L_j . The total number of the edges could be $n^2t(t-1)$ in the worst case. The edge can have $O(l^{d/2}4^{d/2})$ patterns that are consistent with it, so the total time taken will be $O(n^3t^3l^{d/2}4^{d/2})$. Each pattern could be compared separately; therefore the space needed is approximately the same as that necessary for the graph.

4. Results

The algorithms were tested on generated samples containing 20 sequences of 600 nucleotides each. The sequences were implanted with randomly mutated patterns at randomly chosen positions. The tests were carried out on a Pentium-4 3.2 GHz PC with 2GB of memory, running Redhat Linux 9.0. The performance of the algorithms for different values of l and d are shown in Table 4.1. Both the algorithms perform very well on the challenge problem, solving it in around two minutes. It can be seen from the results that the algorithms react very sensitively to the d/l ratio. As d/l ratio increases, there will be more edges in the graph, and consequently more patterns to search. On the other hand, there will be larger number of consistent patterns with each edge as l increases. Therefore, the performance of the algorithms is not very impressive when both the d/l ratio and l are big. However, it can be seen that PRUNER-II performs very well even for large values of l , as long as the d/l ratio is small.

5. Conclusion

Most of the algorithms proposed earlier take longer time as l and d increase. In contrast, as our algorithms perform very well on sparse graphs, irrespective of the sizes of l and d , WINNOWER can be used as a first step in order to eliminate most of the edges, and the PRUNER algorithms can be used as a second step, in order to quickly arrive at accurate results.

Table 4.1: Performance of the algorithms

Test Case (l, d)	d/l	PRUNER-I		PRUNER-II	
		Time (min)	Memory (MB)	Time (min)	Memory (MB)
13,3	0.23	0.17	43	0.14	43
13,4	0.31	12.26	166	29.58	278
14,4	0.28	5.35	198	7.09	178
15,4	0.27	2.28	122	1.34	91
16,4	0.25	0.56	51	0.56	43
16,5	0.31	69.00	540	284.59	247
17,5	0.29	29.54	315	36.58	161
18,5	0.28	13.19	174	13.19	92
18,6	0.33	*	*	2880.00**	457
24,6	0.25	1.12	720	0.16	11
28,7	0.25	*	*	0.36	640

* -- Out of memory, ** -- expected run time

6. References

- [1] Buhler J. and Tompa B. (2001) "Finding Motifs Using Random Projections" *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology (RECOMB01)*, 2001 pp.69-76
- [2] Eskin E., Pevzner P. A. (2002) Finding Composite Regulatory Patterns in DNA Sequences.' *In Proceedings of the Tenth International Conference on Intelligent Systems for Molecular Biology (ISMB-2002)*. Edmonton, Canada: August 3-7, 2002
- [3] Hertz, G. and Stormo, G. (1999) "Identifying DNA and protein patterns with statistically significant alignments of multiple sequences" *Bioinformatics*, 10, 1205-1214.
- [4] Liang S. (2003) "cWINNOWER Algorithm for finding fuzzy DNA Motifs" *CSB2003* Aug 11-14 2003 pp. 260-265.
- [5] Marsan L. and Sagot M. (2000) "Algorithms for extracting structured motifs using suffix tree with applications to promoter and regulatory site consensus identification" *Journal of Computational Biology*, 7, 345-360.
- [6] Pavesi, G., Mauri, G. & Pesole, G. (2001) "An algorithm for finding signals of unknown length in DNA sequences" *Proceedings of the Ninth international Conference on Intelligent Systems for Molecular Biology*.
- [7] Pevzner, P. A. & Sze, S. (2000) "Combinatorial approaches to finding subtle motifs in DNA sequences" *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*. pp. 269-278.
- [8] Price A., Ramabhadran S. and Pevzner A. "Finding Subtle Motifs by Branching from Sample Strings" *Bioinformatics* 19, 149-155