Constructing Sequence Alignments from a Markov Decision model with Estimated Parameter Values [*]

Fern Y. Hunt
Anthony J. Kearsley
Agnes O'Gallagher
Mathematical and Computational Sciences Division
National Institute of Standards and Technology
Gaithersburg, Maryland 20899

May 20, 2004

Abstract

Current methods for aligning biological sequences are based on dynamic programming algorithms. If large numbers of sequences or a number of long ones are to be aligned the required computations are expensive in memory and CPU time. In an attempt to bring the tools of large scale linear programming (LP) methods to bear on this problem, we formulate the alignment process as a controlled Markov chain and construct a suggested alignment based on policies that minimize the expected total cost of the alignment. We will discuss the LP associated with the total expected discounted cost and show the results of a solution of the problem based on a primal-dual interior point method. Model parameters, estimated from aligned sequences along with cost function parameters are used to construct the objective and constraint conditions of the LP. We conclude with a discussion of some alignments obtained from the the LP solutions of problems with various cost function parameter values.

KEYWORDS: Multiple Sequence Alignment, Linear Programming, Markov Decision Processes

[*]Contribution of the National Institute of Standards and Technology and not subject to copyright in the United States.

1 Introduction

Large scale sequencing of DNA has triggered an accumulation of massive amounts of both DNA sequence data and protein data. Applying this information to important problems in areas such as biotechnology, law enforcement and medicine depend on a technique known as sequence alignment. For example, the task of searching a database for a close match between a given DNA sequence and selected members of the database is often accomplished this way. This is also the case with proteins. Information about the biological function or structure of a protein is sought by identifying similar members of a database of proteins with known functional and structural properties.

Alignment methods are based on the assumption that present day protein sequences descended from common ancestor protein sequences. In the course of evolution, differences due to mutation and other genome changes often occur in such a way that biological function is preserved or only slightly modified. Given a model of protein sequence evolution, sequences in a database that appear related to the query sequence have high scores and are consequently identified as "hits". Thus aligning protein sequences is a way to infer information about protein function and structure. Given a set of letter sequences, each letter representing an amino acid residue, an alignment is an array that displays some relationship of the sequences through matching and equivalent substitutions of corresponding amino acid residues and insertions of gap characters that are used to maximize the degree of similarity. The quality of the alignment is evaluated by assigning a total "score" or "cost". The goal is to attain an optimal alignment. We can formalize this as follows:

Definition [14] Given a set of k sequences, $S = \{s_1, \cdots s_k\}$ where each $s_i, i = 1 \cdots k$ is a string of symbols from a finite alphabet $\mathcal{G}$, a $\underline{\text{multiple sequence alignment}}$ is an array $\mathsf{A} = \langle \sigma_1, \cdots \sigma_k \rangle^T$ with σ_i a string from the finite alphabet $\mathcal{G}' = \mathcal{G} \cup \{-\}$, and where $L = |\sigma_i|$, called here the length of the alignment, is the length of the string $\sigma_i, \forall i$. The sequence obtained from σ_i by removing all "–" gap characters is equal to s_i. Algorithms for multiple alignment are based on the minimization of the total cost over the set of all alignments A,

$$\min_{\mathsf{A}} \sum_j c(\mathsf{a_j}), \tag{1}$$

where $\mathsf{a_j}$ is the jth column of the array, and $c(\mathsf{a})$, the cost of the aligned column a, is specified in advance [2]. When each symbol is a letter of the alphabet associated with one of the 20 amino acids or a gap, a column of an alignment is assigned a cost based on the probability the acids represented by the letters in the column descend from a common ancestor. Scoring functions provide for the possibility of gaps which correspond to the insertion or deletion of amino acids. Several tables based on widely accepted models of protein evolution can be employed. Two examples are the **PAM** and the **BLOSUM** scoring matrices ([8]). Clearly the choice of a cost function is essential to locating a meaningful solution alignment but the present work concentrates on the construction and solution of (1).

Dynamic programming algorithms are often employed for solving this problem [15] particularly for aligning small numbers of sequences. However, as alignments for larger numbers of sequences are sought, speed and memory demands suggest that an alternative method could be valuable.

Using an approach similar to the hidden Markov models technique, [8, 19] aligned sequences are modeled as sample paths of a statistical process in an enhanced state space. The parameters of the process are estimated from aligned protein sequences as is done in the training process for hidden Markov models. Alignment is then treated as a stochastic control problem. Rather than calculate the minimum cost of a fixed alignment, we seek to minimize the expected or average cost of a family of alignments that are related to each other by a model. This can be done within the framework of Markov decision theory. Here a linear programming problem formulation is employed providing an inexpensive, effective and robust alternative to solving large-scale dynamic programming problems. Our goal here is to outline the basic ideas of the Markov decision model and to present the results of our first steps towards constructing alignments of sequences coming from a single protein family. In section 2 basic ideas in Markov decision theory are presented with particular attention to the expected total discounted cost. This is one of the most frequently discussed costs in the literature. The associated linear programming problem comes from a fundamental result due to Richard Bellman. In Section 3 we show that the effects of changes in the cost function parameters on the solution of the LP, i.e. its sensitivity can be estimated by the solution of the dual LP. The elements of this solution vector are called Lagrange multipliers and Bellman's result is used to establish their existence. In section 4, we describe the implementation of the algorithm and end with a study of the alignments of sets of triple sequences coming from the training set. Values of the cost function parameters leading to the most consistency between the alignments obtained from the algorithm and the alignments performed by CLUSTALW are identified.

2 Markov Decision Process:

A Markov chain is a random sequence of elements Z_t indexed by a non-negative integer time variable $t = 0, 1 \cdots$, with each $Z_t \in \mathbf{Z}$. An individual element of $\mathbf{Z}$ is called a state. The transition from $Z_t = i$ to $Z_{t+1} = j$ is a random event whose probability is given by $P_{ij} = Prob\{Z_{t+1} = j | Z_t = i\} \geq 0$ with $\sum_{j \in \mathbf{Z}} P_{ij} = 1$. Here we assume the number of states is finite. Over the course of time, the chain transitions from one state to another. When the state space is finite and the chain is not periodic it is called ergodic if a state can be reached with positive probability by any other state through a sequence of transitions (see Chapter 4 [17] for further details).

A Markov decision process is a family of Markov chains indexed by elements of a set called actions and denoted by $\mathcal{A}$. Thus for each $a \in \mathcal{A}$, we have $\{P_{ij}(a)\}$, as transition probabilities for a Markov chain that for each a satisfy $\sum_{j=1}^{m} P_{ij}(a) = 1$ where m is the number of states. Here each state is identified with one of the numbers $\{1, 2, \cdots m\}$. Over time, the states of the Markov chain and the index of the chain can change. Thus at time t, the state at time t call it $X_t = i$ and the the index of the chain at time t, $a_t = a$ are used to determine what the state X_{t+1} will be through the transition probability $Prob\{X_{t+1} = j | X_t = i, a_t = a\} = P_{ij}(a)$. By choosing an action at each time step, a controller decides which Markov chain will be used to generate the next state by looking at the past states and actions. The rule the controller uses for selecting an action is called a policy. In this work we will consider only stationary deterministic policies. This means there is a function $f : \mathbf{Z} \to \mathcal{A}$ such that the action at any time t satisifes $a_t = f(X_t)$. When the state and action spaces are finite, it is sufficient to consider stationary deterministic policies.

An alignment of k sequences can be interpreted as a sample path of a random process where the state of the process at time t is the t-th column of the alignment. Thus X_t is a k-tuple of symbols from an alphabet. At the next time step the process transitions to a new state X_{t+1} determined by X_t and the action a_t taken by at time t. From the point of view of stochastic control, an action is a decision at each time t about which positions in the $t+1$st column will contain the gap symbol "-". Thus a_t can be encoded as a column vector with k elements of 0 or 1 depending on whether or not a gap symbol is used in a given row. To fix ideas consider the task of finding an optimal global alignment of 3 protein sequences. The associated Markov Decision Process (MDP) model is based on a set $\mathbb{P}$ of 20 letters representing amino acids. The alphabet for the states is based on $\mathbb{P} \cup \{-\}$ and thus the state space $\mathbf{X} = \{\mathbb{P} \cup -\}^{\mathbf{3}}$, consist of column vectors with 3 elements coming from the set $\{\mathbb{P} \cup -\}$. The protein alignment problem therefore, has a state space with 21^3 elements and 8 actions. Each pair (X_t, a_t) has a cost $C(X_t, a_t)$ associated with state X_t and action a_t. We will assume these costs are known. The goal is to find a sequence of actions that results in a minimum total alignment cost (or maximum alignment score). A variety of expressions for the total cost exist but one that is frequently discussed in the literature is the expected discounted total cost ([7]). Given the path $(X_0, a_0, X_1, a_1, \cdots)$ and a discount rate α, $0 < \alpha < 1$, the expected (infinite horizon) discounted cost under policy β and starting at state i is,

$$V_{\alpha,\beta}(i) = \mathbf{E}_\beta[\sum_{t=0}^{\infty} \alpha^t C(X_t, a_t) | X_0 = i]. \tag{2}$$

Another frequently discussed cost is the long term expected average cost. If L is the length of an alignment i.e. the length of an aligned sequence, then this cost starting at i is

$$\widehat{U}_\beta(i) = \overline{\lim}_{L\to\infty} \frac{1}{L} \sum_{t=1}^{L} \mathbf{E}_\beta[C(X_t, a_t) | X_0 = i]. \tag{3}$$

In the case where there are an infinite number of positive costs, the sum in equation (2) could diverge as $\alpha \to 1$, but it can be shown that when β is a stationary policy and α approaches 1, the quantity $(1-\alpha)V_{\alpha,\beta}$ converges to $\widehat{U}_\beta$ (see [1]). We now sketch a derivation of the LP problem for the cost given in (2) and show how the solution is used to obtain an alignment.

Let $V_\alpha(i) = \inf_\beta V_{\alpha,\beta}(i)$. Then $V_\alpha(i)$ is the least mean discounted cost for every starting state i. The goal is to find an optimal policy, say β^*, such that $V_{\alpha,\beta^*}(i) = V_\alpha(i)$ for every state i. Such an optimal β^* exists and is often a stationary and/or a deterministic policy. The linear programming problem can be derived as a consequence of the fact that V_α satisfies the Bellman equation:

$$V_\alpha(i) = \min_a [C(i,a) + \alpha \sum_{j=1}^{m} P_{ij}(a) V_\alpha(j)] \tag{4}$$

See [16] for a proof of this. Let $B(\mathbf{X}) = \{u | u : \mathbf{X} \to \mathbb{R}\}$, be the space of real valued functions defined on $\mathbf{X}$ with norm $\|u\| = \sup_{i\in\mathbf{X}} |u(i)|$. Define the operator $T_\alpha : B(\mathbf{X}) \to B(\mathbf{X})$, by

$$T_\alpha u(i) = \min_a \{C(i,a) + \alpha \sum_{j=1}^{m} P_{ij}(a)u(j)\}.$$

When $0 < \alpha < 1$, the operator T_α is a contraction mapping with unique fixed point $V_\alpha \geq 0$ [16]. For any $u \in B(\mathbf{X})$, $T_\alpha^{(n)} u \to V_\alpha$ as $n \to \infty$. Here $T_\alpha^{(n)}$ means applying T_α n times. Supposing $T_\alpha u \geq u$, repeated application of T_α yields $V_\alpha \geq u$. Thus V_α is the largest function satisfying the condition $T_\alpha u \geq u$. It is natural then to use the following method to compute $V_\alpha(i)$. If m is the number of states we pose the following optimization problem for the expected discounted cost.

$$\max \sum_{i=1}^{m} u(i) \tag{5}$$

subject to

$$C(i,a) + \alpha \sum_{j=1}^{m} P_{ij}(a)u(j) \geq u(i) \tag{6}$$

Such an optimal u can be shown to be V_α (see section 3.2 in [1] also [16]).

Now let a_i be such that

$$C(i,a_i) + \alpha \sum_{j=1}^{m} P_{ij}(a_i)V_\alpha(j) = \min_a [C(i,a) + \alpha \sum_{j=1}^{m} P_{ij}(a)V_\alpha(j)]. \tag{7}$$

Setting $f_\alpha(i) = a_i$ an optimal stationary deterministic policy is defined. This policy yields the sequence of actions that leads to the optimal alignment defined by the expected discounted cost for a fixed α. A crucial element of the effectiveness of this method is the construction of of $P_{ij}(a)$ from alignment data. The estimation of these parameters from alignment data will be discussed in a later section.

3 Sensitivity of Optimal Cost LP

An interesting question for all multiple sequence algorithms is the effect of small changes in the cost parameters on the alignment. We can make a step towards answering this question for the problem (5)-(6) by examining the effect of such parameter changes on the optimal cost. The Karush- Kuhn- Tucker theorem can be used to give us information about the sensitivity of the optimal cost in (5) because it implies the existence of Lagrange multipliers [5]. These numbers are the components of the solution vector of the dual linear programming problem and they provide a measure of the sensitivity (i.e. the amount of change) in the optimal cost due to changes in the corresponding constraint. In particular, indices where components vanish are positions of “inactive” constraints, i.e. the corresponding components of the cost vector C where small changes have no effect on the optimal cost. Lagrange multipliers exist for the problem (5)- (6)- a consequence of Bellman's equation. This result is well known but in this section we present the proof of this fact for convenience. First we begin with a more formal description of Lagrange multipliers. This is based on some ideas from optimization theory which we briefly describe next [4]. Suppose we are given the problem,

$$\begin{aligned} &\inf \quad f(x) \\ \text{subject to: } &g_i(x) \leq 0, \qquad i = 1, 2 \cdots, n \\ &x \in \mathcal{C} \end{aligned} \tag{8}$$

where $f : \mathbb{R}^m \to \mathbb{R}$ and $g_k : \mathbb{R}^m \to \mathbb{R}, k = 1, 2, \cdots, n$ are differentiable functions and $\mathcal{C}$ is a convex subset of $\mathbb{R}^m$. A point $x \in \mathcal{C}$ that satisfies the constraints (8) is called <u>feasible</u>. A point $\overline{x} \in \mathcal{C}$ is called a local minimizer if $f(x) \geq f(\overline{x})$ for all feasible x that are close to $\overline{x}$. Given such an $\overline{x}$, assuming that $\overline{x} \in int(\mathcal{C})$, we call a vector $\lambda \in \mathbb{R}^n_+$ a *Lagrange multiplier vector* if $\overline{x}$ is a critical point of the Lagrangian, $L(x; \lambda) = f(x) + \sum_{k=1}^n \lambda_k g_k(x)$. Thus,

$$\nabla f(\overline{x}) + \sum_{i \in I(\overline{x})} \lambda_i \nabla g_i(\overline{x}) = 0$$

where $I(\overline{x}) = \{k | g_k(\overline{x}) = 0\}$ is the so-called set of active constraints. Secondly, we require that a complementary slackness condition be satisfied. If $k \notin I(\overline{x})$ we have $\lambda_k = 0$, otherwise $\lambda_k > 0$. The Karush-Kuhn-Tucker theorem provides sufficient conditions for the existence of Lagrange multipliers in terms of the a condition on the gradient vectors $\nabla g_k(x) \quad k = 1, 2, \cdots, n$. This condition holds for example when they are linearly independent [4]. Let A be the matrix defined by the constraint equations in (5)-(6). The conditions of the Karush-Kuhn-Tucker theorem will be met if the submatrix formed by rows with indices in $I(\overline{x})$ is non-singular. Now A is given by,

$$A = \begin{pmatrix} I - \alpha \mathbb{P}(1) \\ \vdots \\ I - \alpha \mathbb{P}(a) \\ \vdots \\ I - \alpha \mathbb{P}(q) \end{pmatrix}$$

where $a = 1, 2, \cdots, q$ indexes the actions of the MDP, $\mathbb{P}(a) = \{P_{ij}(a), i, j = 1, 2, \cdots m\}$ and I is the $m \times m$ identity matrix. Any row with index in $I(\overline{x})$ can be associated with the pair $(i, \overline{a})$ denoting the i-th row of the matrix $\mathbb{P}(\overline{a})$. At such a position the solution $\overline{x}$ of the LP satisfies,

$$\overline{x}(i) - \alpha \sum_{j=1}^m P_{i,j}(\overline{a}) \overline{x}(j) = C(i, \overline{a}) \tag{9}$$

Suppose a non-optimal vector u satisfies the constraints (6) and for all $(i, \overline{a}) \in I(\overline{x})$ satisfies (9). Then for any a, $u(i) = C(i, \overline{a}) + \alpha \sum_{j=1}^n P_{ij}(\overline{a}) u(j) \leq C(i, a) + \alpha \sum_{j=1}^n P_{ij}(a) u(j)$. Therefore, $\min_a [C(i, a) + \alpha \sum_j P_{ij}(a) u(j)] = C(i, \overline{a}) + \alpha \sum_j P_{ij}(\overline{a}) u(j)$. Thus,

$$u(i) = \min_a [C(i, a) + \alpha \sum_{j=1}^n P_{ij}(a) u(j)]$$

i.e. u satisfies Bellman's equation. We can deduce the following lemma.

Lemma 3.1 *If y is a feasible solution of (5)- (6) and satisfies (9) for any $(i,\overline{a})$ in $I(\overline{x})$, then y satisfies Bellman's equation at state i for action $\overline{a}$.*

We use this lemma to prove that the rows of the matrix A associated with the active constraints are linearly independent, whenever a positive solution to the LP (5) -(6) exists.

Theorem 3.2 *If X is a solution of the linear programming problem (5)- (6), with all components positive, then the $m \times m$ submatrix of A associated with the active constraints has linearly independent rows.*

Proof: The proof is by contradiction. If the $n \times m$ submatrix does not have m linearly independent rows, there is a non-zero column vector u that solves the following set of linear equations.

$$u(i) - \alpha \sum_{j=1}^{m} P_{ij}(a)u(j) = 0.$$

Set $\sigma_X(i,a) = C(i,a) - [X(i) - \alpha \sum_{j=1}^{m} P_{ij}(a)X(j)]$ and $\rho_u(i,a) = u(i) - \alpha \sum_{j=1}^{n} P_{ij}(a)u(j)$. Now if (i,a) corresponds to an index in $I(X)$, $\sigma_X(i,a) = 0$ because we are at an active constraint. Meanwhile $\rho_u(i,a) = 0$ by assumption. When the index is not in $I(X)$, $\sigma_X(i,a) > 0$ because X is feasible. We choose $\epsilon > 0$ so small that for all i and a, $\sigma_X(i,a) - \epsilon\rho_u(i,a) \geq 0$. If $Y = X + \epsilon u$, note that

$$C(i,a) - [Y(i) - \alpha \sum_{j=1}^{n} P_{ij}(a)Y(j)] = \sigma_X(i,a) - \epsilon\rho_u(i,a) \geq 0. \tag{10}$$

By choosing a smaller ϵ if necessary we can also insure that $Y(i) \geq 0$ because $X(i) > 0$. At (i,a) corresponding to active constraints, the left hand side of (10) is zero so Y satisfies the hypotheses of lemma 3.1. Hence Y satisfies Bellman's equation. The fact that u is non-trivial now means that X and Y are distinct solutions of Bellman's equation. However as we noted in section 1, the solution of Bellman's equation is unique and hence $X = V_\alpha = Y$, thus all the components of u vanish. This gives us the required contradiction. □

4 Software and Computer Experiments

We have written a software package that combines a matrix generator for the linear programming problem defined by (5) and (6) with an LP solver. The process then finds for each state i the minimum over all a as in (7), thereby producing the suggested alignment. After a discussion of the computational issues involved in implementing this algorithm in sections 4.1-4.3, we will discuss several alignments that come from the solution of the LP. Alignments were constructed for sets of 3 sequences coming from the data used to estimate the parameters of the model. They were evaluated for consistency with the original data - a set of sequences aligned with CLUSTALW. In other words, we evaluated the degree of agreement of the LP created alignment with the original. The behavior of these alignments as a function of cost function parameters is discussed in section 4.4.

4.1 Cost and other parameters for the MDP

The objective functions dealt with in the software, like all those used in the linear programming problems discussed in this paper, depend on a set of predefined constants that comprise an array known as a substitution or scoring matrix. Each element is a score that measures the likelihood that is, the log of the ratio of the probability of seeing an aligned pair of letters (representing a pair of amino acid residues) because they are truly related to each other by function or evolutionary relationship versus the probability that the pair are aligned simply by chance. If a column that appears in the alignment of a pair of protein sequences is assumed to be statistically independent of any other then the score of the entire aligned pair is found by summing the likelihoods of each column. If more than 2 sequences are to be aligned the most frequent method of assigning a score to a column is to add the likelihoods of all possible pairs of residues in that column. Then the score of the entire alignment is found by summing the scores of all the constituent columns. The cost of a column can be found by replacing each likelihood or pairwise score, by a cost obtained by subtracting the score from a large enough constant e.g. the maximum possible pairwise score.

Unfortunately, very little is known about the probability of a biological relationship between two arbitrary amino acids. Most estimates depend on models of protein evolution (with its attendant simplifications), empirical methods or some combination of these. Two very widely used substitution matrices are the PAM matrices which come from a Markov model of protein evolution with probabilities obtained from observations of aligned data, and the BLOSUM matrices which are the result of observing blocks of sequences aligned by hand. Recent advances in improving PAM matrices by Gonnet and Korostensky [9] on the one hand and on the other, progress in phylogenetic phylogenetic tree estimation methods [10] have brightened the prospect for a more rigorous basis for constructing cost functions for alignments.

4.2 Build Matrix of Transitions from Alignment Data

Once the the cost parameter values are known, the model parameters must be determined. These make up the matrix $P = \{P_{ij}(a) : a = 1, 2, \dots q, i, j = 1, 2, \dots m\}$ where m is the number of unique states in the training data and q is the number of actions. Therefore, P and the constraint matrix itself, i.e. $I - \alpha P$ have dimensions $(mq) \times m$. The $P_{ij}(a)$ are estimated from a set of aligned protein sequences. Counts of the number of transitions from one state i.e. an aligned column, to the next along with the action represented by the letter-gap pattern of the suceeding column are made and the frequency is taken to be the estimate of the transition probability. Thus if $n(i, j, a)$ = number of times $X_t = i, X_{t+1} = j$ and action a is taken, and $n(i, a)$=number of times $X_t = i$ and action a is taken then

$$P_{ij}(a) \approx n(i, j, a)/n(i, a).$$

Finding these frequencies requires cycling through the training data on the order of mq times. This part of our program does provide opportunities for parallelization. In the current code, we take advantage of the efficiency of Fortran90 vector operations.

4.3 Interface with LP Solver

The matrix generator writes the LP problem in MPS format. This allows us to use any of several, available LP solvers. We have, been using PCx [6] which uses a primal-dual interior-point method and is written in C. The matrix generator and PCx can be run separately, or, if desired, run from within a Java graphical user interface which also calls the Fortran 90 programs that find and test the suggested alignment. All the parts of the package, and the Java GUI run on a Sun workstation and also on a PC using a command window in a Windows environment. We timed our package on a set of 100 sequences of length 775 from a block of cytochrome p450 data. The number of unique states in this case was 4714. The dimensions of $I - \alpha P$ were therefore $32,998 \times 4714$. We were able to build P for this case in about 131 cpu seconds on a workstation. PCx solved the LP in about 250 seconds.

4.4 Parameter Study

We use the results from the LP solver and the problem data to find the optimizing policy as defined in (7). This means that, for any state encountered in the training data, we have a prediction for what the action of the next state should be to be properly aligned. If this prediction is accurate, we will be able to align sequences that consist of states in our training data.

As a first step towards understanding the behavior of the model, we undertook an investigation into four of these parameters, α, d, e, and ps. The parameter α is the discount factor that appears in (5). Parameters d and e are used in calculating the cost vector C, that appears on the right had side of the inequality. The gap initiation penalty d is a penalty that is applied to a transition from state i to state j when a new gap is introduced in state j. The gap extension penalty e is applied when an additional gap in j is adjoined to one that was already in state i. ps is a penalty used to discourage the choice of actions a which lead to rows in which all $P_{ij}(a) = 0$. This situation arises because of the sparseness of data. We did our tests using a modest size block of 40 aligned sequences from the cytochrome P-450 data. We generated 38 triple subsequences. We used all 38 as training data and then used the individual subsequences as test triples. We set $e = 0$ and investigated the behavior of the remaining three parameters. Several simple methods for evaluating the performance of the algorithm were used. First, for each sequence we counted the total number of discrepancies between the predicted action for each symbol and the actual action defined by the alignment. Figure#1 is a histogram showing the number of sequences that attained each percentage of correct predictions. For this data we used the Blosum62 scoring matrix with $\alpha = 0.4, ps = 1$, and $e = 0$. Generally speaking, for smaller values of α, we found that the gap initiation penalty d had very little effect, but at larger values, starting at about $\alpha = 0.6$ increasing d improved the performace The best values of α appeared to be between 0.1 and 0.5 after which the number of errors increased. Next,at each set of parameter values, we computed the total number of discrepancies or errors over all 38 sequences. The totals varied from a low of 25 for small α to a high of 3000- reflecting the divergence of the total cost when α approaches 1. Figure 2 shows the location of parameters where the total error was 25.

Figure 1: This histogram shows the number of sequences which obtained each of the shown percentages correct.

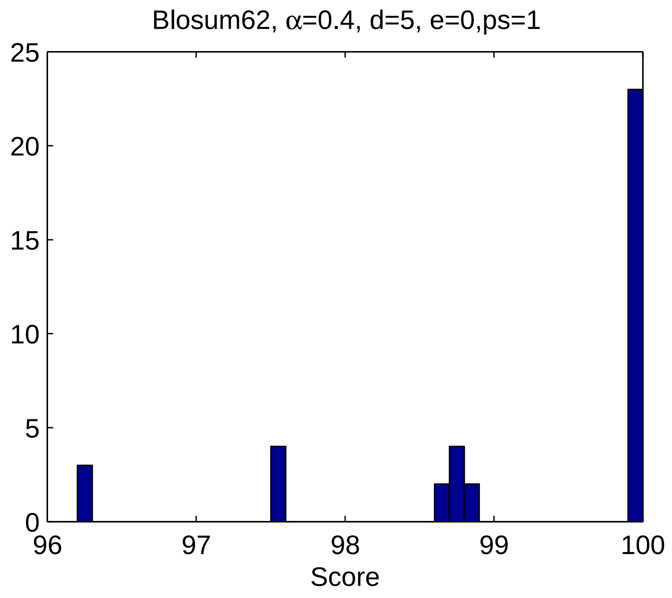
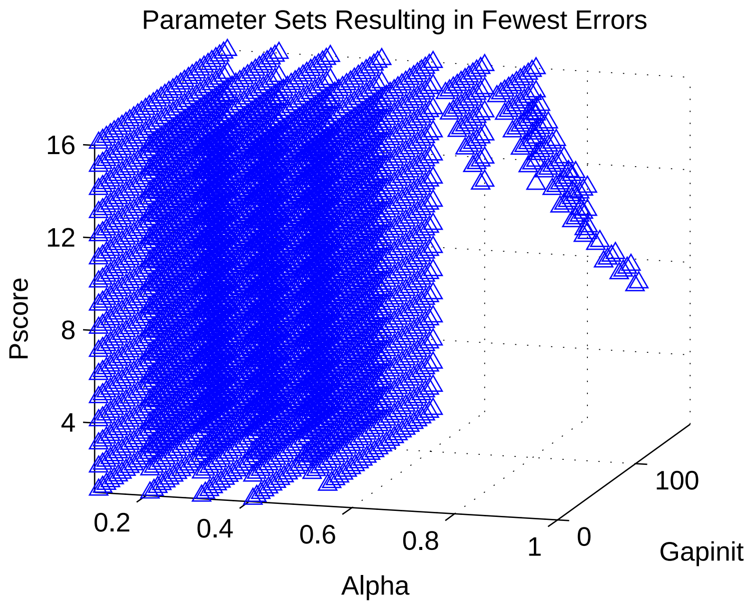


Figure 2: Triangles show combinations of parameters that produced the minumum num ber of errors.



5 Conclusions

A novel linear programming approach to multiple sequence alignment has been presented along with promising preliminary numerical results. Using modern optimization techniques this approach provides a new way of addressing a large class of important bioinformatics problems. The algorithm presented in this paper successfully locates reasonable solutions to a typical instance of the multiple sequence alignment problem in a reasonable amount of (computational) time. The algorithm is quite flexible in allowing the addition or relaxation of constraints. We have also shown that the degree of change (i.e. the sensitivity) of the linear program solution resulting from changes in the cost function can be quantified. This gives us a great deal information about the sensitivity of the alignment itself. Present and future work center on the development of methods to align sequences that are not seen in the training process. A very first step to addressing this issue is to use a sequence evolution model to create training sequences from a single data sequence. We recently used the sequence program ROSE [18] developed by Stoye, Evers and Meyer to create a set of aligned sequences from the 100,000 long DNA sequence from Chlamydophila pneumoniae J138. The matrix generator (i.e. training) part of the computation was performed on 48 sequences was slow,it took approximately an hour, but the solution of the linear programming problem and construction of the alignment of 3 sequences of that length took between 3 and 4 minutes on a Pentium III PC. [1]

ACKNOWLEDGEMENT: F.H. and A. O'G would like to express their gratitude to Dirk Evers for modifying ROSE to accomodate very long sequences.

References

[1] E. Altman 1999 *Constrained Markov Decision Processes*, Chapman and Hall/CRC, Boca Raton, London

[2] A. Apostolico, R. Giancarlo 1998 Sequence Alignment in Molecular Biology *J. Comput. Biol.*, 5:2:173-196

[3] R. E. Bixby 1992 Implementing the simplex method: The initial basis *ORSA Journal on Computing*, 4:267-284

[4] J. M. Borwein, A.S. Lewis 2000 Convex Analysis and Nonlinear Optimization: Theory and Examples Springer Verlag, New York, Berlin

[5] V. Chvátal 1983 Linear Programming, W.H. Freeman and Company, New York, N.Y.

[6] Joe Czyzyk, Sanjay Mehrotra, and Steve Wright May, 1996 PCx User Guide Technical Report OTC 96/01, Optimization Technology Center

[1] Certain commercial equipment, instruments or materials are identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

[7] C. Derman 1970 Finite State Markovian Decision Processes Academic Press

[8] R. Durbin, S. Eddy, A. Krogh, G. Mitchison 1998 Biological sequence analysis: Probabilistic models of proteins and nucleic acids, Cambridge University Press

[9] G. Gonnet, C. Korostensky 1999 Optimal Scoring Matrices for Estimating Distances Between Aligned Sequences, preprint

[10] J. Hueisenbeck, B. Rannala 1997 Phylogenetic Methods Come of Age: Testing Hypotheses in an Evolutionary Context *Science*, 276:11:227-232

[11] F. Hunt, A. J. Kearsley, A. O'Gallagher 2003 Alignment of Biological Sequences based on a Linear Programming Problem preprint

[12] F. Hunt, A. J. Kearsley, H. Wan 2003 An optimization approach to multiple sequence alignment, *Applied Mathematics Letters*, 16:785-790

[13] F. Y. Hunt 2003 Sample Path Optimality for a Markov Optimization Problem, preprint

[14] C. Korostensky, G. Gonnet 1999 Gap Heuristics and Tree Construction using Gaps, Technical Report Institute of Scientific Computing, ETH Zurich

[15] S. Needleman, C. Wunsch 1970 A general method applicable to the search for similarities in the amino acid sequence of two proteins,*J.Mol. Bio.* 48:443-453,

[16] S. Ross 1970 Applied Probability Models with Optimization Applications, Holden-Day, San Francisco

[17] S. Ross 2000 Probability Models, 7th Edition, Harcourt Academic Press, San Diego, San Francisco

[18] J. Stoye, D. Evers, F. Meyer 1998 Rose: generating sequence families, *Bioinformatics*, 14:2:157-163

[19] M. Waterman 1995 *Introduction to Computational Biology*, Chapman and Hall, London

[20] S. Wright 1997 *Primal Dual Interior Point Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA.