A Branch and Bound Algorithm for Local Multiple Alignment

Paul Horton

Computer Science Division
University of California
Berkeley, CA 94720

Abstract

A Branch and Bound Algorithm has been developed to find a set of window positions in a compilation of sequences with globally maximal information content. We have also developed an algorithm for brute force evaluation of solutions which is faster by a factor of the length of the windows than the naïve brute force algorithm. The combination of these two algorithms allows us to solve problems to optimality that were previously amenable only to heuristic algorithms.

1 Introduction

In recent years the amount of DNA and protein sequence available to computer analysis has increased dramatically. The availability of such sequence data has inspired formulations of local multiple sequence alignment problems, which are designed to have solutions that give the positions of local patterns within a collection of sequences. In this paper we introduce two algorithms for solving Stormo and Hartzell's formulation of local multiple sequence alignment [1], whose objective function relates to the entropy of the proposed patterns. This paper is divided into sections with the following purposes:

- Describe the local multiple sequence alignment formulation in more detail.
- Introduce an exhaustive algorithm.
- Introduce a modification of an earlier heuristic search algorithm.
- Introduce a branch and bound algorithm.
- Prove the validity of the bound used.

- Describe precomputation and an additional heuristic speedup.
- Describe the data sets used in this study.
- Summarize and discuss the results of the algorithms on the data sets.
- Suggest future directions for research and conclude.

2 Problem Formulation

The formulation of local multiple sequence alignment which we use was proposed by Stormo and Hartzell [1]. Informally the idea is to select a substring or *window* for each string such that the distribution of the ith character in each window is far away from the expected distribution based on a prior distribution of characters. More formally, the input is a set of n strings $S_1, S_2, \ldots, S_n$ over a fixed alphabet A, and an integer window length w such that w is between 1 and the length of the shortest string. Let V specify one position in each string, i.e. V is a vector of length n whose elements v_i are integers satisfying $\forall i\ 1 \leq v_i \leq l_i$, where l_i is the length of string i. The frequency of a character b in V is denoted by the matrix entry $F(V, b)$, where $\forall V\ \sum_{b \in A} F(V, b) = 1$, and the vector of frequencies for all the characters at a given set of positions V is denoted $F(V)$. The prior probabilities of the characters is given by a vector P, with element p_b of P representing the prior probability of character b. We can now define the information content (also known as the relative entropy) introduced by Schneider *et al* [2] which provides a measure of the distance between the distributions $F(V)$ and P. The function is given by

$$I(V) = \sum_{b \in A} F(V, b) * log_2(F(V, b)/p_b).$$

Schneider *et al* denote this function $R^*_{sequence}$ and use it to extend the standard information theory entropy,

$$H(\, F(V)\,) = -\sum_{b \in A} F(V, b) * log_2 F(V, b),$$

to include prior probabilities. Bailey [3] shows that $I(V)$ can be interpreted as the ratio of the likelihood that a random character generator that generates characters according to the distribution $F(V)$ would generate the n characters observed at position V to the likelihood that a generator using the distribution P would generate those n characters.

We wish to maximize $F(V)$ over a window of length w. Thus our evaluation function becomes

$$E(V) = \sum_{i=0}^{w-1} I(V + i),$$

where we abuse notation to denote $(v_1+i, v_2+i, \ldots, v_n+i)$ as $V+i$ for a scalar i, and define $E(V)$ only for V such that $\forall i\ 1 \leq v_i \leq l_i - w + 1$ to prevent the window from extending past the end of the sequences. The alignment problem then is to find a V such that $E(V)$ is maximal.

3 An Exhaustive Algorithm

Let l be the length of the longest string. A naïve algorithm would calculate the frequencies for the characters in the n strings for each of the w columns covered by each possible window position. There are $O(l^n)$ possible window positions so this would require $O(w(l^n))$ calculations of F and $O(nw(l^n))$ time. However, it is possible to remove the factor of w from the running time. The main observation is this: Let V be a set of window positions such that $\forall i\ v_i \leq l_i - w$, and $V' = V + 1$. In other words let V' be V shifted one base to the right. Then $E(V')$ may be obtained as follows:

$$E(V') = E(V) - I(V) + I(V + w).$$

Using this observation to good advantage the algorithm is straightforward:

$Max\_E \longleftarrow -\infty$

Iterate over all starting vectors V such that at least one element is equal to one and for all elements v_i of V, $1 \leq v_i \leq l_i - w + 1$.

$E\_V \longleftarrow \sum_{i=0}^{w-1} I(V+i)$

Do

$Max\_E \longleftarrow \max\{Max\_E, E\_V\}$
$E\_V \longleftarrow E\_V - I(V) + I(V+w)$
$V \longleftarrow V + 1$

Loop while no element v_i of V exceeds l_i

Return Max_E

There are $l^n - l^{n-1} = O(nl^{n-1})$ starting vectors when at least one element is constrained to equal one. Each starting vector requires $O(wn)$ time for initialization. For all other vectors, $E(V)$ can be calculated by adding the information content of one column and subtracting the information of another (actually the

value of $I(V + w)$ can be cached to avoid the need to calculate $I(V)$ w iterations later, thus reducing the information content calculation to one column per iteration), which requires $O(n)$ time per vector. Thus the overall running time is $O(n^2 w l^{n-1} + n l^n)$. Which is $O(n l^n)$ as long as nw is not much greater than l. In practice for DNA sequences with $n \leq 7$ we precompute the value of $I(V)$ for all possible vectors V thus reducing the time to $O(l^n)$; if array lookups are counted as constant time operations.

4 Heuristic Search

The optimization problem we are concerned with can be formulated as a search problem over a search tree whose leaves represent each possible choice of window positions. More formally, the search tree can be constructed as follows: start with a root at level 0, then give the root $l_1 - w + 1$ child nodes representing each possible position for a window in the first sequence. Likewise for i from 1 to $n - 1$ give each node at level i $l_{i+1} - w + 1$ child nodes to represent the possible choices of the window position in S_{i+1}. There is a one to one correspondence between paths from the root to leaves in the tree and the possible assignments of the window positions for the sequences.

Stormo and Hartzell [1] used a form of greedy search on the search tree as a heuristic for quickly finding good, but not necessarily optimal, solutions. Their algorithm actually performs $l_1 - w + 1$ greedy searchs in parallel, one starting from each node at level 1 of the tree, and returns the best solution found. The greedy search can be described iteratively from $i = 1$ to $n - 1$. When the level of the current node is i, then for each child of the current node, calculate the function $E(v_1, v_2, \ldots, v_{i+1})$ (simply denoted E for the rest of this section) for $S_1, S_2, \ldots, S_{i+1}$, where $(v_1, v_2, \ldots, v_{i+1})$ is the set of window positions implied by the path from the root to the child node. If there is one child whose path maximizes E then make that child the current node and iterate. Otherwise in the case of a tie between the children at level $i+1$ for maximizing E then "split" the search into independent greedy searches for each child starting with that child as its current node.

We applied a similar heuristic that can be described as a kind of beam search. This type of beam search was suggested for multiple alignment problems by Bacon and Anderson [4]. The algorithm descends the tree in a breadth first search fashion except that at each level of the tree it only keeps the nodes corresponding to the top k values of E for that level. The parameter k is user specified. When k is equal to one this type of search is equivalent to a simple greedy search.

5 Branch and Bound

We developed a branch and bound algorithm which uses the same search tree as the beam search algorithm but finds a guaranteed optimal solution. The algorithm is essentially depth first search with pruning. An inequality is used to compute an upper bound on $E(V)$ for any V whose first i elements are determined by the path to the node at level i of the search tree. In order to describe the bound formally we need to introduce some more notation. Let

$$E(v_1, v_2, \ldots, v_i, v_{i+1} \rightsquigarrow v_n) = \max_{v_{i+1}, v_{i+2}, \ldots, v_n} E(v_1, v_2, \ldots, v_n),$$
$$E(v_{i+1} \rightsquigarrow v_n) = \max_{v_{i+1}, v_{i+2}, \ldots, v_n} E(v_{i+1}, v_{i+2}, \ldots, v_n).$$

In words, $E(v_1, v_2, \ldots, v_i, v_{i+1}, \rightsquigarrow v_n)$ is the optimal value of $E(V)$ for a set of window positions V which starts with a path to a particular node at level i; while $E(v_{i+1} \rightsquigarrow v_n)$ is the optimal value over V' of the evaluation function $E(V')$ where V' is a set of window positions in sequences $S_{i+1}, S_{i+2}, \ldots, S_n$. The upper bound we use can now be written as:

$$n * E(v_1, v_2, \ldots, v_i, v_{i+1} \rightsquigarrow v_n) \leq i * E(v_1, v_2, \ldots, v_i) + (n-i) * E(v_{i+1} \rightsquigarrow v_n). \quad (1)$$

Note that the second term of the right hand side can be obtained by solving a smaller version of the original problem. Utilizing that fact we have developed a recursive algorithm that uses two arrays (which are defined to be global variables) to hold bounds:
Array L is defined such that $L[i]$ will be assigned a lower bound for the maximum value of E for the sequences $S_i, S_{i+1}, \ldots, S_n$,
Array U is initiated to have undefined values for all its elements and will be assigned values during the execution of the algorithm such that $U[i]$ is either undefined or it contains the exact maximal value of E for the sequences $S_i, S_{i+1}, \ldots, S_n$. The level at which the algorithm begins pruning is denoted d, which is a user defined parameter. A description of the algorithm is given in table 1.

Pruning is generally ineffective at the first 2 levels of the tree, therefore we used $d = 3$ when the program was called with less than 8 sequences, i.e. when $n - start\_seq + 1 \leq 7$, and $d = 4$ otherwise. Note that the array L can be computed in one pass of the beam search heuristic by inputing the sequences in reverse order and letting $L[i]$ be the best candidate for the sequences $S_n, S_{n-1}, \ldots, S_i$.

6 Proof of the Bound

We first prove the inequality for $I(V)$ and then for $E(V)$. Let F, F_0, and F_1 be probability vectors with elements $F(b)$ specifying the probability of generating character b. Furthermore let the probability vectors satisfy the relation:

Description of Branch and Bound Algorithm

Use the beam search heuristic to calculate the values of an array L such that $L[i]$ holds a lower bound for the maximum value of E for the sequences $S_i, S_{i+1}, \ldots, S_n$.

Call Program(1).

Program($start\_seq$)

if $start\_seq \geq n - d$ then
    Calculate $U[start\_seq]$ with the exhaustive algorithm
    Return $U[start\_seq]$

for($i = n$ to $start\_seq + d$ step -1)
    if $U[i]$ is undefined $U[i] \longleftarrow$ Program(i)
endfor

$Max\_E \longleftarrow -\infty$

Visit all the nodes at level d of the search tree using the exhaustive algorithm described above. For each node, use $U[start\_seq + d]$, $L[start\_seq]$, the value of $E(v_1, v_2, \ldots, v_d)$ for the path $v_1, v_2, \ldots, v_d$ leading to the node, and inequality 1 to determine if the node can be eliminated.
    For each node generated that cannot be eliminated by inequality 1
        perform a depth first search,
            pruning any node generated that can be eliminated with the inequality.
        For each node generated at level n update Max_E if necessary.

Return Max_E

Table 1: The main function "Program" is recursively called with the argument $start\_seq$ specifying from which sequence "Program" is to begin. For example, if "Program" were called with $start\_seq = 5$, it would return an optimal solution for input consisting of the 5th through nth sequences.

$$\forall b \in A \; F(b) = \theta F_0(b) + (1-\theta)F_1(b), \; 0 \leq \theta \leq 1.$$

Define I analogously with $I(V)$ defined earlier, i.e.

$$I = \sum_{b \in A} F(b) * log_2(F(b)/p_b).$$

Where the p_b's are constants. Define I_0 and I_1 likewise using F_0 and F_1.

Theorem 1 $I \leq \theta I_0 + (1-\theta)I_1$.

The meat of the proof is provided by the following lemma:

Lemma 1

$$H \geq \theta H_0 + (1-\theta)H_1 \tag{2}$$

where:

$$\begin{aligned} H &= -\sum_{b \in A} F(b) * \; log_2 F(b), \\ H_0 &= -\sum_{b \in A} F_0(b) * log_2 F_0(b), \\ H_1 &= -\sum_{b \in A} F_1(b) * log_2 F_1(b). \end{aligned}$$

Proof of lemma 1:
The proof closely follows the proof of a similar theorem by Gallager [5]. Consider F, F_0, and F_1 to be probability vectors over the sample space B equal to the space of generating single characters from the alphabet A. Furthermore consider the probabilities $F_0(b)$ to be conditioned on a binary variable z with sample space $Z = \{0, 1\}$ such that

$$F_0(b) = F_{B|Z}(b|0), \; F_1(b) = F_{B|Z}(b|1).$$

Let $z = 0$ with probability θ. Then inequality 2 is equivalent to the following series of equations:

$$H(B) \geq \sum_b Prob[z=0] * F_0(b) * log_2 F_0(b) + \sum_b Prob[z=1] * F_1(b) * log_2 F_1(b)$$

$$H(B) \geq \sum_{b,z} Prob[b,z] * log_2(Prob[b|z])$$

$$H(B) \geq H(B|Z).$$

The steps follow from the identity $Prob[b,z] = Prob[z]Prob[b|z]$ and the standard information theory definitions of $H(B)$ and $H(B|Z)$. The last equation is a well know identity and this completes the proof of lemma 1.

The proof of theorem 1 follows from lemma 1 by noting that I can be written as: $I = -H - F \cdot C$, where C is a vector defined such that $C(i) = log_2(p_i)$. C is a constant vector with respect to F so we have:

$$F \cdot C = \theta F_0 \cdot C + (1-\theta) F_1 \cdot C,$$

$$I - \theta I_0 - (1-\theta) I_1 = \theta H_0 + (1-\theta) H_1 - H$$

By lemma 1 the right hand side of the last equation is less than zero, thus finishing the proof of theorem 1.

7 Precomputation and Heuristic Speedup

7.1 Precomputation

The function $I(V)$ requires the calculation of several logarithms. Without precomputation those calculations would become a bottleneck, thus we employed two kinds of precomputation. First, as mentioned in the section on the exhaustive algorithm, for each possible column of up to 7 characters we precompute the value of $I(V)$ for that column. For nucleic acid sequences this requires $4^7 + 4^6 + \cdots + 4 \approx 22K$ double precision numbers worth of memory. This kind of precomputation is less effective for the larger alphabet size required for amino acid sequences but precomputing the values for columns of up to 3 or 4 amino acids is possible.

For columns with too many characters to cover with the above form of precomputation it is still possible to avoid calculating logarithms during the main phase of the algorithm. The key observation is that the character frequencies for which logarithms need to be computed always equal the ratio of the number of times a character is observed, which must be a nonnegative integer no greater than the number of sequences in the subproblem, divided by the number of sequences in the subproblem, which can be no greater than n. Thus there are only $O(n^2)$ possible frequencies. To exploit this fact, we precompute a $n+1 \times n+1$ array M, where for $i \leq j$ the $M[i][j]$ entry holds the value of $\frac{i}{j} * log_2(\frac{i}{j})$. Although this array does not allow $I(V)$ to be calculated with a single array lookup, the space required for it is only quadratic in the number of sequences.

7.2 Heuristic Input Ordering Speedup

Recall that when descending the search tree, the condition for eliminating a node and its descendants is dependent on the value of E for the path to the node and on the value of E for the optimal set of window positions maximized independently over the remaining sequences. Thus more nodes can be eliminated early if the sequences with the lowest maximal value of E come last in

the input. To give a concrete example consider the case where there are 11 input sequences and one is trying to eliminate nodes at the fourth level of the tree. If the maximal value of E for the last 7 sequences of the input is low then the right side of inequality 1 will provide a lower upper bound on the value of E for descendents of the node in question.

We exploit this observation by ordering the sequences based on the results of running the heuristic beam search on the initial, arbitrary ordering of the input. The input sequences are sorted in descending order according to the value of $\sum_{i=0}^{w-1} log_2(F(i, B(i,j))/p_{B(i,j)})$, where $F(i, B(i,j))$ denotes the frequency in column i, in the window position vector chosen by the heuristic, of the character in the ith column of the window in the jth sequence. And $p_{B(i,j)}$ denotes the prior probability of that character. The logic behind this is that if you believe that there is one strong signal in the data and that the heuristic can find that signal, then the sequences where that signal is weakest should have a low optimal value of E and thus should come last.

8 Data Sets

8.1 LexA Data Set

For this study we used a set of 11 E.coli DNA sequences of length 200, each of which is known to contain at least one binding site for the protein LexA. We copied this data set directly from Hertz *et al* [6]. LexA binds in or near promoters and may bind to sites which occur on either the sense strand or the anti-sense strand of the DNA. Thus we needed to consider possible occurrences of the pattern on both the sequence and the reverse complement of the sequence. Instead of designing our program to consider the reverse complement of each input sequence, we simply concatenated the reverse complement of each input sequence onto itself to create 11 sequences of length 400 each. This added $w-1$ bogus window positions in the middle of each input string, but as these window positions never appeared in a solution the solutions obtained were valid. As in Hertz *et al* we used prior probabilities of 0.25 for each of the four DNA bases.

8.2 Artificial Data

Intuitively one would expect stronger motifs to be easier to find. In order to investigate the relationship between motif strength and the running time of our algorithm we generated sets of artificial sequences with artificial motifs of varying strength planted in them. We somewhat arbitrarily chose to generate sets of nine sequences of length 320.

The sequence generator receives four parameters: the length of the sequences to be generated l, the number of sequences to be generated n, the length of the motif w, and the minimum information content c of each column of the motif. The generator can be described with the following pseudo-code:

Generate n DNA sequences of length l using a uniform distribution over the four possible bases.

For $i = 1$ to w
    Do
        Generate a motif column i of length n using a uniform distribution over the four possible bases.
    Loop while the information content of motif $i <= c$.
EndFor

Generate a set of n random motif starting positions using the uniform distribution over all possible starting positions.

Replace the characters in the sequences whose positions correspond to motif columns with the characters from the motif columns generated above.

9 Results

9.1 LexA Dataset

We ran the branch and bound program on the LexA data set with three different window lengths. The patterns found by the branch and bound program with a window length of 24 are shown in table 2. The running time was quite slow, requiring 71 hours on a Hewlett Packard 9000/715 workstation. The program ran faster with window sizes of 20 and 22, requiring 20 and 28 hours respectively. However since the window size of 24 is the only one for which the optimal solution escaped Stormo and Hartzell's heuristic, we decided to show that result. Table 3 shows how many nodes survived at each level of the search tree. Even with the least favorable window size of 24, the bound does well in terms of the percent of nodes eliminated. Remembering that the fan out for this particular search tree is $400 - 24 + 1 = 377$ for each level of the tree we can see that the bound eliminates all but 6×10^{-4} percent of the nodes at level 4 and all but 7×10^{-5} percent of the nodes at level 5 of the search tree. We must mention that the sequence input ordering heuristic was used for the window size of 24.

Sequence	Pattern Found	
umu-operon	CTACTGTATATAAAAACAGTATAA	
cloacin-df13	ATACTGTGTATATATACAGTATTT	
recn	TTACTGTATATAAAACCAGTTTAT	
uvrb	ATACTGGATAAAAAAACAGTTCAT	(complementary strand)
reca	ATACTGTATGAGCATACAGTATAA	
sula	TTACTGTATGGATGTACAGTACAT	(complementary strand)
colicin-ib	TATATGGATACATATACAGTACTA	(complementary strand)
colicin-ia	CATATGGATACATATACAGTATTA	(complementary strand)
colicin-e1	ATGCTGTATATAAAACCAGTGGTT	
uvrd	AATCTGTATATATACCCAGCTTTT	
uvra	ATACTGTATATTCATTCAGGTCAA	

Table 2: The name of each sequence and the 24 base window found by the branch and bound algorithm are given. When applicable the occurrence of the window on the complementary strand is indicated.

Without reordering the sequences the program took so long to execute that it had to be terminated prematurely, however judging from the pace of its progress we estimate the reordering to have sped the calculation up by at least a factor of 10. The reordering heuristic was not necessary for window sizes of 20 and 22.

We did not run extensive empirical tests of the performance of our beam search heuristic with the heuristic of Stormo and Hartzell [1]. However the results from very limited data suggest that our heuristic performs somewhat better. Table 4 shows how their results compare with the beam search heuristic. We chose to keep the number of candidate paths, i.e. the beam width, equal to the length of the sequence, a parameter setting which should keep the amount of memory and time resources used by the two heuristics roughly equal (recall that Stormo and Hartzell's algorithm performs one greedy search for each window in the first sequence). Note that that although the difference is small our beam search always finds an equal or better solution. If one increases the number of candidates kept to 1700 from 400, then our beam search heuristic finds the optimal solution for a window size of 24 for all five orderings. The running time with the 1700 setting is just under 20 minutes.

Level of Tree	Number of Nodes that Survive
4	131232
5	499498
6	3489962
7	597786
8	133618
9	200389
10	27146

Table 3: Results for the branch and bound program on the LexA dataset with a window width of 24. The number of nodes surviving at each level of the search tree for which pruning is done is shown.

Search Type	Window Size	E(V) of Solution Found	Times found / # of trials
Beam Best		27.241	5/5
Stormo Best	20	27.241	1/5
Beam Best		28.426	5/5
Stormo Best	22	28.426	2/5
Beam Best		29.279	2/5
Beam Other		29.181	3/5
Stormo Best	24	28.926	2/5

Table 4: Beam Best labels the best solution found by our beam search heuristic while Beam Other labels a worse solution which was also found. A trial consists of randomly rearranging the order of the sequences and running the programs. The numbers for Stormo and Hartzell's heuristic were taken from their paper, while we randomly generated our own five orderings for our numbers.

9.2 Artificial Dataset

The running times with different window sizes on the LexA data set provides anecdotal evidence that a higher information content per position correlates with shorter running times. The results of a more systematic investigation of this correlation are shown in figure 1. Again the input ordering heuristic was necessary for the harder input sets, in this case the ones with motif strengths of 21.4 and 21.8 bits. It can be seen that the computation becomes infeasible as the information content drops below 21.4 bits for a motif length of 20.

To get a feel for the significance of this motif strength we tried to determine what information content could be expected when no motif was present. When run on sets of nine random sequences of length 320 (without any planted motifs), our beam search heuristic consistently found length 20 windows with an information content of approximately 18.8 bits. Thus for data sets of the size tested here, the possibility remains of subtle motifs appearing with strengths in the range of 18.8 to 21.4 bits over 20 columns which cannot be found with the current implementation of our branch and bound algorithm.

10 Discussion and Future Directions

Our main result was that we were able to solve a non-contrived data set of reasonable size to global optimality. The data set itself was used to demonstrate the heuristic of Stormo [6] and the size of the problem $O(400^{11})$ is comparable in size to the data sets used to test two other heuristic algorithms for maximizing functions very similar to $E(V)$. Specifically the data sets used to test an expectation maximization algorithm by Lawrence and Reilly [7] and an algorithm using Gibbs Sampling by Lawrence *et al* [8] were not much bigger than $O(400^{11})$. We must note here that in practice these heuristics produce good solutions and are more flexible in terms of modifying the objective function, not to mention that they run much faster than our branch and bound algorithm. Our point is that it is somewhat surprising that a problem as large as the one presented here could be solved to optimality.

Branch and Bound algorithms have been effective in optimally constructing global multiple alignments, where *global* refers to aligning the whole strings rather than local patterns in the strings, but for somewhat smaller problem sizes. For example, in regards to his branch and bound algorithm for the maximum weight trace formulation of multiple alignment Kececioglu [9] states "... we can solve instances on as many as 6 sequences of length 250 in a few minutes. These are among the largest instances that have been solved to optimality to

date for any formulation of multiple sequence alignment."

To increase the utility of the branch and bound algorithm three improvements need to be made. One needed improvement would be to speed up the algorithm so that data sets with 20 or more long (length >= 300) sequences could be handled instead of the current limit of about 10. To address this we have considered searching a search tree in which the nodes represent the displacement of windows relative to each other rather than the absolute positions of windows, as well as implementing the algorithm on a parallel machine. Unfortunately we have not significantly developed these ideas at this time. The second improvement would be to include "regularizers" to adjust for small sample effects in the evaluation function. A Laplacian regularizer can be used with the current implementation by simply adding dummy sequences of length w, one for each character in the alphabet consisting entirely of that character. However Karplus [10] has shown that, at least for protein sequences, the Laplacian regularizer performs poorly. He finds Dirichlet mixtures to be the best choice and we are therefore considering how to modify our program to accommodate a Dirichlet mixture regularizer. A compromise would be to use fractional pseudocounts, which Karplus found to be much better than a straight Laplacian regularizer. The third improvement would be to incorporate the so called "ZOOPS" model proposed by Bailey and Elkan [11]. This model allows for the possibility that some sequences do not contain the motif. It is an attractive model computationally because it adds only one bit per sequence to the search space. Moreover the model can be used iteratively to locate motifs that occur a different number of times on different sequences, without any prior knowledge of the number of times the motifs occur.

In conclusion, we have developed a branch and bound algorithm for a widely used formulation of multiple sequence alignment. This algorithm has allowed problems to be solved to optimality that were previously amenable only to heuristic algorithms.

References

[1] G.Stormo and G.W.Hartzell III, *Identifying protein-binding sites from unaligned DNA fragments*, Proc. Natl. Acad. Sci. USA, 86 (1989) pp. 1183-11187.

[2] T.Schneider, G.Stormo, L.Gold and A.Ehrenfeucht, *Information Content of Binding Sites on Nucleotide Sequences*, J. Mol. Biol., 188 (1986) pp. 415-431.

[3] Timothy L. Bailey, *Likelihood vs. Information in aligning biopolymer sequences*, Technical Report CS93-318, University of California, San Diego February 1993.

[4] D.J.Bacon and W.F.Anderson, *Multiple Sequence Alignment*, J. Mol. Biol., 191 (1986) pp. 153-161.

[5] Robert G. Gallager, *Information Theory and Reliable Communication*, John Wiley and Sons, 1968, pp. 90-91.

[6] G.Z.Hertz, G.W.Hartzell III, and G.D.Stormo, *Identification of consensus patterns in unaligned DNA sequences known to be functionally related*, CABIOS, Vol 6. #2. (1990) pp. 81-92.

[7] Charles E. Lawrence and Andrew A. Reilly, *An Expectation Maximization (EM) Algorithm for the Identification and Characterization of Common Sites in Unaligned Biopolymer Sequences*, PROTEINS, 7 (1990) pp. 41-51.

[8] C.E.Lawrence, S.F.Altschul, M.B.Boguski, J.S.Liu, A.F.Neuwald, and J.C.Wootton, *Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment*, Science, 262 (1993) pp. 208-214.

[9] John Kececioglu, *The maximum weight trace problem in multiple sequence alignment*, Proceedings of Combinatorial Pattern Matching, (1993) pp. 106-119.

[10] Kevin Karplus, *Evaluating Regularizers for Estimating Distributions of Amino Acids*, Proceedings of ISMB-95, (1995) pp. 188-196.

[11] Timothy L. Bailey and Charles Elkan, *The value of prior knowledge in discovering motifs with MEME*, Proceedings of ISMB-95, (1995) pp. 21-38.

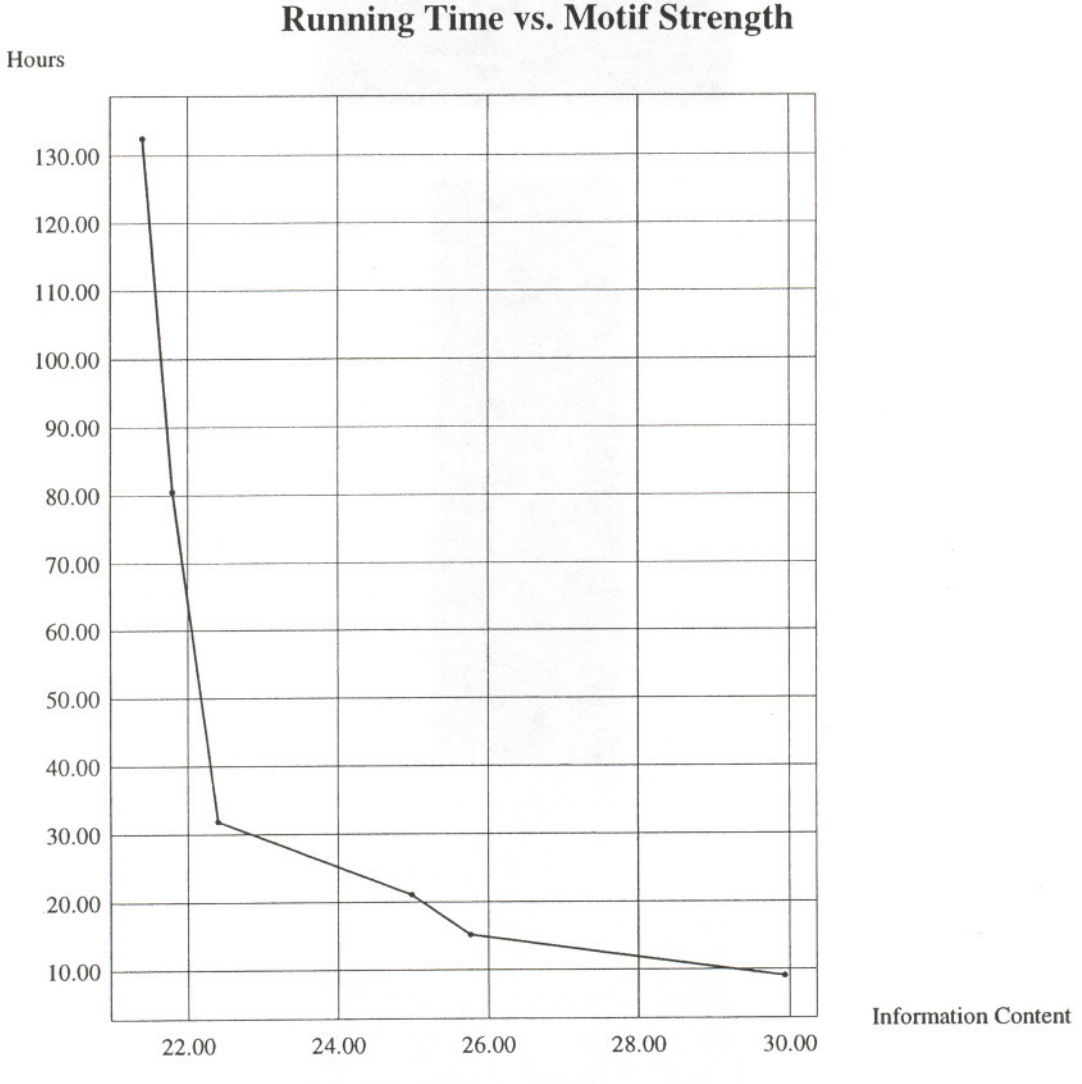

Figure 1: The y-axis gives the running time in hours of the branch and bound algorithm, while the x-axis gives the total information content of the optimal set of window positions. The data shown is from the artificial data set.