Optimally Separating Sequences

Gene Myers[1]
Gene.Myers@celera.com
[1] Informatics Research, Celera Genomics, 45 W. Gude Dr., Rockville, MD 20850, U.S.A.

Abstract

We consider the problem of separating two distinct classes of k similar sequences of length n over an alphabet of size s that have been optimally multi-aligned. An objective function based on minimizing the consensus score of the separated halves is introduced and we present an $O(k^3n)$ heuristic algorithm and two optimal branch-and-bound algorithms for the problem. The branch-and-bound algorithms involve progressively more powerful lower bound functions for pruning the $O(2^k)$ search tree. The simpler lower bound takes $O(n)$ time to evaluate given $O(sn)$ global data structures and the stronger bound takes $O((k+s)n)$ time by virtue of an efficient solution to the problem of finding the second-maximum envelope of a set of piece-wise affine curves. In a series of empirical trials we establish the degree to which classes can be separated using our metric and the effective pruning efficiency of the two branch-and-bound algorithms.

Keywords: branch-and-bound, multi-alignment, error correction, clustering

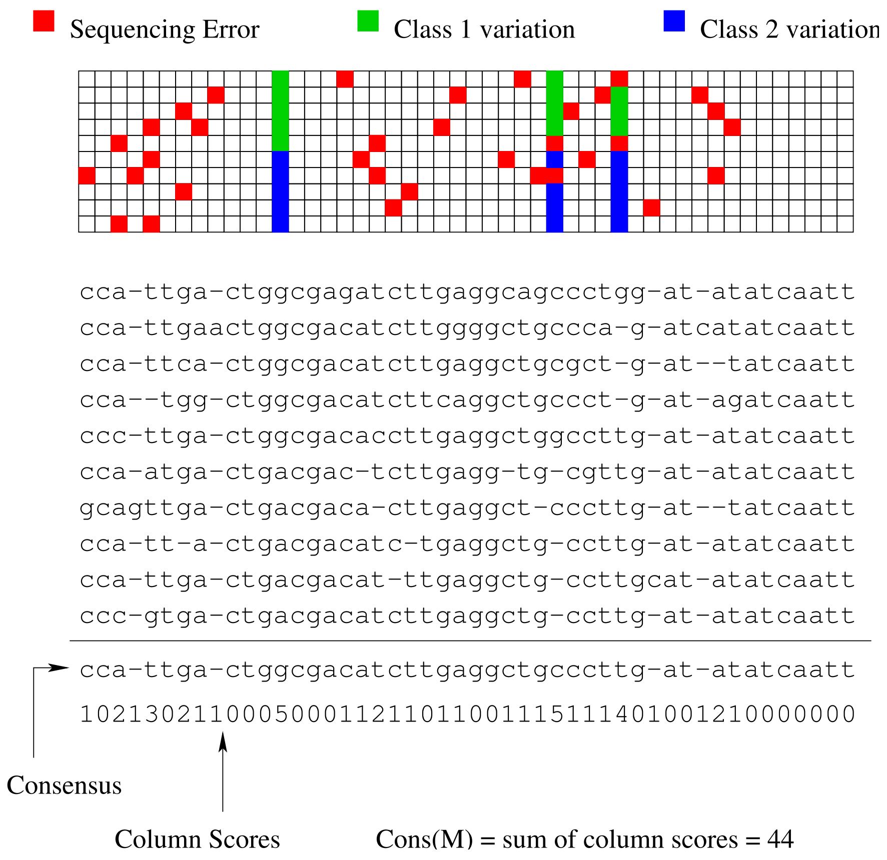
1 Introduction

Phylogenetic analysis of sequences is fundamentally a problem of separating sequences according to their dissimilarity [1]. A simpler variation of this problem occurs in the context of DNA sequence assembly, where one has noisy sequencing reads from a set of repeats all derived from the same ancestral template. Each copy of a repeat typically varies by 1 to 20% from other copies and so instances would be easily distinguishable from each other if the sequence data was exact. Unfortunately, sequencing reads have a 1 to 5% error rate and so even reads sampled from a given repeat instance will not align perfectly. The problem is to correctly separate the reads into those sampled from each distinct repeat instance, in the face of this background error.

Our work begins with a multi-alignment of highly similar DNA read sequences that is presumably optimal or near optimal, as might be produced by a good heuristic algorithm. The set of reads that have been aligned might either be all those that have a significant overlap with a seed read or might be an overcollapsed contig assembly of a collection of reads from a set of nearly identical interspersed repeats [2, 3, 4]. We will assume for simplicity that all reads cover the span of the alignment as in Figure 1. The assumption is that reads sampled from different instances of a repeat family are in the alignment. For simplicity we will again focus on the case that there are just two instances so that the reads need to be separated into two subclasses.

The basic assumption for all separation methods proposed thus far is that sequencing error is a random phenomenon whereas the differences, called micro-heterogeneities by biologists, between repeat instance will correlate in a multi-alignment of the reads. Figure 1 illustrates this, where the haze of red boxes denote errors, and the blue and green boxes denote micro-het variation between the two subclasses. Algorithms to date attempt to distinguish which differences are noise from those that are micro-hets, and to then separate the reads on the basis of the micro-het positions. Doing so heuristically is quite simple and the idea is relatively effective.

In this paper we start with a simple measure of optimal separation and develop a couple of branch-and-bound (B&B) algorithms that find the best possible separation according to this measure, i.e. the



```
cca-ttga-ctggcgagatcttgaggcagccctgg-at-atatcaatt
cca-ttgaactggcgacatcttggggctgccca-g-atcatatcaatt
cca-ttca-ctggcgacatcttgaggctgcgct-g-at--tatcaatt
cca--tgg-ctggcgacatcttcaggctgccct-g-at-agatcaatt
ccc-ttga-ctggcgacaccttgaggctggccttg-at-atatcaatt
cca-atga-ctgacgac-tcttgagg-tg-cgttg-at-atatcaatt
gcagttga-ctgacgaca-cttgaggct-cccttg-at--tatcaatt
cca-tt-a-ctgacgacatc-tgaggctg-ccttg-at-atatcaatt
cca-ttga-ctgacgacat-ttgaggctg-ccttgcat-atatcaatt
ccc-gtga-ctgacgacatcttgaggctg-ccttg-at-atatcaatt

cca-ttga-ctggcgacatcttgaggctgcccttg-at-atatcaatt
102130211000500011211011001115111401001210000000
```

Figure 1: A repeat multi-alignment illustrating the randomness of error and the correlation of micro-het differences.

algorithm is not heuristic [5]. One should note that the problem is NP-hard and the B&B algorithm could in principle take exponential time. We develop two versions of the algorithm, the second having a stronger lower bound concept that results in greater pruning of the search tree and thus better efficiency. In an empirical results section we explore the limits of the algorithm and the level to which repeats can be separated in this fashion.

2 The Problem

We begin with a multi-alignment, M, of k sequences $S_1, S_2, \ldots S_k$ over alphabet Σ. The multi-alignment M is a k by n array where row k is the sequence S_k with $n - |S_k|$ dashes interspersed. The symbol dash, `-`, is assumed not to be in Σ and we let $\Sigma^+ = \Sigma \cup \{\texttt{-}\}$ and $s = |\Sigma^+|$. It is further presumed that the dashes have been interspersed within the sequences so that the consensus score of the multi-alignment is minimal, or as near minimal as a good computer program can make it. In our work we use *realigner* for this purpose [6]. For a given column of M, $[M_{i,j}]_{j=1,k}$, let $R_i^a = |\{j : M_{i,j} = a\}|$ be the set of rows at which the symbol a occurs, for $a \in \Sigma^+$. Then the consensus character c_i for column i is the one for which $|R_i^a|$ is maximal, and the consensus score for the column is $k - |R_i^{c_i}|$. The consensus score, *cons*(M), of M is the sum of the consensus scores of its columns, that is:

$$cons(M) = \Sigma_{i=1}^{n} max_a(k - |R_i^a|) = kn - \Sigma_{i=1}^{n} max_a|R_i^a|.$$

The consensus sequence for M is obtained by taking the sequence $c_1 c_2 \ldots c_n$ and deleting all dash symbols. Figure 1 illustrates the concepts.

Given a multi-alignment M, we consider the problem of finding a partition of its rows into two sets such that the sum of the consensus scores of the two subalignments obtained by including just the rows from each set is minimal. Formally, let $X \subseteq [1,k]$ and let X^c be its complement, i.e. $[1,k]-X$. Let M_X be the alignment obtained by removing every row in X^c.

> *Best Multi-Alignment Partition Problem*: Given multi-alignment M, find a set X, such that its *partition score*, $part(M,X) = cons(M_X) + cons(M_{X^c})$, is minimal.

It is a simple exercise to verify that $part(M,X) \leq cons(M)$ for any choice of X. For example, by partitioning the multi-alignment in Figure 1 by selecting $X = [1,5]$ we obtained a combined consensus score for the two parts of 34, contrasted with the score 44 for the original multi-alignment. If the differences between the sequences in M and its consensus are few in number and randomly positioned within columns, as are the red boxes in Figure 1, then $cons(M) - part(M,X)$ is 0 in expectation for any choice of X. On the other hand, if there are p columns where h of the sequences tend to have one symbol and the remainder have another, as are the green and blue boxes in Figure 1, then the optimal partition score will generally be on the order of $p \cdot max(h, k-h)$ less then $cons(M)$, and the X achieving the optimum will embody the h sequences containing the correlated columns. For example, in Figure 1, even though only 3 columns out of 50 have such correlated differences, and even though they are not perfectly correlated, the optimal partition score is almost 25% smaller than the consensus score and is achieved only for the correct partition $X = [1,5]$. This motivates our measure of separation. Intuitively, we seek the division of the sequences into two groups so that the sum of the differences of the members from the consensus of their group is minimal.

The number of occurrences of the symbol a in column i of M_X is $|R_i^a \cap X|$. Thus:

$$\begin{aligned} part(M,X) &= cons(M_X) + cons(M_{X^c}) \\ &= \Sigma_{i=1}^n max_a(|X| - |R_i^a \cap X|) + \Sigma_{i=1}^n max_b(|X^c| - |R_i^b \cap X^c|) \\ &= kn - \Sigma_{i=1}^n(max_a|R_i^a \cap X| + max_b|R_i^b \cap X^c|). \end{aligned}$$

Because kn is fixed throughout, one sees that an equivalent objective is to maximize $|R_i^a \cap X|$ and $|R_i^b \cap X^c|$ in every column. In what follows we will think in terms of this optimization problem.

3 A Branch-and-Bound Framework

In a purely exhaustive algorithm, one could try all possible 2^k choices of X, evaluating the partition score for each choice, and identifying the best. The obvious implementation results in an $O(k2^kn)$ time algorithm. This can be reduced to $O(2^kn)$ time, by traversing a search tree modeling all subsets of $[1,k]$, incrementally building the score of each partition during the traversal. We pursue this problem now as a preliminary exercise that permits us to develop a few key concepts.

The search tree is the complete binary tree of height k. At each vertex in the tree one considers either including or excluding in X a row that has not been previously considered along the path from the root to the given vertex. These two choices are modeled as edges leading to the vertices at the next level. In order to keep the scheme simple, we will assume that the same row, π_h, is considered by every vertex at a given height h, subject to $(\pi_1, \pi_2, \ldots \pi_k)$ being a permutation of $[1,k]$. For the moment π can be the identity permutation. The 2^{k-h} vertices at height h, model all subsets of $\{\pi_{h+1}, \pi_{h+2}, \ldots \pi_k\}$ (denoted hereafter as $\pi[h+1,k]$). During a traversal of the tree one can consider maintaining data structures that are functions of the set Y modeled by the current vertex, at height say h, and its relative complement $Yc = \pi[h+1,k] - Y$. As a traversal progresses rows are added or deleted from Y and Yc, so that the challenge is to incrementally update the data structures efficiently so that they reflect these changes.

To solve our preliminary problem, one must maintain the $O(sn)$ quantities, $In_i^a(Y) = |R_i^a \cap Y|$ and $Out_i^a(Yc) = |R_i^a \cap Yc|$. Whenever Y and Yc are implicit from the context, we will drop the function dependence and for example, write simply In_i^a indicating the quantities value at the appropriate moment in time. Maintaining these values is easy in $O(n)$ time per edge traversal as $T_i^a(Z \pm x) = T_i^a(Z) \pm \delta_{x \in R_i^a}$ for any Z and T being either *In* or *Out*. What is more difficult is that one further needs to maintain the permutations $\eta_i = (\eta_i^1, \eta_i^2, \ldots \eta_i^s)$ such that $In_i^{\eta_i^1} \geq In_i^{\eta_i^2} \geq \ldots \geq In_i^{\eta_i^s}$, and ρ_i such that $Out_i^{\rho_i^1} \geq Out_i^{\rho_i^2} \geq \ldots \geq Out_i^{\rho_i^s}$. Fortunately, when an edge is traversed, only one quantity in each of a column's two orders increases or decreases by one. By maintaining an anchored doubly-linked list of the quantities with a given value, and a doubly-linked list of these anchors in order, one can incrementally update a column order in $O(1)$ time when a single quantity changes by 1. Note that even though the data structures involve a fair bit of linking overhead, space is still $O(s)$ per column, for $O(sn)$, total. Finally, when a leaf is reached, the partition score of each column is computable in $O(1)$ time as $k - (In_i^{\eta_i^1} + Out_i^{\rho_i^1})$. We thus have an $O(2^k n)$ time, $O(sn)$ space algorithm.

The algorithm above quickly becomes untenable as k increases due to its exponential complexity. Often, it is possible to greatly prune the portion of the search tree explored using the branch-and-bound paradigm. This approach requires the design of the following two elements:

1. a heuristic algorithm, *Heuristic*, that provides an initial partition set Z_{init} whose partition score, $part(M, Z_{init})$ gives an initial upper bound on the optimum score.

2. a function, *LB*, that for a vertex v gives a lower bound on the best partition score for any set modeled by a leaf that is a descendant of v. That is, if v is of height h and represents $Y \subseteq \pi[h+1, k]$, then *LB* has the property that:

$$LB(Y, h) \leq min(part(M, Y \cup X) : X \subseteq \pi[1, h]).$$

A branch-and-bound algorithm, maintains a global record of the best solution, Z_{best}, it has seen thus far, and uses the lower bound to avoid traversing the subtree rooted at a vertex v, in the event that the lower bound is not less than the score of the current best. Figure 2 gives the framework of our branch-and-bound algorithm. Because the problem is symmetric with respect to X and its complement, note that we can without loss of generality always chose to place π_1 in X. With a good initial solution Z_{init} that gets close to the optimum solution, and a tight lower bound function one can significantly prune the amount of the search tree that gets explored, sometimes to the extent of achieving polynomial performance in expectation. Employing the paradigm, thus involves designing a heuristic and a lower bound function that achieve the right compromise between efficient computation and providing bounds sufficient to prune the search tree effectively. In the next three subsections we present a heuristic and two lower bounds, the second being more intricate but also more tightly constraining than the first.

```
global $Z_{best}$: set of $[1,k]$

Procedure Search($Y$: set of $[1,k]$, $h$: int)
  { if $h = 0$ then
      { if $part(M,Y) < part(M,Z_{best})$ then $Z_{best} \leftarrow Y$ }
    else if $LB(Y,h) < part(M,Z_{best})$ then
      { Search($Y + \pi_h$,$h-1$)
        if $LB(Y,h) < part(M,Z_{best})$ and $h < k$ then
          Search($Y$,$h-1$)
      }
  }

$Z_{best} \leftarrow Heuristic()$
Search($\emptyset$,$k$)
```

Figure 2: The Branch and Bound Framework.

3.1 The Initial Heuristic Algorithm and Selection Order

Most of the heuristic programs for repeat separation begin by trying to identify columns in the multi-alignment where the two sequence classes differ. Using Figure 1 as an example, one sees that these columns are generally characteristic in that there are two symbols vying for the consensus, whereas in all other columns there is only one or two occasional "noise" disagreements with the predominant symbol. So one might say that any symbol that occurs not more than threshold parameter $\tau < k/2$ times in a column is noise, and otherwise it is signal. Any column containing two or more signal symbols is likely to denote a position of micro-heterogeneity between the two classes and should be used to partition the sequences into their respective classes. Call these the discriminating columns and denote them by D_M^τ. Formally, $D_M^\tau = \{\ i : \exists a \neq b \text{ s.t. } |R_i^a| > \tau \text{ and } |R_i^b| > \tau\}$.

Given a choice for τ that defines a set of discriminating columns, we will try to pick a good partitioning set by starting with a given row s and adding rows to its partition that have good agreement with it over the discriminating columns. To this end let $cons_\tau(s,h) = |\{\ i : i \in D_M^\tau \text{ and } M_{i,h} = M_{i,s}\}|$ be the number of discriminating columns where rows s and h have the same symbol. We can then let our heuristically chosen partition be

$$I_{s,\tau} = \{\ h :\ cons_\tau(s,h) \geq (|D_M^\tau|/2)\}$$

It takes $O(kd_\tau) = O(kn)$ time to compute $I_{s,\tau}$ where $d_\tau = |D_M^\tau|$. To insure the best possible initial guess, we compute $I_{s,\tau}$ for every choice of $s \in [1,k]$ and every choice of $\tau \in [1, \lceil k/2 \rceil - 1]$, and take the partition, I^* that gives the best score. The total time to explore all the possibilities is $O(k^3 n)$, but $\Sigma_i d_i$ is generally much smaller than kn so performance is generally much better in practice.

We will examine the performance of this heuristic in the empirical results section. For the moment we note that in any problem for which there are sufficient discriminating columns in M, the heuristic finds an optimal partition. It is thus the case that often the B&B portion of the algorithm is really rapidly confirming the optimality of this choice.

The depth at which the lower bound begins to trim effectively can depend substantially on the order π in which rows are considered for incorporation into a partition set. Our experience is that by favoring elements in the true partition set first, one directs the algorithm to the correct solution thus improving the upper bound quickly, and that every branch at a shallow depth that does not choose one of these elements gets quickly trimmed as its absence increases the lower bound. So given the initial partition I^*, let $E = e_1 e_2 \ldots e_n$ be the consensus string for the sub-alignment M_{I^*} and $sim(h,E) = |\{\ i : M_{i,h} = e_i\}|$ be the number of columns where row h and sequence E have the same symbol. The permutation $\pi = (\pi_1, \pi_2, \ldots \pi_k)$ of $[1,k]$ that has the characteristics we desire is one for which $sim(\pi_1,E) \geq sim(\pi_2,E) \geq \ldots \geq (\pi_k,E)$. A simple valuation of $sim(h,E)$ for each h followed by a sort produces π in $O(kn + k\log k)$ time.

3.2 A First Lower Bound

We now develop a lower bound on the best possible score for any partition that is modeled by a leaf of the current vertex v in the search tree. As in our preliminary exercise above, let Y model the set for the current vertex v which we will suppose is at height h, and let Yc represent the complement of Y with respect to the set $\pi[h+1,k]$ of rows considered on the descent to v. Furthermore we'll use all the machinery developed for computing the $O(sn)$ In and Out quantities and their sorted orders η and ρ, save for complementary quantities $NotIn$ and $NotOut$ to be introduced below.

Our first lower bound concept is quite simple: in each column permit the selection of rows for the completion of Y that maximizes the score of that column. Surely permitting the selection of rows to be independent for each column creates an additional freedom not present in the real problem, and thus guarantees a lower bound on the true value of the best extension of Y.

So consider that there must be some symbol a giving rise to the maximum $|R_i^a \cap X|$ for the extension of Y to X, and some symbol b giving rise to the maximum $|R_i^b \cap X^c|$ for the extension of Yc to X^c. If symbol a gives the maximum in X, then the best score one can attain for the X-half is $NotOut_i^a = |R_i^a \cap Yc^c|$, the number of a's in rows not already allocated Yc, and one attains this by assigning all unassigned rows containing an a to X. Similarly if symbol b gives the consensus in X^c, then the best score one can get in the X^c-half is $NotIn_i^b = |R_i^b \cap Y|$, the number of b's in rows not already allocated to Y. So if the symbols a and b giving a maximum sum of the two halves are not the same then the best attainable score is $max_{a \neq b} NotIn_i^a + NotOut_i^b$. The other possibility is that the same symbol c is used in both halves in which case the score over both halves is just $|R_i^c|$. Thus it follows that the score of the best possible extension of Y in column i through the next h levels is:

$$Extend_i(Y) = max\{max_{a \neq b} NotIn_i^a(Y) + NotOut_i^b(Y), max_c |R_i^c|\}$$

As in the basic exercise one can maintain the current value of $NotIn_i^a(Y)$ and $NotOut_i^a(Y)$ and their orders, η_i and ρ_i in $O(n)$ time per search tree edge traversal, and with $O(sn)$ space. In addition in a preprocessing step prior to the start of the search one should compute the $O(sn)$ fixed quantities $Ceil_i^a = max_a |R_i^a|$. To compute $Extend_i(Y)$ one could consider every pair of symbols $a \neq b$ but with the orders η_i and ρ_i one can compute the maximum sum in $O(1)$ time. If η_i^1 does not equal ρ_i^1, then the maximum sum is clearly the sum of the two maxima for each half, i.e. $NotIn_i^{\eta_i^1} + NotOut_i^{\rho_i^1}$. On the otherhand, if the symbols giving the maximum in each half are the same, then the maximum sum is the better of the maximum in the first half and the second-maximum in the second half or the second-maximum in the first half and the maximum in the first half, i.e. $max\{NotIn_i^{\eta_i^1} + NotOut_i^{\rho_i^2}, NotIn_i^{\eta_i^2} + NotOut_i^{\rho_i^1}\}$. Then taking the maximum with $Ceil_i^a$ delivers the desired value. So in constant time one can compute $Extend_i(Y)$, and in $O(n)$ time our first lower bound:

$$LB1(Y) = \Sigma_i Extend_i(Y)$$

Computing this lower bound is as efficient as updating the structures needed for the basic exercise. But as we will see in the empirical results section it can eliminate the exploration of a significant portion of the search tree.

3.3 A Stronger Lower Bound

Computing a stronger lower bound requires further constraining the possibilities for extending each column, but not to the point of requiring that one make the same choices in every column, which reduces to the original problem. An intermediate is to consider the constraint that exactly g rows be added to Y in each column for each choice of g in $[0, h]$. For a given g each column has the freedom to pick whichever g rows it wishes to add to Y, but exactly g must be added. The result is thus clearly a lower bound on the best score achievable from Y.

Consider then the case where exactly g rows are to be added to Y. Suppose symbol a gives the best score in the X-half of the partition. Then this score is $Inpart_i^a(g) = min\{NotOut_i^a, In_i^a + g\}$ which is the smaller of the unrestricted case, $NotOut$, and adding exactly g unassigned rows containing a to the In_i^a already in Y. Similarly the best score obtainable in the X^c-half of the partition with symbol b as the maximizer, is $Outpart_i^a(g) = min\{NotIn_i^b, Out_i^b + (h-g)\}$. So the best score obtainable is either the maximum of the sum of the two terms for every choice of $a \neq b$, or if the same symbol gives the maximum in both halves, then the score would be $Ceil_i^a$ as in the simpler bound above. Thus it follows that the score of the best possible extension of Y with the addition of exactly g rows is:

$$Extend_i(Y, g) = max\{max_{a \neq b} Inpart_i^a(g)(Y) + OutPart_i^a(g)(Y), Ceil_i^a\}$$

Two obvious ways to compute this stronger lower bound at each vertex are as follows. First use the basic machinery to keep current values for $NotIn$, $NotOut$, In, and Out, but not the orders η and ρ, and compute $Extend_i(Y, g)$ in each column directly in $O(ks)$ time for a total of $O(ksn)$ time per

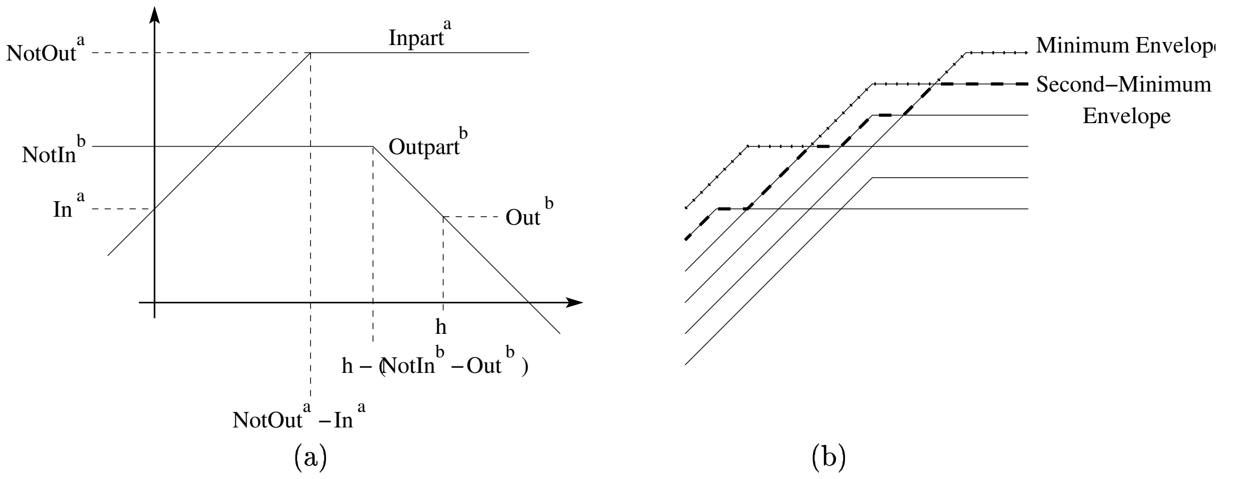


Figure 3: Anatomy of the *Inpart* and *Outpart* curves and an illustration of maximum and second-maximum envelopes.

vertex and $O(sn)$ space overall. The other alternative is to use the basic machinery to incrementally keep count of $Inpart(g)$ and $Outpart(g)$ for each $g \in [0, k]$ explicitly along with orders of each using $O(ksn)$ overall space. As for our first bound, this permits evaluation of $Extend_i(Y, g)$ in $O(1)$ time, implying the evaluation of the bound at each vertex in $O(kn)$ time. But we now show that with $O(sn)$ space, one can perform the evaluation of the bound in only $O((s + k)n)$ time.

As for the first lower bound we maintain the current value of $NotIn_i^a(Y)$ and $NotOut_i^a(Y)$ and their orders, η_i and ρ_i. We also maintain the current values of $In_i^a(Y)$ and $Out_i^a(Y)$, but not their orders, as in the basic exercise. Figure 3(a) illustrate that the function $Inpart_i^a(g)$, considering g to be unbounded, is a 2-piece affine curve where the left part is a line of slope 1, the right part is a line of slope 0, and the transition occurs at $g = NotOut_i^a - In_i^a$. Similarly, $Outpart_i^a(g)$, is a 2-piece affine curve where the left part has slope 0, the right part slope -1, and the transition occurs at $g = h - (NotIn_i^a - Out_i^a)$. Both curves are concave and so we know immediately that their maximum envelope can be computed in $O(s)$ time by our earlier work on sequence comparison with concave gap penalties [7]. The maximum envelope $Max_{Inpart_i}(g) = max_a Inpart_i^a(g)$ is the maximum value that one can obtain over all s *Inpart*-curves, for each choice of g. Figure 3(b) illustrates the maximum envelope of a set of curves.

We review this basic result for finding the maximum *Inpart*-envelope as we will need to extend it to finding the second-largest envelope, a new variation of this fundamental problem. Suppose that $\rho = (\rho^1, \rho^2, \ldots \rho^s)$ is the decreasing order of $NotOut$ where the column i and its subscript will be implicit. First, observe that for any two letters $a = \rho^t$ and $b = \rho^u$ where $t > u$, the curves for a and b intersect at most once, where the a curve is higher at the right, and the curves intersect at $g = NotOut^b - In^a$ if and only if $In^a < In^b$.

Starting at the right (large g) the curve for ρ^1 defines the maximum envelope and remains dominant until a curve lower in the order intersects it. If there are several such curves, then the one highest in the order ρ will begin to define the envelope immediately after intersecting the curve for ρ^1 as lower curves must intersect the ρ^1-curve at a lower value. Recursively, as one proceeds right to left, if the current curve defining the envelope is ρ^i, then it remains dominant until intersected by the next highest curve in the order ρ. Figure 4 sketches the $O(s)$ algorithm that traces the maximum envelope from right to left. One can record the maximum value for each $g \in [0, h]$ in $O(k)$ additional time.

A symmetric algorithm delivers the value of $Max_{Outpart_i}(g)$ for every choice of $g \in [0, h]$ in $O(s+k)$ time. But as for our first lower bound, to efficiently evaluate $Extend_i(Y, g)$ for a specific value of g it is necessary to not only know the maximum value, and the symbol attaining it, but if the symbols for the two parts are the same, one must know the second largest value. Formally, the second-maximum envelope $Max^2_{Inpart_i}(g) = max_b\{Inpart_i^b(g) : \exists a \neq b \; Inpart_i^b(g) \leq Inpart_i^a(g)\}$. Certainly, it is easy

```
"$\rho^1$ gives the maximum for $g$ arbitrarily large"
$a \leftarrow \rho^1$
for $t \leftarrow 2, 3, \ldots s$ do
     if $In^{\rho^t} > In^a$ then
       { "$\rho^t$ takes over as the maximum for $g \leq NotOut^{\rho^t} - In^a$"
         $a \leftarrow \rho^t$
       }
```

Figure 4: Computing a maximum envelope in $O(s)$ time.

to note the symbol a giving the maximum envelope value for g as the maximum values are computed. Given that we can get the second largest values for all choices of g in $O(k+s)$ time then clearly, as in the lower bound, the *Extend* values can be compute in $O(1)$ time per point. So it remains to show how to efficiently compute the second-maximum envelope.

Starting at the right (large g) the curve for ρ^2 defines the second-maximum envelope and remains dominant until either (a) the curve for the first maximum intersects and moves below it, or (b) a curve lower in the order intersects it. As before if there are several such curves for case (b), then the one highest in the order will intersect first. But it is also true that if case (a) occurs, then it always occurs before case (b) as the maximum curve pierces the slope 0 part of the second-maximum, and the lower curve pierces the slope 1 part. So one checks for case (a) and if it occurs, then the current maximum and second-maximum switch at the point of intersection as one scans from right-to-left. Otherwise case (b) occurs and one replaces the second-maximum with the intersection curve. Figure 5 outlines a complete algorithm embodying this logic and Figure 3(b) shows an example of a second-maximum envelope to aid ones intuition.

```
"$\rho^2$ gives the $2^{nd}$-maximum for $g$ arbitrarily large
$a \leftarrow \rho^1$
$b \leftarrow \rho^2$
for $t \leftarrow 3, 4, \ldots s$ do
  {  if $In^b > In^a$ then
       { "$a$ becomes the $2^{nd}$ maximum for $g \leq NotOut^b - In^a$"
         $a \leftrightarrow b$
       }
     if $In^{\rho^t} > In^b$ then
       { "$\rho^t$ takes over as the $2^{nd}$-maximum for $g \leq NotOut^{\rho^t} - In^b$"
         $b \leftarrow \rho^t$
       }
  }
```

Figure 5: Computing the 2^{nd}-maximum envelope in $O(s)$ time.

4 Empirical Results

A data set generator was built to test and evaluate the algorithm. The generator produces a random progenitor sequence of length 500 over the DNA alphabet, introduces $500c$ differences to produce the second progenitor, and then produces k copies of each progenitor perturbed by $500e$ differences. The $2k$ sequences are then initially aligned by pairwise comparisons and then submitted to *realigner* to produce the multi-alignment input to our algorithms. We will study the algorithms only with respect to the three parameters k, c, and e. In a more extensive set of trials the length of the sequences, the number of classes, and the number of copies in each class should be studied.

First we examined the relative performance of the two branch and bound algorithms with respect to how much of the search tree they pruned and how much CPU time they took to solve a particular instance. Table 1 gives tables showing the percent of the total search tree explored and the time taken

	e		
k	5%	10%	15%
5	2.54%	5.26%	9.44%
10	.0140%	.0266%	.0678%
15	.000029%	.000145%	.000483%

Percent of tree explored (Bound 1)

	e		
k	5%	10%	15%
5	25	53	96
10	147	279	711
15	307	1566	5183

Vertices explored (Bound 1)

	e		
k	5%	10%	15%
5	.70%	1.31%	2.30%
10	.0016%	.0022%	.0055%
15	.0000026%	.0000038%	.0000149%

Percent of tree explored (Bound 2)

	e		
k	5%	10%	15%
5	7	13	23
10	16	22	57
15	28	41	160

Vertices explored (Bound 2)

	e		
k	5%	10%	15%
5	2.9	3.8	5.1
10	16.2	21.4	33.8
15	45.3	79.9	189.3

Time Taken (Bound 1)

	e		
k	5%	10%	15%
5	4.0	4.9	6.7
10	17.3	21.5	33.2
15	50.5	60.2	108.3

Time Taken (Bound 2)

Table 1: Percent of the search tree explored, absolute number of vertices of the search tree explored, and time taken (in seconds) by the B&B algorithm as a function of the bound and the number (k) and error rate (e) of the copies.

over 100 trials at each of several settings of k and e. The parameter c was held constant at 5%, a differential large enough to guarantee that the algorithm was always correctly separating copies as will be seen below. One sees that significantly smaller number of vertices and an even smaller percentage of the search tree is explored, with the stronger bound performing a factor of roughly 4, 10, and 40 times better for $k = 5$, 10, and 15 respectively. Despite this, because the time for computing the stronger bound is greater, it generally takes more time to solve a problem with this bound, unless the error rate is particularly high. Finally, one should observe that the number of vertices explored is still growing exponentially in both k and e, so like most branch-and-bound algorithms, we have significantly ameliorated the exponential nature of the problem but not eliminated it.

Next we consider how effectively our notion of separability correctly identifies the two subclasses. That is, if one computes a set X^{opt} that produces the optimum partition score $opt(M) = min_X \, part(M, X)$, how often is X^{opt} actually the desired answer? Recall that the design rationale was that the freedom to separate in the case of just noise does not induce much of a drop between $cons(M)$ and $opt(M)$, but when there are, say t discriminating columns, then a separation along the lines of the two classes results in a drop in score of $tk(1-2e)$ where e is the level of noise and k is the number of copies in each class. So the question is, "When is this drop large enough that it is distinct from the improvement one can obtain just by chance in the presence of only noise?"

We ran 1000 trials for each of several choices of k and e, measuring $cons(M) - opt(M)$ in the case where only noise was present (that is, there was only one class of sequences) and there were 500 columns. Table 2 gives the average and maximum value obtained for each choice of parameter settings. The average and maximum appear to grow slightly super-linearly in e and to increase quite slowly in k. The maximum value gives a rough indication of the largest value one would see with probability .001. Using the expected drop of $tk(1 - 2e)$ in the case of t discriminating columns, we

	e		
k	5%	10%	15%
5	5.03	11.41	19.29
10	6.14	13.01	20.94
15	6.71	13.73	21.86

Average Difference

	e		
k	5%	10%	15%
5	9	18	27
10	11	20	29
15	12	21	29

Maximum Difference

	e		
k	5%	10%	15%
5	2	5	9
10	2	3	5
15	1	2	3

Predicted

	e		
k	5%	10%	15%
5	4	5	9
10	4	5	6
15	4	4	5

Observed

Table 2: The first row of tables summarize the average and maximum value of $cons(M) - opt(M)$ when M is a matrix of $2k$ sequences of 500 letters that differ from a common ancestral sequence by e percent at random. The table at the lower left gives the predicted number of discriminating columns need to accurately separate two classes based on the table of maximums above. The table at the lower right gives the observed number of discriminating columns needed to tease the subclasses apart correctly 999 times out of 1000.

predict in the lower left table of Table 2 the number of columns needed to correctly separate with probability .999. We then show the actual number from empirical trials. The agreement is reasonable with the prediction being overly optimistic. It is especially interesting in the low error rate case ($e =$ 5%) that as k increases, the number of discriminating columns needed to correctly separate does not seem to improve. Nonetheless, the final table shows that one can separate classes that are 1% or more different in the presence of noise up to 10%.

Acknowledgements: The author wishes to thank Marie France Sagot for several conversations on the measure of separability and the problem formulation in the early going.

References

[1] Gusfield, D. *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Chapter 17, 1997.

[2] Huang, X. and Madan A. CAP3: A DNA Sequence Assembly Program, *Genome Research* **9**, 9 (1999), 868-877.

[3] Gordon, D., Abajian, C., and Green, P. Consed: A Graphical Tool for Sequence Finishing, *Genome Research*, 8(3):195–202, 1998.

[4] Myers, E. Toward Simplifying and Accurately Fragment Assembly, *J. of Computational Biology* 2(2),275–290, 1995.

[5] Horowitz, E., Sahni, S. and Rajasekaran, S. *Computer Algorithms / C++*, Computer Science Press, Chapter 8, 1995.

[6] Anson, E., and Myers, E. ReAligner: A Program for Refining DNA Sequence Multi-Alignments, *J. of Computational Biology* 4(3),369–383, 1997.

[7] Miller, W., and Myers, E. Sequence Comparison with Concave Weighting Functions, *Bull. of Mathematical Biology* 50(2),97–120, 1988.