An Algorithm for Finding Novel Gapped Motifs in DNA Sequences [*]

Emily Rocke   Martin Tompa

Department of Computer Science and Engineering
Box 352350
University of Washington
Seattle, WA, U.S.A. 98195-2350

{ecrocke,tompa}@cs.washington.edu

October 25, 1999

[*] This material is based upon work supported in part by the National Science Foundation and DARPA under Grant DBI-9601046.

Abstract

We propose a novel algorithm that, given a long DNA sequence, finds approximate repeats, allowing general insertions and deletions. The idea is based on the Gibbs sampling approach of Lawrence *et al.* The novelty of our algorithm stems largely from our handling of gaps in the motif occurrences. Another contribution is the use of relative entropy as a natural scoring function for alignment of the motif occurrences. We describe the results of experiments with this algorithm on the full *H. influenzae* genome.

1. Introduction

With large-scale DNA sequencing under way, many of the questions on the horizon deal with sequence analysis. Once an organism's genomic DNA is sequenced, how can we discover which of its subsequences are functional, and what their functions are?

For some of these questions, significant progress has already been made. For instance, the question of locating the genes has received much attention, and is well understood in some instances, particularly in the case of prokaryotes.

Less well understood is the question of identifying the multitude of protein binding sites, regulatory sequences, and sequences involved in replication, three-dimensional folding, nuclear anchoring, and organization. By this we do not mean locating instances of known regulatory sequences, such as promoters, for which there has been much successful work (see, for example, [1, 5, 10, 13, 14]). Rather, we are interested in identifying novel functional sequences.

One approach that has been suggested to tackle this problem is to find sequence motifs, that is, approximately repeating substrings that occur more frequently than expected by chance. (As a simple example that frequency sometimes implies functionality, the most frequent sequences in the *H. influenzae* genome contain the uptake signal sequence or its reverse complement [7].) Such a motif could be hypothesized to have functional significance, and a laboratory experiment could then be devised to test that hypothesis. In addition to possible functional significance, these repeats provide valuable information on the evolutionary history of the DNA.

Thus, we are interested in a general computational tool that, given a long DNA sequence, identifies frequently occurring substrings. This tool should adaptively find arbitrary numbers of occurrences of the motif, find motifs of arbitrary length, and allow general insertions and deletions in occurrences. Although the question of locating repeats has been studied by many researchers (see, for example, [2, 3, 9, 16]), we do not know of any work that describes a tool this general. In particular, we are not aware of any work that finds approximate repeats allowing general insertions and deletions.

We propose a novel algorithm that finds approximate repeats with general insertions and deletions, and adaptively adjusts the length and number of occurrences. The idea is based on the Gibbs sampling method of Lawrence *et al.* [8], which is summarized in Section 2. Their method does not allow for general insertions and deletions; the novelty of our algorithm stems largely from our handling of gaps in the motif occurrences, which is discussed in Sections 3 – 5. Another contribution discussed in Section 3 is the use of relative entropy as a natural scoring function for alignment of the occurrences.

We have implemented the algorithm and done some experimentation on the noncoding regions of the full *H. influenzae* genome. Some of these experimental results are presented in Section 6.

2. The Gibbs Sampler of Lawrence *et al.*

Lawrence *et al.* [8] proposed an algorithm for local multiple alignment of t protein sequences. We will describe their algorithm for the simple case of ungapped alignments consisting of a substring of fixed length w from each of the t proteins. We will refer to w as the *width* of the alignment. (Lawrence *et al.* eventually generalize to allow w to vary, to allow the alignment to contain more than one substring from each of the proteins, and to accommodate limited spacers, but we will not discuss these here.)

Their basic algorithm iterates many times over the following steps. Each iteration discards one sequence from the alignment and replaces it with a new one, as follows:

1. At the beginning of each iteration, a substring of length w from each of the t proteins has been identified.

2. Choose one of the t proteins P uniformly at random.

3. Create a $20 \times w$ frequency matrix Q from the remaining $t-1$ substrings of length w. That is, $Q_{r,j}$ is the frequency with which residue r occurs in the j-th position over all $t-1$ substrings of length w, for $0 \leq j < w$. (The authors use "pseudocounts", but we will ignore that technicality for the purpose of this overview.)

4. For each residue position i in P, let P_i be the substring of P of length w starting at position i. There are two natural competing models that might generate P_i, namely Q and the background distribution B. Let p_i be the likelihood ratio of these two models. That is, $p_i = \prod_{j=0}^{w-1}(Q_{r_{i+j},j}/B_{r_{i+j}})$, where r_k is the residue at position k of P, and B_r is the background frequency of residue r.

5. Choose the starting position i of the new substring in P randomly, with probability proportional to p_i.

3. Adapting to DNA Sequences, and Accommodating Gaps

Assume for the moment that t is fixed. It is straightforward to adapt the algorithm of Section 2 to find t ungapped, nonoverlapping, approximately matching substrings of length w in a single long DNA sequence (which we will call the "genome" for convenience). Finding a good way to accommodate insertions and deletions is less straightforward.

Ideally, what one would like to do in place of steps 3 and 4 of the algorithm is to compute, for each position i in the genome, a multiple alignment of t sequences of length w (the $t-1$ that remain, plus the one starting at position i). In step 5 one would then choose the starting position i of the new substring based on the scores of those multiple alignments. This, of course, is prohibitive: multiple alignments are either slow or approximate, and we would be asking for one multiple alignment per residue in the genome per iteration.

On further reflection, it is reasonable to expect that, at the beginning of each iteration, we have not only $t-1$ selected substrings, but also (inductively) a good alignment A of width w of those substrings. The problem then is much more feasible: for each position i in the genome, compute an optimal *pairwise* alignment of A and the substring of length w beginning at i. (Actually, we do not know that the length of the substring should be exactly w, because of insertions and deletions. We will return to this question in Section 4.) For this pairwise alignment, the standard dynamic programming algorithm [12, 15] is efficient.

Such a pairwise alignment, though, requires an appropriate scoring function $\sigma(r, j)$, where r is a single residue (or gap), and j is a column of A, containing $t-1$ residues and gaps. Let us ignore momentarily the possibility of gaps in r and j.

Let j' be the column of t residues consisting of the residues in j plus r. For $k \in \{A, C, G, T\}$, let $P_k^{j'}$ be $1/t$ times the number of occurrences of residue k in j', and let B_k be the background frequency of residue k in the genome. We will define the scoring function $\sigma(r, j)$ to be the relative entropy $D(P^{j'} \parallel B)$. That is,

$$\sigma(r, j) = \sum_{k : P_k^{j'} \neq 0} P_k^{j'} \log_2(P_k^{j'}/B_k).$$

Why is this a reasonable scoring function? First, note that it treats r symmetrically with the remaining residues in the column j, so that r does not carry undue weight. Second, imagine that the distributions $P^{j'}$ and B represent competing models for the residue r under consideration; that is, r is either generated by the distribution of column j', or by the background process, respectively. Then the relative entropy $D(P^{j'} \parallel B)$ measures the expected log likelihood ratio of these two models. The more different the distributions $P^{j'}$ and B, the greater the score $\sigma(r, j)$.

How should this scoring function be extended to the case in which j' may contain gaps? Here we experimented with several natural possibilities, and settled on substituting the background distribution B for each gap in j'. That is, when computing $P_k^{j'}$, we add gB_k to the number of occurrences of k in j', where g is the number of gaps in j'.

As a final note on the scoring function, $\sigma(r, j)$ as defined is always nonnegative. This means that adding more columns in the alignment necessarily increases the score, which rewards even poorly conserved columns and the insertion of many gaps. To counteract this, we normalize by subtracting the expected value of $\sigma(r, j)$ from each column's score, so that the expected score is now 0. In addition, we subtract a small penalty for each gap in the column, to reflect the fact that substitutions are preferable to insertions and deletions in DNA motifs. This penalty is under user control, which provides some flexibility in this matter.

4. Improving the Dynamic Programming

With the scoring function in hand, recall our subgoal: given an alignment A of $t-1$ sequences, we want to find the best pairwise alignment of A with the sequence starting at position i of the genome, for each i. This raises two problems:

1. If the width of A is w, we should consider approximately w residues starting at i, but the exact number for the best alignment is unclear: insertions or deletions could make it more or less than w.

2. If the length of the genome is G, then the proposed algorithm requires time $\Theta(w^2 G)$ per iteration, since each pairwise alignment uses time quadratic in the length w of its input sequences. The alignments corresponding to consecutive values of i seem to be doing redundant work.

Both of these problems are solved by the following modification. Consider the $(w+1) \times (G+1)$ dynamic programming table that computes the optimal pairwise alignment of A and the entire genome, where every entry in row 0 of the table is initialized to 0. Then the score in row w and column i is the score of the best alignment *ending* at position i. (This is similar to an "end-space free" alignment [4, Section 11.6.4].) Note that this simultaneously solves the problems above:

1. The length of the best alignment ending at position i (and the best alignment itself) is determined automatically, by tracing back in the table from row w in column i to row 0 [4, Section 11.3.3].
2. The time per iteration is improved to $O(wG)$.

Of course, there is no need to keep the entire $(w+1) \times (G+1)$ table in memory at once: the scores in each column only depend on the scores in the previous column. In order to perform the traceback that constructs the alignment itself, we actually retain approximately w consecutive columns of the dynamic programming table.

5. Summary of the Motif Algorithm

Here is a summary of each iteration of the motif algorithm:

1. At the beginning of each iteration there is an alignment A' of t nonoverlapping substrings of the genome. This alignment may contain gaps.
2. Choose a substring to remove, leaving an alignment A of $t-1$ substrings.
3. Make one dynamic programming pass over the genome, computing the score p_i of the best alignment of A with the substring ending at position i, for each i in the genome.
4. Use p_i to choose a new nonoverlapping t-th substring, and the dynamic programming traceback [4, Section 11.3.3] to align it with A, yielding the alignment A' for the next iteration.

We must also specify an evaluation function for the $t \times w$ alignment A', for the purposes of measuring progress and controlling the number of iterations. We use the obvious evaluation function, namely the sum, over all columns j' of A', of the score of column j'. If P denotes the probability distribution of w consecutive residues, where the distribution of the residue corresponding to column j' is $P^{j'}$, then this evaluation function gives an approximate measure of the likelihood of finding a site generated by distribution P in a genome with background distribution B.

The algorithm summary above does not specify how the choices in steps 2 and 4 are made. There are several combinations of possibilities:

1. The new t-th substring added to the alignment in step 4 can be chosen deterministically (maximum p_i) or randomly (with probability proportional to p_i). Our experiments so far have been done using the deterministic choice for this step.

2. The substring removed in step 2 can be chosen deterministically (maximum evaluation function, over all t choices of removed substring) or randomly (either uniform, or with probability proportional to the evaluation function with the sequence removed). Our experiments so far have been done using uniform random choice for this step.

Of course, the width w of the alignment will vary adaptively from iteration to iteration, due to insertions and deletions in the pairwise alignment. It is a simple matter to allow t, the number of instances of the motif, to vary adaptively as well: t can be increased by skipping step 2, which removes one of the t substrings, or decreased by removing two substring in that step. The decision of when to allow t to change can be controlled by the evaluation function.

6. Experiments on *H. influenzae*

We have implemented the algorithm and done some experimentation on the noncoding regions of the full *H. influenzae* genome. (Karlin [6] identifies many repeats in this and other bacterial genomes.) We focus on the noncoding regions because the novelty of our method lies in allowing insertions and deletions; motif instances with point insertions or deletions within coding regions are unlikely to be of biological interest, since they cause reading frame shifts. Thus we first removed any sequences from the full genome that are coding regions on either strand, reducing the sequence length from 1.8 Mb to 350 kb. We inserted a delimiter at the end of each noncoding region, to prevent motif instances from crossing such points where a coding sequence had been excised. We then doubled the sequence by appending its reverse complement, to allow instances of the motif on either strand. We will call the resulting sequence "the genome" in what follows.

6.1. Experiments Using the Basic Algorithm

Each run of the algorithm begins by initializing the rows of A' to t_0 randomly chosen ungapped substrings each of length w_0 from the genome, where t_0 and w_0 are under user control. The algorithm then iterates as summarized in Section 5. (In the experiments reported here, we did not allow t to vary over the course of these iterations.) Since our implementation of step 4 adds to the alignment the substring that maximizes the score p_i (see Section 5), we use the following termination condition: the algorithm continues as long as the alignment contains a row r such that, if the next iteration chose to remove r, it would be replaced with a different substring. We will call the alignment at which the algorithm terminates the *motif.*

```
0 CAT < CGCCCTTTCA > CGG   at position 118666.
1 CAT < CGCCCTTTCA > CGG   at position 642660.
2 AAT < CGCCTTTTCA > AAA   at position 425287.
3 ATC < CGCCC-TTCA > TGA   at position 330462.
4 TTG < CGCCC-TTCA > CTA   at position 558509.
5 AAC < CGCCCATTCA > ATC   at position 237890.
6 GCA < CGCCC-TTCA > CGT   at position 495353.
7 TCT < CGCCTTTTCA > TTG   at position 34553.
8 CAT < CGCCCTTTCA > CGG   at position 677174.
9 GCA < CGCCC-TTCA > GGG   at position 222102.
```

Figure 1: A sample motif (score 16.6) produced by the basic algorithm

Figure 1 shows an unexceptional example of a motif output by this first version of the algorithm. The motif instances are shown within angle brackets, and some context of each instance is shown outside. The position numbers are with respect to the noncoding portion of the genome only.

With t fixed at 10 and w initialized to $w_0 = 10$, we ran 160 trials of this version of the algorithm on the genome, and 160 trials on a random sequence of the same length and background distribution as the genome, and with the same simulated noncoding region lengths. The results are summarized in the histograms of Figure 2. The horizontal axis represents motif scores (the "evaluation function" of Section 5), and the vertical axis shows how many of the 160 trials resulted in a motif that achieved that score. These histograms suggest that even this simple version will find motifs in a real genome that would not be expected to occur by chance.

6.2. Refining the Motif Window

The sample motif shown in Figure 1 is typical in many ways of the motifs produced by this simple version:

1. There are a few misfit sequences in the alignment (such as sequence 5 in Figure 1).

2. The remaining sequences may be instances of two or three close but distinct motifs. In Figure 1, for instance, sequences 0, 1, and 8 are identical, including their contexts; sequences 6 and 9 are identical, including their left contexts. Similar context is often a quick way to see which sequences belong together.

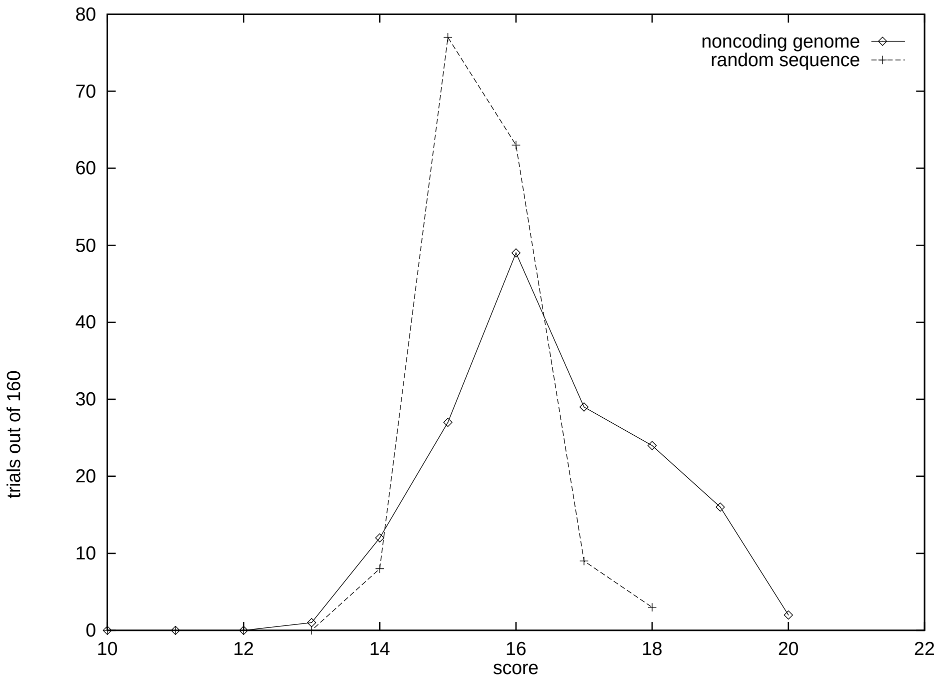

Figure 2: 160 trials of the basic algorithm on the noncoding genome vs. a random sequence

```
0 GGA < CATCGCCCTTTCACGG > CGG   at position 118663.
1 GGA < CATCGCCCTTTCACGG > CGG   at position 642657.
2 GCT < GCACGCCC-TTCAGGG > TTC   at position 222099.
3 GGA < CATCGCCCTTTCACGG > CGG   at position 677171.
4 AAA < GCACGCCC-TTCACGT > AAT   at position 495350.
```

Figure 3: The motif of Figure 1 after rewindowing (score 20.8)

3. The contexts of the instances of each of these distinct motifs suggest that the motifs should be extended.

As a result of these observations, our next version of the algorithm "rewindows" the motif obtained by the algorithm above. This operation selects a subset of the rows, and readjusts the left and right boundaries of the motif, possibly extending into the context, so as to create the maximum resulting score. (To avoid considering all subsets of the rows, our current implementation uses a greedy heuristic that approximates this maximum score.) The result of rewindowing the motif of Figure 1 is shown in Figure 3.

The process of iterating the algorithm of Section 5 until it terminates, followed by rewindowing, will be called a *phase*. Figure 3 suggests that it may be profitable to use the rewindowed alignment as the input to the next phase.

We repeated the experiment of Figure 2, but this time each trial performed two phases of the algorithm. The results are summarized in the two histograms of Figure 4. The difference between the distribution of scores on the genome and on the random sequence is now much more pronounced, indicating that the algorithm is succeeding more often in finding motifs that would not be expected to occur by chance.

We discovered that the peak around score 35 is due largely to a noncoding sequence of 5.3 kb that occurs in six nearly identical instances. Although these six instances account for only 9.2% of the noncoding genome, more than 50% of the 160 trials resulted in motifs clearly identifying some portion of these six instances. An inspection of the GenBank entry [11] reveals that each of the six instances is a cluster of three different rRNA genes, plus some short interspersed tRNA genes that account for the differences among the six instances. These eighteen rRNA genes account for all the annotated rRNA genes in the *H. influenzae* genome.

We removed the entire noncoding regions containing these long repeats, reducing the genome length from 690 kb to 610 kb, and repeated the experiment. The result is shown in Figure 5. Even with these long repeats removed, there are still many trials in which the algorithm found unusually high-scoring motifs.

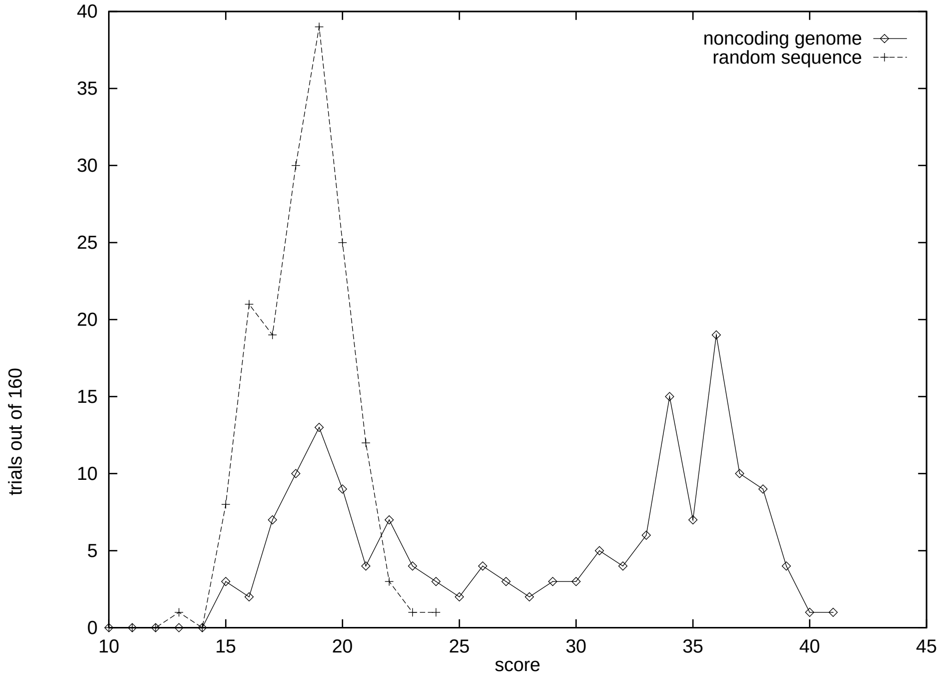

Figure 4: 160 trials of the two-phase algorithm on the noncoding genome vs. a random sequence

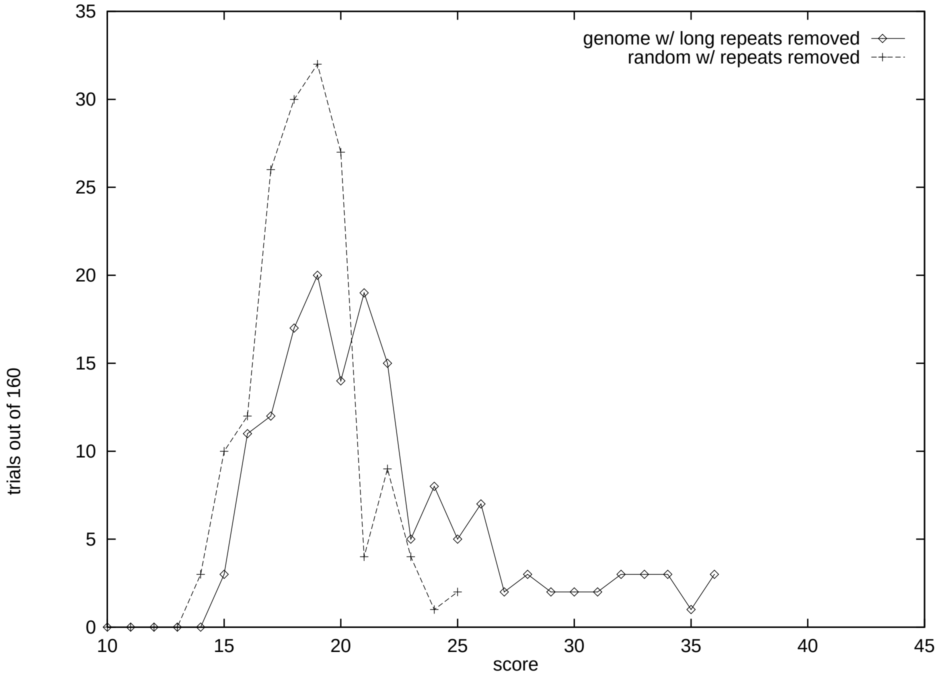

Figure 5: 160 trials of the two-phase algorithm on the noncoding genome with long repeats removed vs. a random sequence

```
0 TCG < GCAGCTCCCCCCATAAATGG > GTG   at position 449120.
1 TCG < GCAGCTCCCCCCATAAATGG > GTG   at position 448927.
2 GCG < ACAGCTCCCCCCATAAATGG > GTG   at position 232857.
3 GCG < CCAGCTCCC-CCGTAAACGG > GTG   at position 88280.
```

Figure 6: A sample motif (score 25) produced by two phases

```
0 TAT < CCCCCCTCA--C-CTTCG-G-CAGCTCCCCCCATAAATGGGTGGAGCCAAGAT > TAG   at position 449105.
1 ATC < CCCCCCTCA--C--TTCG-G-CAGCTCCCCCCATAAATGGGTGGAGCCAAGAT > TAG   at position 448913.
2 GTA < TCCCCCTCAGTCACTTCGCGACAGCTCCCCCCATAAATGGGTGGAGCAAAGTT > AAT   at position 232837.
3 AAT < CCCCCCTCAGTC--TTCGCGCCAGCTCCCCCG-TAAACGGGTGGAGCCAAGGG > ATC   at position 88262.
```

Figure 7: The motif of Figure 6 after seven phases (score 62)

Figure 6 shows a sample motif produced by the two-phase version of the algorithm with the long repeats removed. It is again clear from the context that the complete motif is wider than shown. To discover its extent, we performed five additional phases of the algorithm, which resulted in the motif shown in Figure 7. Although a few columns at each end of this motif might be altered by hand, the remainder is quite suggestive. (Sequences 0 and 1 continue to be identical for another 55 residues beyond the right end of Figure 7, and approximately equal, with only 5 errors, for an additional 44 residues.)

7. Closing Remarks

There are numerous variations of the basic algorithm given in Section 5, all appealing to explore:

1. The new t-th substring added to the alignment in step 4 can be chosen randomly (with probability proportional to p_i).

2. The substring removed in step 2 can be chosen deterministically (maximum evaluation function, over all t choices of removed substring) or randomly (with probability proportional to the evaluation function with the sequence removed).

3. The case of random removal and random addition corresponds to Lawrence *et al.*'s Gibbs sampler. There are also possibilities of extending this to a Metropolis algorithm (accept a decrease in evaluation function only with some probability less than 1), or a full simulated annealing algorithm.

Acknowledgement

We thank Phil Green for suggesting the problem, and for his expertise, guidance, and encouragement.

References

[1] L. R. Cardon and G. D. Stormo. Expectation maximization algorithm for identifying protein-binding sites with variable lengths from unaligned DNA fragments. *Journal of Molecular Biology*, 223:159–170, 1992.

[2] Y. M. Fraenkel, Y. Mandel, D. Friedberg, and H. Margalit. Identification of common motifs in unaligned DNA sequences: application to Escherichia coli Lrp regulon. *Computer Applications in the Biosciences*, 11(4):379–387, 1995.

[3] D. J. Galas, M. Eggert, and M. S. Waterman. Rigorous pattern-recognition methods for DNA sequences: Analysis of promoter sequences from Escherichia coli. *Journal of Molecular Biology*, 186:117–128, 1985.

[4] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.

[5] P. B. Horton and M. Kanehisa. An assessment of neural network and statistical approaches for prediction of E. coli promoter sites. *Nucleic Acids Research*, 20(16):4331–4338, 1992.

[6] S. Karlin. Assessing inhomogeneities in bacterial long genomic sequences. In *RECOMB97: Proceedings of the First Annual International Conference on Computational Molecular Biology*, pages 164–171, Santa Fe, NM, Jan. 1997.

[7] S. Karlin, J. Mrázek, and A. M. Campbell. Frequent oligonucleotides and peptides of the *Haemophilus influenzae* genome. *Nucleic Acids Research*, 24(21):4263–4272, 1996.

[8] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, 8 October 1993.

[9] M.-Y. Leung, B. E. Blaisdell, C. Burge, and S. Karlin. An efficient algorithm for identifying matches with errors in multiple long molecular sequences. *Journal of Molecular Biology*, 221:1367–1378, 1991.

[10] K. Nakata, M. Kanehisa, and J. V. Maizel, Jr. Discriminant analysis of promoter regions in Escherichia coli sequences. *Computer Applications in the Biosciences*, 4(3):367–371, 1988.

[11] National Center for Biotechnology Information. Haemophilus influenzae Rd complete genome. `ftp://ncbi.nlm.nih.gov/genbank/genomes/bacteria/Hinf/hinf.gbk`, Nov. 1997.

[12] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[13] D. S. Prestridge. Predicting Pol II promoter sequences using transcription factor binding sites. *Journal of Molecular Biology*, 249:923–932, 1995.

[14] K. Quandt, K. Frech, H. Karas, E. Wingender, and T. Werner. MatInd and MatInspector: New fast and versatile tools for detection of consensus matches in nucleotide sequence data. *Nucleic Acids Research*, 23:4878–4884, 1995.

[15] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, Mar. 1981.

[16] F. Wolfertstetter, K. Frech, G. Herrmann, and T. Werner. Identification of functional elements in unaligned nucleic acid sequences by a novel tuple search algorithm. *Computer Applications in the Biosciences*, 12(1):71–80, 1996.