Extensions of Some Efficient Algorithms for Similarity Searches

Nimisha Gupta Priti Shankar

IISc-CSA-TR-2000-1

http://archive.csa.iisc.ernet.in/TR/2000/1/

Computer Science and Automation
Indian Institute of Science, India

January 2000

Extensions of Some Efficient Algorithms for Similarity Searches

Nimisha Gupta
Priti Shankar*
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560012
India

January 28, 2000

Abstract

Local similarity searches in biological databases and multiple sequence alignment are important problems in computational biology. In this paper, an algorithm called the stack algorithm, well known in the area of coding theory, has been proposed for use in the last phase of a local alignment algorithm like Gapped BLAST, to overcome the inefficiencies of an approach based on dynamic programming. The A^* algorithm, originally proposed for applications in artificial intelligence, was later adapted for multiple sequence alignment. We show that it is possible to modify this algorithm to incorporate natural gap costs, while retaining its computational efficiency. We also investigate the effect of using upper bounds in reducing the search space of this algorithm. Some results from the implementation of both these algorithms are reported.

1 Introduction

The role of comparative sequence analysis in the task of identifying and characterizing the genes of an organism is becoming increasingly important [24] [8]. Several software tools have been developed for searching biological databases. Some well known tools are SSEARCH [1], FASTA[27], BLAST [2], Gapped BLAST and PSI BLAST[3] and PSW [11]. Tools like SSEARCH and PSW find an optimal alignment and are based mostly on the dynamic programming algorithm[31] [34][18] [33] [37]. Frequently, to find pairwise alignments, costs must be charged for gaps, which are runs of nulls, as well as for aligning individual elements with nulls. Algorithms that employ constant gap costs are widely used [18, 7, 15, 30] SSEARCH is an implementation of the local alignment dynamic programming technique proposed by Smith and Waterman(1981). PSW is a modified version of the Smith-Waterman algorithm where the score computed for two sequences is a log likelihood ratio of two probabilities, one given the scoring system and one given a null model. FASTA, BLAST and Gapped BLAST are extremely fast heuristic tools that find an alignment with high probability. FASTA efficiently counts the number of pairs in each diagonal of the dynamic programming matrix and then examines these diagonal counts to infer similarity in the query and database. Even though FASTA is able to identify biologically significant sequences, it sometimes misses sequences with weak similarities. BLAST directly approximates alignments

*Corresponding author. E-mail:priti@csa.iisc.ernet.in

that optimize a measure of local similarity, the maximal segment pair score. Gapped BLAST uses dynamic programming to extend a central pair of aligned residues in both directions, and runs about three times as fast as the original algorithm.

The dynamic programming problem is best viewed as the problem of computing an optimal path through a graph. The graph can be considered as a grid of $M \times N$ nodes where M and N are the lengths of two sequences. All possible alignments of the prefix of length i of one sequence with that of length j of the second are represented by the set of paths from $(0,0)$ to (i,j). The dynamic programming algorithm when applied to sequence alignment with huge biological databases is somewhat expensive in time. Some algorithms [13] confine the dynamic programming algorithm to a banded section of the full graph. Others [38] consider only nodes for which the optimal local alignment score falls no more than some parameter below the best alignment score yet found. Algorithms such as BLAST and Gapped BLAST usually use a variation of the dynamic programming algorithm in the final phase of a multi-phase algorithm where a gapped alignment is sought.

In this paper we describe an algorithm, which is well known in the coding literature and is conceptually extremely simple. This is the *stack algorithm* , originally proposed independently in [26] and [39] for the sequential decoding of convolutional codes used for error correction in communication channels. The problem addressed by the original algorithm was to find a path that optimized a certain metric, through a regularly structured graph called a *trellis*. The key idea behind the original algorithm is that only one path is extended at a time, several others being stored as an ordered list of their metric (suitably chosen) in the form of a stack. The top node of the stack is the one with the greatest value of metric among all its nodes. The chosen path is extended branch by branch ensuring that at each step it does correspond to the path with greatest metric. As soon as the metric drops below some other path stored on the stack, this path is stacked and the one that looks most promising is extended. The path that is on top of stack when the end of the trellis is reached is declared a winner. The problem of local alignment requires a slightly different model from the one used for the communication channel. We have therefore adapted the stack algorithm to take into account differences in the model.

Multiply aligned sequences provide implicit protein structural information. A multiple alignment can discover relationships not evident from pairwise comparisons. Multiple sequence comparison has been shown to be NP-complete [17] implying that any algorithm will ultimately exhibit exponential behavior in d (number of sequences being aligned). The classical dynamic programming approach for the *optimal* multiple alignment problem reduces multiple alignment of d sequences of length n to the minimum path problem in the graph with n^d vertices representing a lattice in the d-dimensional space (e.g in [33]). Carrillo and Lipman(1988) made a significant advance when they presented an algorithm to bound the search space in a dynamic programming(DP) approach. Using their technique it became possible to align up to 6 sequences of length up to 200 in reasonable time. Spouge(1989) suggests the use of the A^* algorithm [22] used by the artificial intelligence community, and has also proposed the use of upper bounds in reducing the search space. Gupta et.al() have used branch and bound techniques for multiple sequence alignment Recent work of Ikeda and Imai(1999) presents some efficient approaches for optimal and suboptimal multiple sequence alignment using variations of shortest path algorithms. Altschul(1989) has argued that *natural gap costs* are the costs of choice in the sum of pairs cost approach to multiple alignments. While extending an alignment by an additional edge, the additional cost incurred is the sum of the substitution and gap costs. While the substitution cost depends only on the new edge, the gap cost may depend on more of the alignment's history. Altschul notes that the number of histories to be taken into account is too large to be able to find optimal alignments of more than five sequences in reasonable space or time. We have proposed and implemented a technique

that needs to maintain an array of size approximately d^2, with each node in order to be able to encode the relevant history. Entries in this array, are from a ternary alphabet. With this, we have been able to incorporate natural gap costs for the multiple sequence alignment problem, in the framework set up by Ikeda and Imai(1999), without significant additional overheads.

Section 2 describes our adaptation of the stack algorithm for local similarity searches and compares the results with those using Gapped BLAST. Section 3 describes the A^* algorithm and its implementation using natural gap costs. It also investigates the role of upper bounds in reducing the number of vertices searched and expanded. Finally section 4 presents our conclusions.

2 Adapting the Stack Algorithm for Local Alignments

2.1 Background

The problem for which the original stack algorithm was proposed as a solution, was to find a path from a source to a destination node of a regularly structured graph called a *trellis*, such that a metric associated with the path was maximized. The graph consisted of an $M \times N$ grid of nodes; one axis of the grid corresponded to states of a finite state machine defining a convolutional code, and the other to a received q-ary message of N symbols which was to be decoded into the most likely codeword transmitted. An exact solution to the problem was provided by the $Viterbi$ algorithm, which was essentially a dynamic programming solution to the problem. A good approximation to the solution was provided by the *stack algorithm* proposed independently by Jelinek(1969) and Zigangirov(1966), which computed an approximation to the optimal path using a metric called the *Fano* metric[14].

In the original stack algorithm an ordered list of nodes called a stack is maintained. Thus the top node is the one with greatest value of metric among the nodes currently on the stack. At the outset, the origin node is loaded on stack. Computation proceeds as follows :

1. Compute the metric for the successors of the node on top of the stack and insert them into the appropriate location on the stack, deleting the node whose successors were just added.
2. If the new top node corresponds to node (M, N) stop, else repeat step 1.

When the algorithm halts, the path associated with the node on the top of the stack is the path selected.

There are some differences between the problem for which the stack algorithm was proposed as solution and the problem we address here.

1. The grid used here has on each of its axes one of the pair of sequences for which the best local alignment is sought.
2. The original context considered only substitutions. Here we also allow insertions and deletions. Thus non null characters in one of the strings may be aligned with null characters in the other.
3. The length of the grid $M \times N$ was fixed for the decoding problem, in the sense that one sought a path of length N as the solution. Here we permit the algorithm to stop even if we do not reach the destination node, as partial alignments are acceptable.

4. In the original context only *codewords* were accepted as solutions. these were guaranteed to be at a minimum distance(in terms of the number of substitutions) from one another. Here all possible alignments are possible as solutions.

The original Zigangirov- Jelinek algorithm was impractical because of the overheads in maintaining the stack. Jelinek suggested an approximation which placed nodes in bins depending on their values. This can be considered as a quantized version of the original algorithm. For some $q > 0$, a node with the metric value $V(n)$ is placed in bin k if $kq \leq V(n) \leq (k+1)q$. The original algorithm is modified in the following way. One selects any node from the highest ordered bin as the node to be examined next. Further there is no need to search the stack when the nodes are added. An additional modification used in our adaptation of the algorithm has to do with selecting only one of the paths that merge at a node. This requires the use of an array which stores a pointer to the *best* path seen so far to a node examined. A high level description of the algorithm follows.

1. Compute the metric for the three successors of the node on top of the stack and insert them into the appropriate location on the stack, deleting the node whose successors were just added. Store the node corresponding to the best metric seen so far on top of stack.
2. If the new top node corresponds to node (M, N),continue with the next best path.
3. If the metric of the topmost node falls X (a parameter) below the best metric seen so far, stop.

Step 2 essentially says if node (M, N) is reached, we attempt to empty the stack, unless the condition in 3 is satisfied at some point, in which case we exit. The reason for this modification to the original algorithm will become clear later. The algorithm is formally described below.

Algorithm STACK
Input: A network $N = (G = (V, E), l)$ where V is the set of vertices, E the set of edges, and l is a function that maps each edge into a metric. s is the source vertex.
Output: A possibly suboptimal alignment P_{max}.
/* $p(u)$ is the metric or *score* associated with the path terminating at vertex u; $p_{max}(u)$ is the best score seen at u so far;$P(u)$ is the path to u with score $p_{max}(u)$; p_{max} is the best score seen on any path so far; P_{max} is the path corresponding to the best score seen so far. A path is stored as a sequence of edges $(e_1, e_2.....)$. Let () denote the empty sequence*/
begin
$S \leftarrow \{s\}; p(s) \leftarrow 0; p_{max}(s) \leftarrow 0; P(s) \leftarrow (); p_{max} \leftarrow 0; P_{max} \leftarrow ();$
repeat
Let u be the vertex in S which maximizes $p(u)$ with co-ordinates (i, j) .
$S \leftarrow S \backslash \{u\};$
if $p(u) < p_{max} - X$ **then return** P_{max}
else if $p(u) > p_{max}$ **then**
begin
$p_{max} \leftarrow p(u)$
$P_{max} \leftarrow P(u)$
end
if $i = M$ or $j = N$ **then**
begin

```
            if p(u) > p_max then
            begin
                p_max ← p(u)
                P_max ← P(u)
            end
            continue;
        end
    if p(u) < p_max(u) then continue;
    else
        begin
            p_max(u) ← p(u)
            for each (u, v) ∈ E  do
             p(v) ← p(u) + l(u, v);
             extend P(u) with edge (u, v) to give P(v)
             S ← S ∪ {v};
            endfor
        end
  until S ← ∅ ;
end
```

Initially the origin s is added to the stack S. At every step, a vertex u from the non-empty bin corresponding to the highest score is removed from the stack. If the score at u ($p(u)$) is X (a parameter) below the maximum score seen on stack (p_{max}) so far, then the algorithm stops and returns the alignment corresponding to best path seen on stack (P_{max}). If u corresponds to the end of database or query sequence then we update the information regarding the best path seen so far (if applicable) and continue with the next best entry on the stack. If $p(u)$ is less than the best score seen at u ($p_{max}(u)$) so far, we continue with the next best entry on the stack; otherwise the three successor nodes corresponding to the deletion, insertion and substitution edge are added to the stack in the appropriate bin depending on their scores.

2.2 Choosing a suitable metric

We now provide a brief justification for our choice of metric.

We first define a *partial edit sequence* as a sequence of the form $E_1E_2...E_i$, where E_j ,$1 \leq j \leq i$ is one of the following atomic operations:

I_a : insert symbol a into the subject string.

D_a : delete symbol a from the subject string.

S_{ab} : substitute symbol a in the subject string by b

All these operations are assumed to be performed in the database string (It does not matter whether the database or query is chosen as the subject string). Each edit sequence uniquely defines a partial alignment of possibly different length prefixes of the query and database sequences. For example, suppose the edit sequence is $I_aI_bS_{ab}S_{cd}D_eS_{aa}$ Then this edit sequence transforms the subject (database) string into the query sequence as shown below.

$$\begin{array}{lcccccc} \textit{Database} & : & - & - & a & c & e & a \\ \textit{Query} & : & a & b & b & d & - & a \end{array}$$

Each edit operation is assigned a score and our aim is to compare partial alignments (equivalently edit sequences), perhaps of different lengths.

Let $a_1....a_N$ be the query sequence, and let $E_1E_2.....E_s$ be a sequence of atomic edit operations that transforms the database sequence $b_1....b_l$ to a prefix $a_1....a_m$ of $a_1....a_N$. Our model views the evolutionary process as a communication channel through which the database sequence is transmitted. What is received at the output is the query sequence. The channel is liable to make insertion, deletion and substitution *errors* and each of these is associated with a probability. Each edit sequence thus corresponds to an *error pattern* . Given that the query sequence is received, we have to estimate the most likely error pattern. This most likely pattern then corresponds to the best or highest scoring alignment of the query sequence with the database sequence (The decision as to whether this query sequence is homologous to the database sequence is based on criteria extraneous to the algorithm itself).

Let N be the length of the query sequence and M that of the database. Since there is a unique correspondence between edit sequences and alignments, maximizing the probability of the former is equivalent to maximizing a certain *score* of the latter. In our context it is called a *metric* which we will now define. Let $P(E/(a,b))$ denote the probability of a (partial) alignment E given that a and b are the database and query sequences respectively. We have

$$P(E/(a,b)) = \frac{P(E,(a,b))}{P(a,b)} \tag{1}$$

where $P(E,(a,b))$ is the joint probability of edit sequence E and the (database, query) pair (a,b). The righthand side of the equation above can be rewritten as:

$$\frac{P(E,(a,b))}{P(a,b)} = \frac{P(E)P((a,b)/E)}{P(a,b)}$$

Assume that edit sequence E aligns $a_1...a_l$, $l \leq N$ with $b_1....b_m$, $m \leq M$.

$$\frac{P(E)P((a,b)/E)}{P(a,b)} = \frac{P(E)P(a_1....a_l, b_1....b_m/E)P(a_{l+1}...a_N, b_{m+1}...b_M)}{P(a,b)}$$

since the portions $a_{l+1}....a_N, b_{m+1}....b_M$ are independent of E. Also, assuming that the query and database sequence are independent,

$$P(a,b) = P(a)P(b)$$

and assuming that occurrence of amino acids in the sequence are independent,

$$P(a_1...a_N) = P(a_1)....P(a_N)$$
$$P(b_1....b_M) = P(b_1)...P(b_M)$$

The term $P(a_1....a_l, b_1....b_m/E) = 1$ as edit sequence E transforms $b_1....b_m$ to $a_1....a_l$. Making all these substitutions in equation 1 gives us

$$P(E/(a,b)) = \frac{P(E)}{(P(a_1)....P(a_l)P(b_1)...P(b_m))}$$

.

Let E consist of substitutions S_{ij}, insertions I_g (where an insertion of symbol g is made in the database) and deletions D_h (where the symbol h of the database is deleted)

$$P(E/(a,b)) = \prod_{a_i \to b_j \in substitutions} \frac{P(S_{ij})}{((P(a_i)P(b_j))} \prod_{g \in insertions} \frac{P(I)}{P(g)} \prod_{h \in deletions} \frac{P(D)}{P(h)}$$

where g is the symbol inserted in the database and h is the symbol deleted from database; $P(g)$ is the a priori probability of symbol g occurring in the query, and $P(h)$ the a priori probability of symbol h occurring in the database $P(I)(P(D))$ is $P(I_1)(P(D_1))$, the probability of *beginning* an insertion(deletion) gap if $g(h)$ is the first symbol inserted in a gap, else $P(I)(P(D))$ is $P(I_c)(P(D_c))$ if $g(h)$ is a *continuation* of an insertion(deletion) gap.

Taking logarithms on both sides

$$S(E/(a,b)) = \sum_{a_i \to b_j \in substitutions} log(\frac{P(S_{ij})}{((P(a_i)P(b_j))}) + \sum_{g \in insertions} log(\frac{P(I)}{P(g)}) + \sum_{h \in deletion} log(\frac{P(D)}{P(h)})$$

The score then can be considered to be the sum of the following quantities.

- For substitutions S_{ij} the score is just the log-odds score of Altschul et al.(1991).
- For insertions, the first insertion has a score of
  $log(\frac{P(I_1)}{P(g)})$ where $P(I_1)$ is the probability of beginning an insertion gap and $P(g)$ is the a priori probability of occurrence of g (the symbol in the query string that was *inserted* in the database string) in the query string. Successive insertions have a score of $log(\frac{P(I_c)}{P(g)})$ where $P(I_c)$ is the probability of extending an insertion gap.
- For deletions, the first deletion h has a score of $log(\frac{(P(D_1)}{P(h)})$ where $P(D_1)$ is the probability of beginning of a deletion gap and $P(h)$ is the apriori probability of the occurrence of h the symbol deleted in the database string.Successive deletions have a score of $log(\frac{P(D_c)}{P(h)})$ where $P(D_c)$ is the probability of continuation of a deletion gap.

We may thus associate with each edge in a path , its score corresponding to the metric defined above, the scores being additive along a path. The stack algorithm is based on the principle that along a path corresponding to a more probable alignment the metric would tend to increase with depth and decrease along a path corresponding to an improbable alignment. The following condition is satisfied by the original algorithm which declares the first path that reaches the end, a winner[28]:

Principle of Non-Selection

Assume that there is a pair of paths P and P' that diverge at depth j and let V_j = metric of a path P_1 from depth 0 to j. Let V_{j+1} , $V_{j+2}...V_{j+n}$ and V'_{j+1}, V'_{j+2} ... V'_{j+n} be the sequence of values along the paths P and P'. Then if $min(V_{j+1}, V_{j+2} \ldots V_{j+n}) > min(V'_{j+1}, V'_{j+2} \ldots V'_{j+n})$ then P'_1 cannot be on the top of stack when the algorithm stops.

The principle above suggests that if a path accumulates a very low (negative) value of metric initially, then even if it would have improved substantially later on if extended, it may not be on stack when the (original) algorithm ends. Since such a situation may arise frequently in our context, we do not stop with the first path that is on the top of the stack when we reach the end but explore other paths as well. Some of these may be removed from consideration as their score drops rapidly. Others may reach the end with a larger score than the first one that did.

2.3 Implementation

In order to incorporate the stack algorithm into a similarity searching algorithm, we decided to invoke the algorithm only when there was a reasonable chance of obtaining a good alignment. This is because the algorithm is known to perform badly on a noisy channel, in the sense that the number of paths on stack grows very rapidly. Therefore the first few phases of the algorithm were chosen to be identical to those of Gapped BLAST namely,

1. Look for two word pairs of length 3(called hits) with score at least T(a parameter) on the same diagonal, within length A(a parameter) from each other .
2. Perform an ungapped extension on both sides of the second hit. If the ungapped extension yields a score above a certain threshold, then perform phase 3.

For the last phase (phase 3), the *stack algorithm* was used in both directions beginning at the middle residue of the extended sequence obtained in the second phase.

In order to compare the time efficiency of the algorithm, the dynamic programming algorithm with a cut-off parameter was programmed . This method expands a node only if the score at that node does not drop X(a parameter) below the maximum score seen so far. This significantly reduces the search space of dynamic programming.

2.4 Results

We examined both, the sensitivity as well as the efficiency of the procedure that used the stack algorithm in the last phase. For sensitivity we checked out a set of query sequences that were processed by Gapped-BLAST, as well as by the algorithm using the stack algorithm in its final phase. The stack algorithm obtained practically all the alignments reported by Gapped-BLAST (except for an occasional miss near the cut off score). The total time taken employing the stack algorithm for the last phase was checked versus the time taken using the modified dynamic programming algorithm for the last phase. Table 1 presents the results for the two approaches on a UltraSparc 10 machine with a 300MHz clock cycle and a 128Kb RAM.

Speedups ranged from a factor of 4 to a factor of 20, the improvement being marked for longer query sequences. We have not examined the question of how the algorithm performs in comparison with the improved algorithm described in Zhang et al(1998). A set of sequences reported by Gapped-BLAST and the stack algorithm for a particular query are presented below. The queries processed by Gapped-BLAST were submitted to the www BLAST server[16]; those processed by the Stack algorithm and the modified DP algorithm were processed locally after downloading the database. The BLOSUM62[23] matrix was used for substitution costs; for gaps,

		Stack		Modified DP	
Seq No	Seq Len	User Time(s)	Sys Time(s)	User Time(s)	Sys Time(s)
1	468	157.3	0.8	3140.7	50.4
2	195	40.6	0.3	362.6	1.7
3	245	126.7	0.5	834.0	2.0
4	146	90.0	0.5	357.6	0.8
5	300	61.8	1.1	1314.1	12.5
6	159	33.2	0.5	498	3.7

Table 1: Execution times using the stack and the modified dynamic programming algorithm in the last phase

a cost of 10 was charged for beginning a gap and a cost of 1 for continuing the gap.

A sample of a local alignment output for Stack and BLAST

```
Sequences reported by both the BLAST and Stack algorithms.

Score 62
Query      RLDPSISTIMEKIGRAESCMSALYPNRQRGRRPAGAADKQRLPLEPAGARPRALLGKYVR
           R DP+  T      RA         +R+    PAG AD+ R P+ P G R R
Database   RADPTSGTTRRSARRATCARRPRSSSRRT--EPAGCADRHRCPVRP-GPRTRG-------
Query      KAAPLPDDAAQARQERHDT
            A P P +  + +  R D+
Database   PADPPPANTDRRQNHRRDS

Score 62
Query      AAQARQERHDTIKAHMDSLSPADRLRETVEDRTRAEQ
           A    +ER + I+A +DSL  +D L+E +ED    EQ
Database   ARSFNRERPERIEATLDSLGISDELKEKLEDVLIPEQ

Score 62
Query      PAGAADKQRLPLEPAGARPRALLGKYVRKAAPLPDDAAQARQERHDTIKAHMDSLSPA
           P G AD  R PL P G R   L G ++R   P  D    +   R     AH    +PA
Database   PDGIADHDR-PLAPRGIREAGLAGGWLRANLPAVDAVLCSTATRARQTLAHTGIDAPA

Score 62
Query      AAQARQERHDTIKAHMDSLSPADRLRETVEDRTRAEQ
           A    +ER + I+A +DSL  +D L+E +ED    EQ
Database   ARSFNRERPERIEATLDSLGISDELKEKLEDVLIPEQ

Score 62
Query      AAQARQERHDTIKAHMDSLSPADRLRETVEDRTRAEQ
           A    +ER + I+A +DSL  +D L+E +ED    EQ
Database   ARSFNRERPERIEATLDSLGISDELKEKLEDVLIPEQ

Score 65
Query      KIGRAESCMSALYPNRQRGRRPAGAADKQRLPLEPAGARPRALLGKYVRKAAPLPDDAAQ
           K+ RA + ++A       G     A + ++L   P G + + +     RKA  +    A+
Database   KLKRAGNSLAASTAEETAGSAQGRAQEAEQLLRGPLGDQYQTVKALAERKAQGVLAAQAR
Query      ARQ---ERHDTIKAHMDSLSPADRLRETVEDRTRA
           A Q   E  D ++A  D L     L  T E+  RA
Database   AEQLRDEARDLLQAAQDKLQRLQELEGTYEENERA
```

```
Score 66
Query       ELAMKQPGKRTGPLGHVALEVLDYLTNLVDFGNG
            ELA K   +   P  H A E +DYL ++ D G+G
Database    ELAEKMGARVYDPTAHAAKESIDYLRSIADGGDG

Score 1016
Query       MFQQIGAVQAKSGTDEPAHPCEKFPPERKCEAVFWKPLPRHEAREILLAARKYELAMKQP
            MFQQIGAVQAKSGTDEPAHPCEKFPPERKCEAVFWKPLPRHEAREILLAARKYELAMKQP
Database    MFQQIGAVQAKSGTDEPAHPCEKFPPERKCEAVFWKPLPRHEAREILLAARKYELAMKQP
Query       GKRTGPLGHVALEVLDYLTNLVDFGNGRLDPSISTIMEKIGRAESCMSALYPNRQRGRRP
            GKRTGPLGHVALEVLDYLTNLVDFGNGRLDPSISTIMEKIGRAESCMSALYPNRQRGRRP
Database    GKRTGPLGHVALEVLDYLTNLVDFGNGRLDPSISTIMEKIGRAESCMSALYPNRQRGRRP
Query       AGAADKQRLPLEPAGARPRALLGKYVRKAAPLPDDAAQARQERHDTIKAHMDSLSPADRL
            AGAADKQRLPLEPAGARPRALLGKYVRKAAPLPDDAAQARQERHDTIKAHMDSLSPADRL
Database    AGAADKQRLPLEPAGARPRALLGKYVRKAAPLPDDAAQARQERHDTIKAHMDSLSPADRL
Query       RETVEDRTRAEQLAG
            RETVEDRTRAEQLAG
Database    RETVEDRTRAEQLAG


Sequence reported by BLAST but missed by the stack algorithm :
Score :63
            RRPAGAADKQRLPLEPAGARPRALLGKYVRK------AAPLPDDAAQAR--------QER
            R+P G  DK+    EP+ A  R  +G+  RK      AA LP+ A  ++        +
            RKPGGDGDKKDRKFEPSAAPSR--VGRKQRKQKGPEAAARLPNVAPLSKCRLRLLKLERV

            HDTIKAHMDSLSPADRLR----ETVEDRTRAEQLAG
             D +    + ++  +RLR    +T EDR++ + L G
            KDYLLMEEEFVAAQERLRPTEDKTEEDRSKVDDLRG

The alignment reported by the Stack algorithm for this sequence :
Score :58
            RRPAGAADKQRLPLEPAGARPRALLGKYVRK----AAPLPDDA----AQARQERHDTIKA
            R+P G  DK+    EP+ A  R    +  +K    AA LP+ A     + R  + + +K
            RKPGGDGDKKDRKFEPSAAPSRVGRKQRKQKGPEAAARLPNVAPLSKCRLRLLKLERVKD
            HM----DSLSPADRLR----ETVEDRTRAEQLAG
            ++    + ++  +RLR    +T EDR++ + L G
            YLLMEEEFVAAQERLRPTEDKTEEDRSKVDDLRG
```

The stack algorithm obtained all the matches reported by Gapped-BLAST except for one. The reason for the stack missing an occasional sequence near the cutoff score is that an optimal path which originally accumulates a low metric is overtaken by a suboptimal path which has an initial good score; subsequently, even if the former would have accumulated a higher score if extended, it might not be considered as it might not be unstacked at all.

3 Multiple Sequence Alignment With Natural Gap Costs

The problem of multiple sequence alignment is to find the highest scoring alignment of multiple sequences given a scoring criterion. An issue that arises for multiple sequence alignment concerns the extension of costs from pairwise alignment to multiple sequence alignments. The simplest method is to define the cost of a multiple alignment to be the sum of the costs of all the implied pairwise alignments[29][10]. Another cost that has been proposed is obtained from treating the

input sequences as leaves of an evolutionary tree and reconstructing ancestral sequences that minimize the cost of the tree[33][32].

The dynamic programming approach to multiple sequence alignment is expensive, as the search space grows exponentially with d, the number of sequences to be aligned. Carrillo and Lipman(1988) made a significant advance in the reduction of the search space, which made it possible to align up to six sequences the length of an average protein.Their method makes explicit use of the definition of multiple alignment cost as the sum of the costs of the implied pairwise alignments. Altschul and Lipman(1989) extended the Carrillo-Lipman algorithm to multiple alignments whose cost is defined as that of an evolutionary tree. The A^* algorithm [22], which is a well known heuristic search method in artificial intelligence, finds the optimal alignment by searching a greatly reduced number of vertices. The basis behind the algorithm has been used in the multiple sequence alignment problem to bound the search space of DP[36]. Ikeda and Imai(1999) use the A^* algorithm to find the shortest path on the graph for multiple sequence alignment. They perform what is known as an edge length transformation on the original graph in order to obtain a new network on which a shortest path algorithm can be used. They have shown that up to seven protein sequences of length of around 400 amino acids can be optimally multiply aligned in reasonable time. The gap costs used by them are such that a null character $-$ in each sequence is assigned a fixed cost. (Note that a column of all null characters is not permitted in an alignment.)

Altschul(1989) has argued that natural gap costs are the costs of choice when considering multiple sequence alignment with sum of pairs costs. Gap costs are assessed separately from and in addition to *null costs*, which are the substitution costs for aligning individual letters with nulls. The *cost* of a pairwise alignment is then just the sum of the cost of all aligned pairs of the elements and cost of all gaps.

Among the possible gap costs, the simplest are *constant gap costs* which charge a fixed amount for each gap. In the absence of gap costs, the cost of an SP-alignment is the sum of costs of all its pairwise projections. Suppose that for pairwise alignments, gaps have a constant cost v. Then the number of gaps in a multiple alignment can be defined as the sum of the number of gaps in all its pairwise projections. If a cost v is charged for each of these costs, the corresponding gap cost is known as *natural gap costs* for SP-alignments.Table 2 shows the calculation of natural gap cost.Here, an amino acid has been represented by X and a gap by $-$ (null). The gaps in each pair of sequences have been counted and summed up to give the total gap penalty. Nulls in both the sequences of a pair are ignored.

For the first column of the alignment, no gap is accounted. In the second column a gap penalty is counted for the pair 12, 14, 23, 34. Note that gap penalty is not counted for the pair 13 as both the sequences have a gap at this position. The third column starts a new gap in pair 24 and incurs a gap penalty. Extensions of gaps in pairs 12, 14, 23, 34 do not add a gap. Altschul suggests natural gap costs as the obvious companion to SP substitution cost. We will show below, how natural gap costs can be incorporated into the A^* algorithm. Below is a brief description of the A^* algorithm.

3.1 The A^* Algorithm

Since the multiple sequence alignment problem can be formulated as a shortest path problem, we look for efficient solutions to the problem of finding the shortest path from the source vertex to the terminal vertex of a graph. We use notation from Ikeda and Imai(1999) to formulate the problem.

X	$-$	$-$	$-$	X	$-$	X
X	X	X	$-$	X	X	X
X	$-$	$-$	$-$	X	X	X
X	X	$-$	X	$-$	X	X

Pairs (ij)	12	13	14	23	24	34	Total
Column 1	0	0	0	0	0	0	0
Column 2	1	0	1	1	0	1	4
Column 3	0	0	0	0	1	0	1
Column 4	0	0	0	0	1	0	1
Column 5	0	0	1	0	1	1	3
Column 6	1	1	1	0	0	0	3
Column 7	0	0	0	0	0	0	0
Gap Penalty	2	1	3	1	3	2	12

Table 2: Natural gap penalty calculation

A *network* $N = (G = (V, E), l)$ is a directed graph $G = (V, E)$ where $l(u, v)$ provides the length of the edge (u, v). A *path* from u to v in a network N is an ordered set of edges $((v_1, v_2), (v_2, v_3)....(v_{n-1}, v_n))$ such that $u = v_1$ and $v = v_n$. The length of this path is given by

$$\sum_{i=1}^{n-1} l(v_i, v_{i+1})$$

The *shortest path* from s to t in a network N is the path from s to t which has the minimum length among all paths from s to t in N.

If the a vertex is reachable from s along a path with no negative length cycle, then the *shortest path* between two vertices exist. If source and destination vertices are represented as s and t respectively then the *shortest path problem* is the problem to find a shortest path from s to t. We denote by $L_N^P(u, v)$ the length of a part of a path from u to v in a network N and by $L_N^*(u, v)$ the shortest path length from u to v in N.

The A^* algorithm explores the graph G by partitioning the vertices encountered into two groups, *open*, and *closed*. At a given step in the algorithm, the vertex u under consideration is *closed* and assigned the minimum path weight $L_N^*(s, u)$ found up to that point. The algorithm then *expands* u by opening all successors v of u. If v, a successor of u is not already open, it is assigned the path weight $L_N^*(s, u) + l(u, v)$; if v was previously opened then it is assigned the minimum of the previous weight and the sum above. The algorithm chooses an order of expansion which guarantees that as u is closed, its assigned path weight is optimal. Let $h(u, t)$ be any lower bound such that $h(u, t) \leq L_N^*(u, t)$ and such that $h(u, t)$ satisfies the triangle inequality, i.e. for u a predecessor of v, $h(u, v)+h(v, t) \geq h(u, t)$. We term $h(.)$ an estimator. The algorithm is described below.

Algorithm A^*
Input: A network $N = (G = (V, E), l)$ where $l(u, v) \geq 0$ for any edge (u, v) in E, a source vertex s, a destination vertex t.
Output: The shortest path from s to t.

begin
$S \leftarrow \emptyset; \bar{S} \leftarrow \{s\}; p(s) \leftarrow 0; P(s) \leftarrow$the empty path
repeat
Let u be the vertex in $\bar{S}$ with minimum value of $p(u) + h(u,t)$
$S \leftarrow S \cup \{u\}; \bar{S} \leftarrow \bar{S} \backslash \{u\}$;
if $u = t$ **then return** $P(t)$;
for each $(u,v) \in E$ **do**
if $v \notin S$ **then**
begin
$p(v) \leftarrow min(p(u) + l(u,v), previous(p(v)))$;
append (u,v) to $P(u)$ to give $P(v)$
$\bar{S} \leftarrow \bar{S} \cup \{v\}$;
end
forever;
end

A careful inspection of the algorithm reveals that the dominant factor that determines the space complexity is the management of the open and closed lists $\bar{S}$ and S respectively; the time complexity is decided by how quickly the vertex with minimum value of $p(u)+h(u,t)$ is found and the speed of set operations. We note that once a vertex is closed, the shortest path is finalized. Ikeda and Imai have shown that if $h(.)$ satisfies the triangle inequality, and if the following edge weight transformation is performed

$$l_h(u,v) = l(u,v) + h(v) - h(u)$$

then one can obtain the shortest path in the network by using the well known Dijkstra shortest path algorithm on the network N_h where each edge weight $l(u,v)$ is replaced by $l_h(u,v)$. The triangle inequality property of $h(.)$ implies that l_h is always positive even if the original network had edges with negative costs. The Dijkstra algorithm is similar to the A^* algorithm described above, except that it selects a vertex with minimum value of $p(u)$ (i.e. it does not add an estimator).

3.2 Incorporating Upper Bounds into the A^* Algorithm

Spouge(1989) has noted that if $L_N^+(s,t)$ is an *upper* bound on the length of a shortest path from s to t, then any vertex v such that

$$L_N^*(s,v) + h(v,t) > L_N^+(s,t)$$

need not be expanded. This fact has been used to bound the search space of DP. Spouge has suggested omitting the management of the lists S and $\bar{S}$ and expansion of vertices in topological order. To achieve significant reduction in the search space the upper bound should be tight, and obtaining this is expensive. For the lower bound Ikeda and Imai have used the same estimator proposed by Spouge , namely $h(v,t) = \sum_{1 \le i,j \le d} L_{N_{ij}}^*(v_{ij}, t_{ij})$ where each term in the sum is the shortest path in a pairwise alignment, and the estimator is the sum of the best pairwise alignment over all pairs.

It should be clear that if in addition to maintaining lists, upper bounds are employed to limit the number of vertices expanded and searched, then the algorithm should perform fewer expansions and searches. We have run the A^* algorithm without incorporating upper bounds,

and also with tight and approximate upper bounds to check the effects on the number of vertices processed. We have used the same data as Ikeda and Imai(1999). The sequences used are the first $d, 3 \leq d \leq 7$ sequences of eukaryotic and archaebacterial proteins in Table 3. Table 4 represents the results of our implementation of the Ikeda-Imai algorithm . Table 5 represents the reduction in searches and expansions using a tight upper bound, namely the shortest path itself. Table 6 represents the reduction using Gusfield's approximation[21] as the upper bound. The results indicate that unless the upper bound is really tight, the search space is not significantly reduced. The number of *expanded* vertices is the number of vertices in the set S and the number of *searched* vertices is the number in the union of the sets S and $\overline{S}$.

Species	Protein	Length
Haloarcula Marismortui	EF-1α	421
Methanococcus Vannielii	EF-1α	428
Thermoplasma Acidophilum	EF-1α	424
Thermococcus Celer	EF-1α	428
Sulfolobus Acidocaldarius	EF-1α	435
Entamoeba Histolytica	EF-1α	430
Plasmodium Falciparum	EF-1α	443

Table 3: Proteins used in alignment

d	3	4	5	6	7
Score	3974	7717	12328	18120	24941
Vertices Expanded	462	943	2515	5237	39504
Vertices Searched	3073	9067	30552	95838	710478
Time(s)	0.4	0.7	1.7	20.1	3890

Table 4: Results for the direct application of A* algorithm

3.3 Incorporating Natural Gap Costs into the A^* Algorithm

The definition of natural gap costs[5] stipulates the addition of gap penalties for each gap in every pair of sequences. The problem is the following: Assume that the algorithm is currently tracking

d	3	4	5	6	7
Expanded	462	943	2515	5237	39504
Searched	465	985	2577	5390	40404
Time(s)	0.1	0.3	0.8	5.0	762.2

Table 5: Results for alignment of sequences using the A* algorithm and a tight upper bound

d	3	4	5	6
Expanded	462	943	2515	5237
Searched	3058	9044	28576	85257
Time(s)	0.1	0.3	1.5	21.9

Table 6: Results for the alignment of sequences using the A* algorithm and the Gusfield upper bound

(e_i, e_j) $h_{ij})$	(0,0)	(0,1)	(1,0)	(1,1)
$(X, X) = 0$	0(0)	1(+1)	1(+1)	0(0)
$(X, -) = -1$	-1(0)	1(+1)	-1(0)	0(0)
$(-, X) = 1$	1(0)	1(0)	1(+1)	0(0)

Table 7: Table for computing elements of new history array and additional gap costs

a path corresponding to a prefix of a multiple alignment ending at vertex v. This alignment will, in general, have gaps terminating a subset of the sequences. To compute the edge costs of the $2^d - 1$ edges leaving the vertex v, one needs to know, for each pair of sequences, whether or not a gap penalty needs to be charged. To assess this, it is enough to have an array of size $d(d-1)/2$ containing an entry for each pair of distinct sequences associated with each vertex in the list. The array $H[1..D]$ where $D = d(d-1)/2$ essentially stores the *relevant history* for computing natural gap costs. What needs to be recorded for a pair of sequences (i, j), is whether both relevant histories were substitutions,(i.e. (X, X) in the alignment, where X is a non null symbol), whether sequence i alone had a gap, ($(-, X)$ in the alignment), or whether sequence j alone had a gap ($(X, -)$ in the alignment). If both sequences had gaps, then the relevent history is the history of the previous column.

Given an element h_{ij} of the history array H, and an edge $(x_1, x_2, \ldots x_d), (x_1 + e_1, x_2 + e_2, \ldots x_n + e_n)$, where e_l is 0 or 1, $1 \leq l \leq d$, the contribution by sequence pair (i, j) to the gap cost incurred by the edge is decided by both h_{ij} as well as $(e_i, e_j), 1 \leq i, j \leq d, i \neq j$. h_{ij} can be encoded by the ternary alphabet $\{0, -1, 1\}$ to cover the three possibilities defined above; (e_i, e_j) can be $(1, 1)$ (substitutions in both sequences), $(0, 1)$ (gap only in sequence i), $(1, 0)$(gap only in sequence j), or $(0, 0)$(gap in both sequences). For the last case, the previous history in inherited, and no additional gap cost is incurred. The table below, dictates the computation of the new relevant history array, and the additional gap cost incurred by this edge. Each entry in the table is of the form $h_{ij}(c_{ij})$, where h_{ij} is the ij^{th} element of the new history array and c_{ij} is the additional gap cost incurred. The total gap cost incurred by the edge is then the sum of all the additional costs c_{ij}.

3.4 Results

We have implemented the A^* algorithm incorporating natural gap costs. We have used the same data as Ikeda and Imai(1999). These are the seven protein sequences in Table 7. The PAM-250 matrix has been used for substitution costs. For gap scores, a uniform score of -8 per null has

been used for the first alignment. For the second alignment using natural gap costs, a score of -5 for each gap has been used. The overheads are not significant and the execution time for the algorithm is approximately doubled. Below, we display the optimal alignments for the set of seven sequences considered by Ikeda and Imai. As expected, the alignment using constant gap costs displays fewer fragmented gaps.

Alignment for seven EF-1α protein sequences using constant gap costs

```
MS-DEQHQNLAIIGHVDHGKSTLVGRLLYETGSVPEHVIEQHKEEAEEKGKGGFEFAYVMDNLAEERERG
MAKTKPILNVAFIGHVDAGKSTTVGRLLLDGGAIDPQLIVRLRKEAEEKGKAGFEFAYVMDGLKEERERG
MASQKPHLNLITIGHVDHGKSTLVGRLLYEHGEIPAHIIEEYRKEAEQKGKATFEFAWVMDRFKEERERG
MAKEKPHINIVFIGHVDHGKSTTIGRLLFDTANIPENIIKKFE-EMGEKGK-SFKFAWVMDRLKEERERG
MS-QKPHLNLIVIGHVDHGKSTLIGRLLMDRGFIDEKTVKEAEEAAKKLGKDSEKYAFLMDRLKEERERG
MPKEKTHINIVVIGHVDSGKSTTTGHLIYKCGGIDQRTIEKFEKESAEMGKGSFKYAWVLDNLKAERERG
MGKEKTHINLVVIGHVDSGKSTTTGHIIYKLGGIDRRTIEKFEKESAEMGKGSFKYAWVLDKLKAERERG

VTIDIAHQEFSTDTYDFTIVDCPGHRDFVKNMITGASQADNAVLVVAA-D---D-GV-QP-QTQEHVFLA
VTIDVAHKKFPTAKYEVTIVDCPGHRDFIKNMITGASQADAAVLVVNVDDA--KSGI-QP-QTREHVFLI
VTIDLAHRKFETDKYYFTLIDAPGHRDFVKNMITGTSQADAAILVISARDG--E-GV-ME-QTREHAFLA
ITIDVAHTKFETPHRYITIIDAPGHRDFVKNMITGASQADAAVLVVAV-T---D-GV-MP-QTKEHAFLA
VTINLSFMRFETRKYFFTVIDAPGHRDFVKNMITGASQADAAILVVSAKKGEYEAGMSAEGQTREHIILS
ITIDISLWKFETSKYYFTIIDAPGHRDFIKNMITGTSQADVAILIVAAGTGEFEAGISKNGQTREHILLS
ITIDIALWKFETPRYFFTVIDAPGHKDFIKNMITGTSQADVALLVVPADVGGFDGAFSKEGQTKEHVLLA



RTLGIGELIVAVNKMDLVD--YGESEYKQVVEEV-KDLLTQVRFDSENAKFIPVSAFEGDNIAEESEHTG
RTLGVRQLAVAVNKMDTVN--FSEADYNELKKMIGDQLLKMIGFNPEQINFVPVASLHGDNVFKKSERNP
RTLGVPQMVVAINKMDATSPPYSEKRYNEVKADA-EKLLRSIGFK-D-ISFVPISGYKGDNVTKPSPNMP
RTLGINNILVAVNKMDMVN--YDEKKFKAVAEQV-KKLLMMLGYK-N-FPIIPISAWEGDNVVKKSDKMP
KTMGINQVIVAINKMDLADTPYDEKRFKEIVDTV-SKFMKSFGFDMNKVKFVPVVAPDGDNVTHKSTKMP
YTLGVKQMIVGVNKMDAIQ--YKQERYEEIKKEI-SAFLKKTGYNPDKIPFVPISGFQGDNMIEPSTNMP
FTLGVKQIVVGVNKMDTVK--YSEDRYEEIKKEV-KDYLKKVGYQADKVDFIPISGFEGDNLIEKSDKTP

WYDGEILLEALNELPAPEPPTDAPLRLPIQDVYTISGIGTVPVGRVETGILNTGDNVSFQPSD-V----S
WYKGPTIAEVIDGFQPPEKPTNLPLRLPIQDVYTITGVGTVPVGRVETGIIKPGDKVVFEPAG-A----I
WYKGPTLLQALDAFKVPEKPINKPLRIPVEDVYSITGIGTVPVGRVETGVLKPGDKVIFLPAD-K----Q
WYNGPTLIEALDQMPEPPKPTDKPLRIPIQDVYSIKGVGTVPVGRVETGVLRVGDVVIFEPASTIFHKPI
WYNGPTLEELLDQLEIPPKPVDKPLRIPIQEVYSISGVGVVPVGRIESGVLKVGDKIVFMPVG-K----I
WYKGPTLIGALDSVTPPERPVDKPLRLPLQDVYKISGIGTVPVGRVETGILKPGTIVQFAPSG-V----S
WYKGRTLIEALDTMQPPKRPYDKPLRIPLQGVYKIGGIGTVPVGRVETGILKAGMVLNFAPSA-V----V

-GEVKTVEMHHEEVPKAEPGDNVGFNVRGVGKDDIRRGDVCGPADDPPSVA--ET-FQAQIVVMQHPSVI
-GEIKTVEMHHEQLPSAEPGDNIGFNVRGVGKKDIKRGDVLGHTTNPPTVA--TD-FTAQIVVLQHPSVL
-GDVKSIEMHHEPLQQAEPGDNIGFNVRGIAKNDIKRGDVCGHLDTPPTVV--KA-FTAQIIVLNHPSVI
QGEVKSIEMHHEPMQEALPGDNIGFNVRGVGKNDIKRGDVAGHTNNPPTVVRPKDTFKAQIIVLNHPTAI
-GEVRSIETHHTKIDKAEPGDNIGFNVRGVEKKDVKRGDVAGSVQNPPTVA--DE-FTAQVIVIWHPTAV
-SECKSIEMHHTALAQAIPGDNVGFNVRNLTVKDIKRGNVASDAKNQPAVG-CED-FTAQVIVMNHPGQI
-SECKSVEMHKEVLEEARPGDNIGFNVKNVSVKEIKRGYVASDTKNEPAKG-CSK-FTAQVIILNHPGEI
```

```
TEGYTPVFHAHTAQVACTVESIDKKIDPSSGEVAEE-NPDFIQNGDAAVVTVRPQKPLSIEPSSEIPELG
TDGYTPVFHTHTAQIACTFAEIQKKLNPATGEVLEE-NPDFLKAGDAAIVKLIPTKPMVIESVKEIPQLG
APGYKPVFHVHTAQVACRIDEIVKTLNPKDGTTLKE-KPDFIKNGDVAIVKVIPDKPLVIEKVSEIPQLG
TVGYTPVLHAHTLQVAVRFEQLLAKLDPRTGNIVEE-NPQFIKTGDSAIVVLRPTKPMVIEPVKEIPQMG
GVGYTPVLHVHTASIACRVSEITSRIDPKTGKEAEK-NPQFIKAGDSAIVKFKPIKELVAEKFREFPALG
RKGYTPVLDCHTSHIACKFEELLSKIDRRTGKSMEGGEPEYIKNGDSALVKIVPTKPLCVEEFAKFPPLG
KNGYTPLLDCHTSHISCKFLNIDSKIDKRSGKVVEE-NPKAIKSGDSALVSLEPKKPMVVETFTEYPPLG

SFAIRDMGQTIAAGKV-------LG-VN-E----R
RFAIRDMGMTVAAGMA-------IQ-VTAK--N-K
RFAVLDMGQTVAAGQC-------ID-LE-K----R
RFAIRDMGQTVAAGMV-------IS-IQ-K--A-E
RFAMRDMGKTVGVGVI-------ID-VKPRKVEVK
RFAVRDMKQTVAVGVV-------KA-VT------P
RFAIRDMRQTIAVGIINQLKRKNLGAVTAKAPAKK

 Alignment Length    : 455
```

Alignment of seven EF-1α protein sequences using natural gap costs

```
MSDE-QHQNLAIIGHVDHGKSTLVGRLLYETGSVPEHVIEQHKEEAEEKGKGGFEFAYVMDNLAEERERG
MAKTKPILNVAFIGHVDAGKSTTVGRLLLDGGAIDPQLIVRLRKEAEEKGKAGFEFAYVMDGLKEERERG
MASQKPHLNLITIGHVDHGKSTLVGRLLYEHGEIPAHIIEEYRKEAEQKGKATFEFAWVMDRFKEERERG
MAKEKPHINIVFIGHVDHGKSTTIGRLLFDTANIPENIIKKFEEMGE-KGK-SFKFAWVMDRLKEERERG
MS-QKPHLNLIVIGHVDHGKSTLIGRLLMDRGFIDEKTVKEAEEAAKKLGKDSEKYAFLMDRLKEERERG
MPKEKTHINIVVIGHVDSGKSTTTGHLIYKCGGIDQRTIEKFEKESAEMGKGSFKYAWVLDNLKAERERG
MGKEKTHINLVVIGHVDSGKSTTTGHIIYKLGGIDRRTIEKFEKESAEMGKGSFKYAWVLDKLKAERERG

VTIDIAHQEFSTDTYDFTIVDCPGHRDFVKNMITGASQADNAVLVVAADDG-----V--QPQTQEHVFLA
VTIDVAHKKFPTAKYEVTIVDCPGHRDFIKNMITGASQADAAVLVVNVDDAK--SGI--QPQTREHVFLI
VTIDLAHRKFETDKYYFTLIDAPGHRDFVKNMITGTSQADAAILVISARDG---EGV--MEQTREHAFLA
ITIDVAHTKFETPHRYITIIDAPGHRDFVKNMITGASQADAAVLVVAVTDG-----V--MPQTKEHAFLA
VTINLSFMRFETRKYFFTVIDAPGHRDFVKNMITGASQADAAILVVSAKKGEYEAGMSAEGQTREHIILS
ITIDISLWKFETSKYYFTIIDAPGHRDFIKNMITGTSQADVAILIVAAGTGEFEAGISKNGQTREHILLS
ITIDIALWKFETPRYFFTVIDAPGHKDFIKNMITGTSQADVALLVVPADVGGFDGAFSKEGQTKEHVLLA

RTLGIGELIVAVNKMDLVD--YGESEYKQVVEEVKD-LLTQVRFDSENAKFIPVSAFEGDNIAEESEHTG
RTLGVRQLAVAVNKMDTVN--FSEADYNELKKMIGDQLLKMIGFNPEQINFVPVASLHGDNVFKKSERNP
RTLGVPQMVVAINKMDATSPPYSEKRYNEVKADAEK-LLRSIGFK--DISFVPISGYKGDNVTKPSPNMP
RTLGINNILVAVNKMDMVN--YDEKKFKAVAEQVKK-LLMMLGYK--NFPIIPISAWEGDNVVKKSDKMP
KTMGINQVIVAINKMDLADTPYDEKRFKEIVDTVSK-FMKSFGFDMNKVKFVPVVAPDGDNVTHKSTKMP
YTLGVKQMIVGVNKMDAIQ--YKQERYEEIKKEISA-FLKKTGYNPDKIPFVPISGFQGDNMIEPSTNMP
FTLGVKQIVVGVNKMDTVK--YSEDRYEEIKKEVKD-YLKKVGYQADKVDFIPISGFEGDNLIEKSDKTP

WYDGEILLEALNELPAPEPPTDAPLRLPIQDVYTISGIGTVPVGRVETGILNTGDNVSFQPSDVS-----
WYKGPTIAEVIDGFQPPEKPTNLPLRLPIQDVYTITGVGTVPVGRVETGIIKPGDKVVFEPAGAI-----
```

```
WYKGPTLLQALDAFKVPEKPINKPLRIPVEDVYSITGIGTVPVGRVETGVLKPGDKVIFLPADKQ-----
WYNGPTLIEALDQMPEPPKPTDKPLRIPIQDVYSIKGVGTVPVGRVETGVLRVGDVVIFEPASTIFHKPI
WYNGPTLEELLDQLEIPPKPVDKPLRIPIQEVYSISGVGVVPVGRIESGVLKVGDKIVFMPVGKI-----
WYKGPTLIGALDSVTPPERPVDKPLRLPLQDVYKISGIGTVPVGRVETGILKPGTIVQFAPSGVS-----
WYKGRTLIEALDTMQPPKRPYDKPLRIPLQGVYKIGGIGTVPVGRVETGILKAGMVLNFAPSAVV-----


-GEVKTVEMHHEEVPKAEPGDNVGFNVRGVGKDDIRRGDVCGPADDPPSVA---ETFQAQIVVMQHPSVI
-GEIKTVEMHHEQLPSAEPGDNIGFNVRGVGKKDIKRGDVLGHTTNPPTVA---TDFTAQIVVLQHPSVL
-GDVKSIEMHHEPLQQAEPGDNIGFNVRGIAKNDIKRGDVCGHLDTPPTVV---KAFTAQIIVLNHPSVI
QGEVKSIEMHHEPMQEALPGDNIGFNVRGVGKNDIKRGDVAGHTNNPPTVVRPKDTFKAQIIVLNHPTAI
-GEVRSIETHHTKIDKAEPGDNIGFNVRGVEKKDVKRGDVAGSVQNPPTVA---DEFTAQVIVIWHPTAV
-SECKSIEMHHTALAQAIPGDNVGFNVRNLTVKDIKRGNVASDAKNQPAVGC--EDFTAQVIVMNHPGQI
-SECKSVEMHKEVLEEARPGDNIGFNVKNVSVKEIKRGYVASDTKNEPAKGC--SKFTAQVIILNHPGEI

TEGYTPVFHAHTAQVACTVESIDKKIDPSSGEVAEE-NPDFIQNGDAAVVTVRPQKPLSIEPSSEIPELG
TDGYTPVFHTHTAQIACTFAEIQKKLNPATGEVLEE-NPDFLKAGDAAIVKLIPTKPMVIESVKEIPQLG
APGYKPVFHVHTAQVACRIDEIVKTLNPKDGTTLKE-KPDFIKNGDVAIVKVIPDKPLVIEKVSEIPQLG
TVGYTPVLHAHTLQVAVRFEQLLAKLDPRTGNIVEE-NPQFIKTGDSAIVVLRPTKPMVIEPVKEIPQMG
GVGYTPVLHVHTASIACRVSEITSRIDPKTGKEAEK-NPQFIKAGDSAIVKFKPIKELVAEKFREFPALG
RKGYTPVLDCHTSHIACKFEELLSKIDRRTGKSMEGGEPEYIKNGDSALVKIVPTKPLCVEEFAKFPPLG
KNGYTPLLDCHTSHISCKFLNIDSKIDKRSGKVVEE-NPKAIKSGDSALVSLEPKKPMVVETFTEYPPLG

SFAIRDMGQTIAAGKVLGVN-------------ER
RFAIRDMGMTVAAGMAIQVT-----------AKNK
RFAVLDMGQTVAAGQCIDLE-------------KR
RFAIRDMGQTVAAGMVISIQ------------KAE
RFAMRDMGKTVGVGVIIDVK--------PRKVEVK
RFAVRDMKQTVAVGVVKAVT--------------P
RFAIRDMRQTIAVGIINQLKRKNLGAVTAKAPAKK

Alignment Length  455
```

3.5 Data Structures Employed in the Implementation

It should be obvious that the efficiency of the algorithm is largely determined by the data structures employed to store the graphs and various paths through them. The two main lists to be stored are the open and the closed lists of vertices. The space complexity of the algorithm is then the size of the set $S \cup \overline{S}$. Each vertex in this set needs to be stored, along with some values associated with it, and accessed as fast as possible. In addition we need to maintain the set of searched vertices so that we can quickly access the vertex with maximum value of metric, and also perform insertions efficiently. With these aims in mind, the following data structures were employed.

1. *Data structure for vertex records:* Vertex records contain the information regarding the co-ordinates, shortest distance and estimator value for the vertices searched in course of the algorithm. Vertex records are stored in a hash table, where the hash function is a unique function of the first two co-ordinates of the vertex. Each hash bucket is a linked list of vertex records. This scheme provides access to a vertex record in almost constant time if the number of sequences is not very large.

2. *Data structure for the searched vertex set:* The vertices to be expanded are maintained as a stack. This stack is composed of bins(stack-lists) which correspond to the list of pointers to vertex records based ona metric which is the estimated shortest distance for a path from s to t passing through the vertex. For a vertex v the metric is given by $(p(v) + h(v,t))$. Thus, constant time access is provided to the vertex to be extended next. Moreover, a pointer to a vertex-record can be added to stack in the constant time.

4 Conclusions and Future Work

In conclusion we would like to present some insights obtained while implementing various algorithms. In hindsight, the stack algorithm, though traditionally employed in a different context from Dijkstra algorithm and its variants, has many similarities with the latter. In trying to obtain an optimal path, each of the above algorithms examines one path at a time, namely the one with the best metric. Dijkstra's algorithm chooses a node with the shortest path length from the initial vertex; the A* algorithm uses a combination of the shortest path length to a given node and an estimate of the shortest path length from the node to the destination vertex; the stack uses log-odds scores as the metric. As far as data structures for the implementation are concerned, they are virtually the same. Interestingly there are historical analogies as well. The stack algorithm was used as a good approximation to the Viterbi algorithm for decoding convolutional codes; the Viterbi algorithm is essentially a dynamic programming algorithm. The A* shortest path algorithm was originally suggested by Spouge(1989) to overcome the *dimensionality problem* in the dynamic programming approach to multiple sequence alignment.

An interesting variation to the stack algorithm is the following. Since we are looking for the path with the maximum score, instead of just using the sum of scores up to the present node, we could use the sum of scores up to a node added to an upper bound on the path length from the node to the destination. The destination here is taken to be the end of one of the sequences. To implement this one has to drop the X parameter used for discarding paths. However if the algorithm reaches the end vertex we would be sure that the topmost path on the stack was the best one. This was implemented and it was found that the gains from the avoidance of emptying the stack were offset by the extra computations on the paths that were not fruitful. A restriction of the A* algorithm to two dimensions by Araki et. al (1993), brought to our notice in the paper by Ikeda et. al(1999) is closely related to the above idea. Araki et. al have used an estimator using what are known as profile matrices. An interesting extension to the present work would be the comparison of the A* algorithm in two dimensions to the stack algorithm. We should mention that we have used the insertion and deletion costs employed by Gapped BLAST though these may not be the log-odds costs that our analysis requires. Finally, the stack algorithm can easily incorporate non-linear gap costs, if these are found to be more appropriate.

For the multiple alignment problem, we have found that the alignment using natural gap costs has less fragmented gaps than the one where each null is considered independently of its neighbors. Altschul(1989) has stated that natural gap costs are the appropriate metrics for multiple alignments. The biological significance of the alignments reported here thus needs to be assessed.

References

[1] Agarwal,P. and States,D., (1998), Comparative accuracy of methods for protein sequence similarity Search . *BioInformatics,* **14**, pp. 40-47 .

[2] Altschul,S.F., Gish,W., Miller,W., Myers,E.W., and Lipman,D.J., (1990), A basic local alignment search tool. *J. Mol. Biol.,* **215**, pp. 403-410.

[3] Altschul,S.F., Madden,T.L., Schaffer,A.A., Zhang,J., Zhang,Z., Miller,W. and Lipman,D.J. (1997), Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, **25**(17), pp. 3389-3402.

[4] Altshul,S.F., (1991), Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.*, **219**, 555-565.

[5] Altschul, S.F. ,(1989) Gap costs for multiple sequence alignment. *J. Theor. Biol.*, **138**, pp. 297-309 .

[6] Altschul, S.F and Lipman,D.J., (1989) ,Trees,stars and multiple biological alignment. *SIAM J. Appl. Math.* , **48**, pp. 197-209.

[7] Altschul,S.F. and Erickson,B.W. (1986) Optimal sequence alignment using affine gap costs, *Bull. Mathl. Biol.* **48**, pp 603-616

[8] Apostolico,A.,and Giancarlo,R.,(1998), Sequence alignment in molecular biology,*J. Comput. Biol.* **5**(2), pp 173-196.

[9] Araki,S.,Goshima,M.,Mori,S.I.,Nakashima,H.,Tomita, S.,Akiyama,Y.,Kanehisa,M. Application of parallelized DP and A* algorithm to multiple sequence alignment, *Proc. Genome Informatics Workshop IV,* pp 94-102

[10] Bacon,D.J., and Anderson, W.F.(1986). Multiple sequence alignment.*J. Mol. Biol.* **191**, 153.

[11] Bucher,P. and Hoffman,K. (1996), A sequence similarity search algorithm based on a probabilistic interpretation of an alignment scoring system. In States,D., Agarwal,P.,Gaasterland,T.,Hunter,L., and Smith,R.,(eds),*ISMB-4.* AAAI Press, Menlo Park, CA.

[12] Carrillo,H. and Lipman,D., (1988), The multiple sequence alignment broblem in biology. *SIAM J. Appl. Math.*, **48**(5), pp. 1073-1081.

[13] Chao,K.M,Pearson,W.R. and Miller,W. , (1992), Aligning two sequences within a specified diagonal band. *CABIOS*, **8**(5), pp. 481-487.

[14] Fano, R.M.,(1963), A heuristic discussion of probabilistic decoding, *IEEE Transactions on Information Theory,***IT-9**, pp 64-74.

[15] Fitch, W.M., and Smith,T.F.,(1983) Optimal sequence alignments, *Proc. Natl. Acad. Sci. U.S.A.*,**80**, 1382.

[16] GenBank, National Center for Biotechnology Information, National Library of Medicine, 38A, 8N805, 8600 Rockville Pike, Bethesda, MD 20894, USA. http://www.ncbi.nlm.nih.gov/

[17] Garey,M.R. and Johnson,D.S , (1979), *Computers and Intractability: A guide to the Theory of NP-Complete Prolems.* W.H. Freeman.

[18] Gotoh,O., (1982), An improved algorithm for matching biological sequences. *J. Mol. Biol.* , **162**, pp. 705-708.

[19] Gotoh, O.,(1986) Alignment of three biological sequences with an efficient traceback procedure,*J. Theor. Biol.***121**, pp 327.

[20] Gupta,S.K., Kececioglu, J.D., Schaffer, A.A., (1995) Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment, *J. Comput. Biol* **2**(3), 459-472.

[21] Gusfield,D., (1993), Efficient methods for multiple sequence alignments with guaranteed error bounds. *Bull. of Mathem. Biol.*, **55**, pp. 141-154.

[22] Hart,P.E., Nilson,N.J., Rafael,B. ,(1968), A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Sci. and Cybernet*, **SSC-4**, pp. 100-107.

[23] Henikoff,S. and Henikoff,J.G. (1992),Amino Acid substitution matrices from protein blocks . *Proc. Natl Acad Sci.*, **89**, pp. 10915-10919.

[24] Henikoff, S.,(1994) Comparitive sequence analysis: finding genes, in *BIOCOMPUTING: Informatics and Genome Projects*, Academic Press.

[25] Ikeda, T. and Imai, H., (1999), Enhanced A* algorithms for multiple alignments : optimal alignments for several sequences and k-opt approximate alignments for large cases . *Theoretical Comp. Sci.*, **210**, pp. 341-374.

[26] Jelinek F., (1969), A fast sequential decoding algorithm using a stack. *IBM J. Res. Develop.*, **13**, pp 675-685 .

[27] Lipman,D.J., and Pearson,W.R., (1985), Rapid and sensitive protein similarity searches.*Science*, **227**,pp. 1435-1441.

[28] Massey,J.L., (1976),Error Bounds for Tree Codes, Trellis Codes, and Convolutional Codes with Encoding and Decoding Procedures, in *Coding and Complexity , CISM Courses and Lectures No. 216*, G.Longo(Ed), Springer Verlag.

[29] Murata,M., Richardson,J.S. and Sussman,J.L.(1985),Simultaneous comparison of three protein sequences, *Proc.Natl.Acad. Sci. U.S.A.***82**,7657.

[30] Myers,E.W., and Miller,W., (1988), Optimal alignments in linear space. *CABIOS*, **4**(1), pp. 11-17.

[31] Needleman,S.B., and Wunsch,C.D., (1970), A general method applicable to the search for similarities in the amino-acid sequence of two proteins. *J. Mol. Biol.* , **48**, pp. 443-453.

[32] Sankoff,D., and Cedergren,R.J.(1983). Simultaneous comparison of three or more sequences related by a tree. In:*Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence comparison.*(Sankoff,D. and Kruskal,J.B.,eds) pp253-263, Reading,MA:Addison Wesley.

[33] Sankoff,D. and Kruskal, J.B. eds., (1983), *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison* . Addison-Wesley, London.

[34] Sellers,P.H., (1980), The theory and computation of evolutionary distances: pattern recognition. *J. Algorithms*, **1**, 359-373.

[35] Smith,T.F. and Waterman,M.S., (1981), Identification of common molecular sequences. *J. Mol. Biol.*, **141**, pp. 195-197.

[36] Spouge, J.L., (1989), Speeding up dynamic programming for finding optimal lattice paths. *SIAM J. Appl. Math.* **49**, pp. 1552-1566.

[37] Waterman,M.S., (1984), Efficient sequence alignment algorithms. *J. Theo. Biol.* , **108**, pp. 333-337.

[38] Zhang,Z., Berman,P. and Miller,W., (1998), Alignments without Low Scoring Regions, *J. of Comput. Biol.*, **5**(2), pp 197-210.

[39] Zigangirov, K.S., (1966), Some Sequential decoding Procedures. *Probl. Peredach. Inform.*, 2, pp. 13-25.