# ReAligner: A Program for Refining DNA Sequence Multi-Alignments

Eric L. Anson[*] and Eugene W. Myers[**]

**Abstract.** We present a round-robin realignment algorithm that improves a potentially crude initial alignment of an assembled collection of DNA sequence fragments, as might, for example, be output by a typical fragment assembly program. The algorithm uses a weighted combination of two scoring schemes to achieve superior multi-alignments, and employs a banded dynamic programming variation to achieve a running time that is linear in the amount of sequence in the data set. We demonstrate that the algorithm improves upon the alignments produced by other assembly programs in a series of empirical experiments on simulated data. Finally, we present a pair of programs embodying the algorithms that are available from the Web site `ftp://ftp.cs.arizona.edu/realigner`.

**Keywords:** fragment assembly, round-robin realignment, multiple alignment, consensus score/sequence.

## 1 Introduction

The final step in shotgun DNA sequencing is the multi-alignment of fragment sequences in regions where three or more fragments overlap. This multi-alignment pinpoints errors in individual fragments and results in a consensus sequence representing the final reconstruction of the source DNA. In earlier steps, the fragments were compared to determine all pairwise overlaps and a layout specifying the relative positions of fragments with respect to each other was determined from these overlaps. Given a layout of the fragments, it is then a simple matter to produce a crude multi-alignment from the pairwise alignments of the sequences as the differences between them is rarely greater than 10%. This initial multi-alignment is globally correct but is locally non-optimal. The realignment problem we consider here is to refine any multi-alignment into one that is optimal or near-optimal while preserving its global structure.

Consider sequences $S_1, S_2, \ldots, S_k$ over alphabet $\Sigma$ where $S_i = s^i_1 s^i_2 \ldots s^i_{n_i}$ has length $n_i$. An alignment of the $k$ sequences is a $k$ by $l \geq max_i\{n_i\}$ matrix, $A = (a_{ij})$ of symbols over alphabet $\Sigma \cup \{\text{-}\}$ such that $dash(a_{i?}) = S_i$ and $a_{?j} \neq \text{-}^k$ where $dash$ is the homomorphism that removes dashes, $a_{i?}$ denotes the $i^{th}$ row of $A$, and $\text{-}^k$ is a column of $k$ dashes. In plain words, a multi-alignment is a rectangular array such that

[*] Dept. of Computer Science, University of Arizona, Tucson, AZ 85721 (e-mail: anson@cs.arizona.edu).

[**] Dept. of Computer Science, University of Arizona, Tucson, AZ 85721 (e-mail: gene@cs.arizona.edu).

removing dashes from row $i$ leaves sequence $S_i$ for each $i \in [1, k]$, and every column has at least one sequence symbol in it. Given a scoring function $\delta : (\Sigma \cup \{\text{-}\})^k \mapsto \Re$ that assigns a score to any column, the optimization problem is to determine a multi-alignment of minimal score where the score $\delta(A)$ of an alignment $A$ is the sum of the scores of its columns, $\sum_{j=1}^{l} \delta(a_{?j})$.

In the context of DNA sequencing, the alphabet $\Sigma = \{a, c, g, t\}$ and we believe the relevant objective is to minimize the consensus score which is:

$$\delta(a_1, a_2, \ldots, a_k) = \min_{x \in \Sigma \cup \{\text{-}\}} \mid \{a_i : a_i \neq x\} \mid \tag{1}$$

A consensus symbol for a column, $c(a_1, a_2, \ldots, a_k)$, is a choice of $x$ minimizing (1) above. Thus a consensus sequence $c(A)$ for an alignment $A$ is $dash(c(a_{?1})c(a_{?2}) \ldots c(a_{?l}))$. Choosing consensus as the target is an appeal to parsimony as it minimizes the sum of the differences of each fragment from the consensus sequence. In other words, it explains the data with minimum error.

In DNA sequencing our problem is not determining a global multi-alignment amongst all the fragments, as described above, but rather of multi-aligning the fragments in the regions where they are indicated to overlap in the layout produced in the previous stage of the assembly process. In such a *layout alignment*, dashes which occur before of after all the symbols in a sequence should not affect the consensus score. These dashes are called end-gaps and are denoted in a layout alignment by blanks as illustrated in Figure 1. Ignoring end-gaps when applying $\delta$ to a column has the required effect.

```
cctggt-acgta-cact-tgt
     tcacgtatccctctgttaga
         gta-ccctctgttagaaagctcacgt
                            ctcacttagttctctg-t
                             tcactt-gttctgtg-tag
                                  t-gttccgtgctagtagcta
-----------------------------------------------------
CCTGGTCACGTA-CCCTCTGTTAGAAAGCTCACTT-GTTCTGTG-TAGTAGCTA
```

**Fig. 1.** A sample layout alignment and consensus sequence.

The layout phase in DNA sequencing determines where each pair of sequences overlap. This same information can be gleaned from a layout by noticing that two sequences $S_i$ and $S_j$ overlap exacly in the columns of $A$ which do not have an endgap in either row $i$ or row $j$. We say an alignment $A$ *respects the layout* to precision $\epsilon$ if the overlaps implied by $A$ differ from the overlaps implied by the layout by less than $O(\epsilon)$. Formally, for every pair of integers $1 \leq i, j \leq k$ let $L_{ij}^{<}$ and $L_{ij}^{>}$ be the positions in $S_i$ of the first and last symbols which overlap with $S_j$ according to the layout. If the fragments do not overlap than let $L_{ij}^{<} = L_{ij}^{>} = n_i + 1$ if $S_i$ appears before $S_j$ and $L_{ij}^{<} = L_{ij}^{>} = 0$ otherwise. Similarly define $A_{ij}^{<}$ and $Ab_{ij}^{>}$ to be the positions of the first and last symbols of the overlap according to alignment $A$.

**Definition 1**: An alignment $A$ respects the layout to precision $\epsilon$ if and only if $\forall\ 1 \leq ij \leq k$,

$$L_{ij}^{<} \approx_\epsilon A_{ij}^{<} \qquad \text{and} \qquad L_{ij}^{>} \approx_\epsilon A_{ij}^{>}$$

where $x \approx_\epsilon y$ iff $x \in [y - O(\epsilon), y + O(\epsilon)]$.

Now, given a layout, our problem is to determine a multi-alignment $A$ such that $\delta(A)$ is minimal over all alignments respecting the layout to precision $\epsilon$. Note that the constraint that the layout be respected is essential, as the multi-alignment which places the sequences end to end has score 0 as end-gaps are not penalized.

In this paper, we present a round-robin algorithm and encompassing software tool for optimizing an initial, low-quality layout alignment. That is, the software attempts to produce a minimal cost alignment that respects the layout implied by the submitted alignment. We call this process *realignment* and we call our software tool *ReAligner*. We present empirical results that show that our round-robin approach outperforms other software solutions, frequently producing optimal results when $\epsilon$ is 10% or less. In addition, our algorithm uses a banding idea in its dynamic programming component which effectively renders the method linear in the sum of the lengths of the fragments. For example, the realignment of a multi-alignment involving 1,200,000 basepairs of fragment data took 77.5 seconds on a Dec AlphaStation 200 4/233.

## 2 Background

While the multiple alignment problem is NP complete in the number of sequences, there is a well-known $O(\overline{n}^k)$ dynamic programming algorithm for finding a global alignment among $k$ sequences of average length $\overline{n}$. This algorithm extends the basic pairwise algorithm in the obvious way by filling in an $(n_1+1) \times (n_2+1) \times \cdots \times (n_k+1)$ matrix $C$ such that $C[p_1, p_2, \cdots p_k]$ is the best multi-alignment of the first $p_i$ symbols of each sequence $S_i$. This algorithm can be specialized to our layout alignment problem, by restricting the computation to those matrix entries that model a prefix of an alignment respecting the layout in question. This can be shown to reduce the complexity of the computation to $O(rn_{max}^{c-1})$ worst-case time where $n_{max}$ is the maximum length of any sequence in the assembly, $r$ is the length of the reconstructed sequence, and $c$ is the maximum number of simultaneously overlapping fragments in the layout. Despite this improved complexity, the algorithm is still impractical as the maximum fragment length is typically 500 and $c$ is typically 10. Thus one must resort to heuristic approximation algorithms.

One commonly employed heuristic, *progressive alignment* works as follows. Two of the $k$ sequences are aligned together, and the resulting pairwise alignment replaces the two sequences. This gives an alignment problem with $k - 1$ "sequences", one of which is a sequence of aligned pairs (i.e., each of its "symbols" is a column of the pairwise alignment). Two of those sequences are chosen, aligned together, and replaced by that alignment, until only a $k$-way alignment remains. What makes this possible is that the dynamic programming algorithm for aligning two sequences can be made to work when one or both of the input sequences is itself a multi-alignment. This strategy can be traced to a 1984 paper of Waterman and Perlwitz

[10], and has been implemented, with numerous variations on the basic theme, by many investigators, e.g., [5, 9, 4, 6]. The most common variations require either $O(k^3\overline{n}^2)$ or $O(k^2\overline{n}^2)$ time.

In the context of DNA sequencing, every pairwise alignment between sequences has been computed in an earlier phase, as well as a set of $k-1$ pairwise overlaps that specify the layout by positioning the sequences relative to one another. An initial, but crude layout alignment suitable for input to *ReAligner* can be obtained very rapidly by the following simplification of progressive alignment. The sequences are progressively aligned in any order along the $k-1$ overlaps specifying the layout. A progressive alignment across the overlap between original sequences $A$ and $B$, does not perform a *de novo* comparison between the two multi-alignments containing $A$ and $B$, but rather simply aligns these two multi-alignments according to the already computed alignment between $A$ and $B$. Overall, the computation of this initial layout alignment takes only $O(k\overline{n}) = O(N)$ time where $N = \Sigma_i n_i$ is the total number of basepairs in the layout. Such a simplification would give exceedingly poor results in contexts such as protein multi-alignment, where the sequences generally share little identity. However, in our context, the sequencing error rate, $\epsilon$, is less than 10%, so the alignment obtained is a good initial estimate. Indeed, it is globally optimal in that only local corrections are needed to arrive at the best possible alignment.

Over a period of several years, we have developed and experimented with various heuristics for refining an initial, low-quality layout alignment. One of our first attempts was a *window-sweep* approach where the alignment was locally improved inside a small window as it swept from left-to-right. There are many possible variations depending on what one decides to do inside a window. All our designs involved recursively applying progressive alignment to the subsequences in the window. The most efficient variation used the existing pairwise alignments and simply tried to choose a merge order that was better suited to the subsequences in the window. We also experimented with recomputing the pairwise alignments, and using a variety of scoring schemes. In all variations, we did not compute an alignment between multi-alignments during the progressive merges as in the general algorithm, but used the simplified form described in the paragraph above. While these approaches improved the input layout alignment, the results were not as good as those obtained in our subsequent work.

Our next attempt involved using *hidden Markov models* [8, 1], an approach that has had considerable success in the protein domain. A hidden Markov model is a probablistic model which can be viewed as either generating or matching sequences with a probability determined by the model. The idea is to arrive at a model which is the most likely to have generated the observed sequences. An initial model is built from the initial layout, and then a gradient descent algorithm refines the model. Each refinement step consists of first finding the state occupancy of each sequence in the current model, and then readjusting the emission probabilities of each state of the model according to these occupancies. The resulting algorithm was very slow in practice but gave better results than the window-sweep heuristics.

Finally, we arrived at a *round-robin* algorithm which we present in detail in this paper. Like the hidden Markov model, it also has the virtue of using the entire multi-alignment to derive each refinement. The iterative technique was first introduced in the context of aligning sequences of proteins [2]. Our realization of the round-robin algorithm iteratively aligns each sequence with the multiple alignment of the remaining sequences. Thus the algorithm is comparatively efficient since it is just a series of pairwise alignments where one sequence is ordinary and the other is a multi-alignment. The process repeatedly realigns the sequences in some order until the multi-alignment stabilizes. Iterative techniques have been used to align sequences of proteins, where they generally do not perform as well as progressive alignment or hidden Markov model approaches. But surprisingly in the context of DNA sequencing the method gives the best results with the greatest efficiency. We are the first to apply the round-robin technique to DNA sequence layout alignments. Our novel contributions consist of the selection of scoring scheme for realignment and a banding approach to the dynamic programming computation that effectively gives us an $O(N)$ algorithm.

## 3 The Algorithm

### 3.1 The Round-Robin Paradigm

The general round-robin paradigm consists of a series of realignment steps. In a given realignment step, the multi-alignment is partitioned into two parts, the two subalignments are pairwise compared, and then merged according to the results of the comparison. Some of the dimensions of variation on this idea are (1) the nature of the partition, (2) the method use to select the partition, and (3) the scoring function used to pairwise compare the two subalignments.

An alignment $B$ is a *subalignment* of alignment $A$ if $B$ can be obtained from $A$ by deleting a number of rows and then removing any columns that consist only of dashes. In the partition step, a set of rows $P \subset [1,k]$ of $A$ is selected and the subalignments $A_P$ and $A_Q$ result, where $A_P$ is the subalignment consisting of the rows in $P$ and $A_Q$ is the subalignment over the remaining rows $Q = [1,k] - P$. Suppose $p$ is the size of $P$ and $q = k - p$ is the size of $Q$. Next observe that $A_P$ is a sequence of columns each of which may be thought of as meta-symbol in the alphabet $\Sigma_p = (\Sigma \cup \{\text{-}\})^p - \{\text{-}^p\}$. Similarly, $A_Q$ may be thought of as a sequence over alphabet $\Sigma_q$. Thus the comparison step requires a function $\delta_2 : (\Sigma_p \cup \{\text{-}^p\}) \times (\Sigma_q \cup \{\text{-}^q\}) \mapsto \Re$ that scores pairwise alignments of the sequences of meta-symbols in $A_P$ and $A_Q$. If $\mathcal{X} \equiv [x_1, x_2, \ldots, x_p]$ is a symbol in $\Sigma_p$ and $\mathcal{Y} \equiv [y_1, y_2, \ldots, y_q]$ is a symbol in $\Sigma_q$, then $\delta_2(\mathcal{X},\mathcal{Y})$ is the score of aligning $\mathcal{X}$ and $\mathcal{Y}$, $\delta_2(\mathcal{X},\text{-}^q)$ is the score of leaving $\mathcal{X}$ unaligned, and $\delta_2(\text{-}^p,\mathcal{Y})$ is the score of leaving $\mathcal{Y}$ unaligned. If the overall objective is to produce an optimal multi-alignment with respect to column scoring function $\delta$, then a common choice of $\delta_2$ is to define it so that:

$$\begin{aligned}
\delta_2(\mathcal{X},\mathcal{Y}) &= \delta(x_1, x_2, \ldots, x_p, y_1, y_2, \ldots, y_q)\\
\delta_2(\mathcal{X},\text{-}^q) &= \delta(x_1, x_2, \ldots, x_p, \text{-}, \text{-}, \ldots, \text{-})\\
\delta_2(\text{-}^p,\mathcal{Y}) &= \delta(\text{-}, \text{-}, \ldots, \text{-}, y_1, y_2, \ldots, y_q)
\end{aligned}$$

That is, the score of aligning two columns $\overline{\mathcal{X}} \in \Sigma_p \cup \{\text{-}^p\}$ and $\overline{\mathcal{Y}} \in \Sigma_q \cup \{\text{-}^q\}$ is exactly the score $\delta$ assigns to the column $\overline{\mathcal{X}} \cdot \overline{\mathcal{Y}}$ of $p + q = k$ symbols obtained by concatenating the two columns. In brief, $\delta_2(\overline{\mathcal{X}},\overline{\mathcal{Y}}) = \delta(\overline{\mathcal{X}} \cdot \overline{\mathcal{Y}})$. The final merge step consists of optimally aligning $A_P$ and $A_Q$ with respect to $\delta_2$ and then simply returning conceptually to viewing the meta-symbols of the two subalignments as columns of symbols.

For our work, we chose the following instantiation of the general paradigm. The nature of the partition is to choose a single sequence and compare it against a multi-alignment consisting of all the rest, i.e., $p = 1$ and $q = k - 1$. The method for selecting the partition is to iteratively choose each sequence in turn. Thus in outline, our realignment algorithm is as follows:

```
ReAligner(A: a layout alignment of $S_1, S_2, \ldots, S_k$)
1. repeat
2.   for $i = 1$ to $k$ do
3.     { $Q \leftarrow [1,k] - \{i\}$
4.       Compute an alignment $a$ such that $\delta_2(S_i \overset{a}{\sim} A_Q)$ is minimal over
              all $a$ such that $S_i \,||^a\, A_Q$ respects the layout to precision $\epsilon$.
5.       $A \leftarrow S_i \,||^a\, A_Q$
6.     }
7. until $\delta(A)$ does not decrease in an execution of Steps 2-6.
```

The notation $S_i \overset{a}{\sim} A_Q$ simply asserts that $a$ is an alignment (correspondence of symbols) between $S_i$ and $A_Q$ when both are viewed as sequences of symbols. The notation $S_i \,||^a\, A_Q$ denotes the multi-alignment that results if the columns of $S_i$ and $A_Q$ are aligned according to alignment $a$. Note that because we are dealing with layout alignments we must restrict ourselves to considering in Step 4 only alignments between $S_i$ and $A_Q$ that if subsequently merged in Step 5 give an alignment that respects the layout in question. Our banded alignment approach described in Section 3.3 will effectively guarantee this.

A somewhat awkward problem arising in the context of layout alignments is that the value of $\delta_2$ becomes position dependent because end-gap dashes are ignored whereas inter-sequence dashes are not. For example, if one were to define $\delta_2(x, \text{-}^{k-1})$ to be $\delta(x, \text{-}, \ldots, \text{-})$ where $\delta$ is the consensus scoring function, then the score of inserting the symbol $x$ of $S_i$ depends on which column of $A_Q$, the symbol is being inserted after. The insertion will have the effect of placing the $k - 1$ dashes in question right after the indicated column of $A_Q$. Given this context it is easy to determine which dashes will be end-gap and which will not. In our example, the score of inserting $x$ is 1 if at least one dash will not be in an end-gap, and 0 otherwise. Henceforward we make the simplifying assumption that the underlying comparison algorithm will distinguish end-gap dashes as it proceeds, and remove them from any meta-symbol given to $\delta_2$. So in our example, if a comparator requires the value of inserting $x$ in a context where all dashes introduced into the other subalignment will be end-gap, it scores inserting $x$ as $\delta_2(x, \text{-}^0) = \delta(x) = 0$.

## 3.2 The Scoring Function

We experimented with two distinct choices for $\delta_2$ presented below and eventually determined that an evenly weighted combination of the two gave the best results in practice. In what follows, we will assume that $S$ is the sequence that has been selected to be realigned, and that $B$ is the remaining layout subalignment of $A$.

Our first choice of pairwise scoring function, $\delta_c$, is based on the consensus symbols of $B$ and is defined as follows:

$$\delta_c(x, \mathcal{X}) = \begin{cases} 0 \text{ if } x \in \overline{c}(\mathcal{X}) \text{ or } \mathcal{X} = \epsilon \\ 1 \text{ otherwise} \end{cases} \tag{2}$$

where $\overline{c}(\mathcal{X})$ is the *set* of consensus elements of column $\mathcal{X}$. Note carefully that there may be more than one symbol that minimizes Equation 1 of the introduction, and here we define $\overline{c}$ as the function returning the set of all these symbols, whereas $c$ of the introduction arbitrarily returns one of these. We similarly define the *full consensus* of $A$ as $\overline{c}(A) = \mathit{dash}(\overline{c}(a_{?1}), \overline{c}(a_{?2}), \cdots \overline{c}(a_{?l}))$ where *dash* removes a set from the resulting sequence of sets only if it is the singleton set {-}.

First observe that we are effectively comparing $S$ to the full consensus $\overline{c}(B)$ under the standard edit-distance scoring scheme, *diff*, that minimizes the number of mismatches, insertions, and deletions in an alignment. Formally, this observation is:

$$\delta_c(S \overset{a}{\sim} B) = \mathit{diff}(S \overset{a}{\sim} \overline{c}(B)) \tag{3}$$

where one considers a symbol and a set containing that symbol as a match, and where one does not count end-gaps as differences. For example, in Figure 2, we illustrate the selection of the third sequence in the layout of Figure 1 as $S$ and the resulting full consensus $\overline{c}(B)$ that it gets compared against. Note in this example that $A$ breaks into two parts when $S$ is removed. Thus $B$ consists of two sequences that are relinked by alignment with $S$ as illustrated in the figure. The columns $\mathcal{X}$ of $B$ for which $\overline{c}(\mathcal{X})$ is a set are denoted by placing the members of the set between square braces.

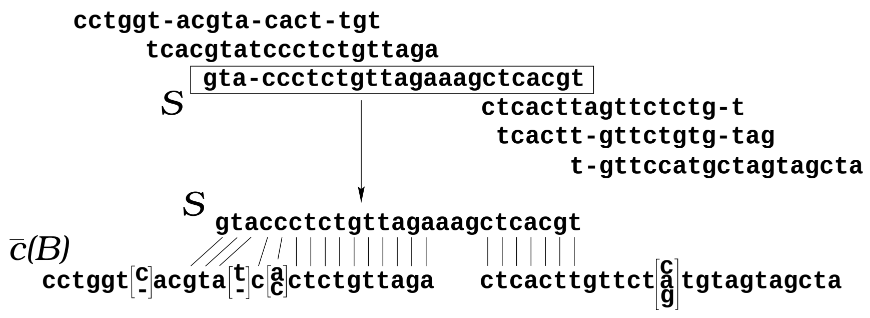

cctggt-acgta-cact-tgt
tcacgtatccctctgttaga
gta-ccctctgttagaaagctcacgt
S
ctcacttagttctctg-t
tcactt-gttctgtg-tag
t-gttccatgctagtagcta
S
gtaccctctgttagaaagctcacgt
c̄(B)
cctggt[c -]acgta[t -]c[a c]ctctgttaga
ctcacttgttct[c a g]tgtagtagcta

**Fig. 2.** Sample round-robin consensus comparison.

From the definition of $\delta_c$, it follows immediately that $\delta_c(x, \mathcal{X}) = \delta(x \cdot \mathcal{X}) - \delta(\mathcal{X})$. Applying this to each column of an alignment leads directly to the following lemma.

**Lemma** 1: For all pairwise alignments $a$ between $S$ and $B$,

$$\delta(S \parallel^a B) = \delta(B) + \delta_c(S \overset{a}{\sim} B) \tag{4}$$

That is, the consensus score of the alignment obtained by aligning $S$ with $B$ according to $a$ is the sum of the consensus score of $B$ and the $\delta_c$ score of $a$. A consequence of this lemma is that the consensus score of $A$ cannot increase with any application of Steps 4 and 5. To see this, observe that if the alignment between $S$ and $B$ implied by $A$ is, say $b$, then $\delta(A) = \delta(S \parallel^b B) = \delta(B) + \delta_c(b)$ by Lemma 1. In Step 4, an optimal alignment, say $a$ is found, and by definition $\delta_c(a) \leq \delta_c(b)$. Thus the alignment $S \parallel^a B$ formed in Step 5 has a lesser or equal score to that of $A$ by another application of Lemma 1. Thus, using $\delta_c$, our round-robin algorithm produces progressively better alignments until there is no change in a cycle through all $k$ sequences.

Another property that follows from Lemma 1 is that minimizing $\delta_c(S \overset{a}{\sim} B)$ is equivalent to minimizing $\delta(S \parallel^a B)$ as $\delta(B)$ is constant. Thus using $\delta_c$ is exactly equivalent to using $\delta_2(\overline{\mathcal{X}}, \overline{\mathcal{Y}}) = \delta(\overline{\mathcal{X} \cdot \mathcal{Y}})$ as discussed in Subsection 3.1. The difference is that $\delta_c$ can be computed in $O(1)$ time versus $O(|\mathcal{X}|) = O(k)$ time for the standard choice of $\delta_2$. Doing so requires computing $\overline{c}(A)$ at the start of the computation, and then incrementally computing $\overline{c}(B)$ and the changes to $\overline{c}(A)$ as the computation progresses. So from one point of view $\delta_c$ is simply a more efficiently computable formulation of the standard comparison objective function.

A final property of $\delta_c$ is that in the final layout alignment, call it $A^\star$, produced by the round-robin algorithm, every sequence is optimally aligned, under the *diff* measure, with the consensus $\overline{c}(A^\star)$ of $A^\star$. Suppose that $A^\star = B \parallel^a S$ for an arbitrary choice of sequence $S$ and let $a\!\uparrow$ be the alignment between $S$ and $A^\star$ that aligns every symbol of $S$ with the copy of $S$ in the multi-alignment $A^\star$. By the property of monotone improvement it must be that *diff*$(S \overset{a}{\sim} B)$ is optimal. We claim that *diff*$(S \overset{a\uparrow}{\sim} \overline{c}(A^\star))$ is also optimal. We term a column $x \cdot \mathcal{X}$ of $A^\star$ where $x \in S$ an *x-pivot* iff $x = \overline{c}(x \cdot \mathcal{X}) - \overline{c}(\mathcal{X})$, i.e., $x$ is a consensus symbol for the column only if it is in the column. For any alignment $b$ between $S$ and $A^\star$ it then follows by the definition of $\delta_c$ that:

$$\mathit{diff}(S \overset{b}{\sim} \overline{c}(A^\star)) = \mathit{diff}(S \overset{b\downarrow}{\sim} \overline{c}(B)) - No_{pivot}(S \overset{b}{\sim} \overline{c}(A^\star))$$

where $No_{pivot}(S \overset{b}{\sim} \overline{c}(A^\star))$ is the number of $x$-pivot columns in $A^\star$ aligned by $b$ with an $x$ in $S$, and $b\!\downarrow$ is the subalignment obtained by deleting the copy of $S$ in $A^\star$ from the multi-alignment $S \parallel^b A^\star$. Now $b = a\!\uparrow$ minimizes *diff*$(S \overset{b\downarrow = a}{\sim} \overline{c}(B))$ and maximizes $No_{pivot}(S \overset{b}{\sim} \overline{c}(A^\star))$, thus *diff*$(S \overset{a\uparrow}{\sim} \overline{c}(A^\star))$ is minimal over all alignments between $S$ and $A^\star$.

Our round-robin algorithm with $\delta_c$ is superior to our earlier window-based and hidden Markov model designs. Even so, it does not always give optimal results as illustrated in Figure 3. The alignment at left is not realigned into the optimal alignment at right if the pairwise alignment chosen in Step 4 happens to be

```
acct
at-t
```

when $S =$ `acct` is the third sequence and $\overline{c}(B) =$ `att`. In some sense, the problem is that the two `c`'s in the second column are not represented in the consensus `att`.

In an attempt to remedy the preceding situation, we then tried a scoring function which fractionally scored aligning a symbol of $S$ according to the content of a column of $B$ instead of assigning a 0-1 score based on the consensus of the column. Let $\delta_a$ be defined as follows:

$$\delta_a(x, (a_1, a_2, \cdots, a_n)) = \begin{cases} |\{a_i : a_i \neq x\}|/n & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases} \qquad (5)$$

```
  a c t - t            a c t - t
  a c t - t            a c t - t
  a - c c t            a c c - t
  a - c - t            a c - - t
  a - t - t            a - t - t
  a - t - t            a - t - t

 Score = 5            Score = 4
```

**Fig. 3.** An alignment (left) not improved to its optimum (right) by round-robin with $\delta_c$.

That is, the score for aligning a symbol $x$ of $S$ with a column of $B$ is equal to the fraction of symbols in the column not equal to $x$.

As shown in Section 4 on experimental results, it is usually the case that the round-robin algorithm using $\delta_a$ produces better results than when using $\delta_c$. However there are cases where the scoring scheme $\delta_c$ produces better results than $\delta_a$. Figure 4 gives an example where $\delta_a$ fails to realign the alignment at left into the optimal one at right (whereas $\delta_c$ does). To see why, observe that for any choice of $S$ as one of the six sequences, the best alignment between $S$ and the resulting $B$ is always the existing alignment at a score of $0 + 3/5 + 3/5 + 0 = 6/5$. The cost of aligning one of the last three sequences with its $B$ in a manner which moves its `c` to the left, is $0 + 1 + 2/5 + 0 = 7/5$. Thus the alignment at left does not change under the application of round-robin with $\delta_a$.

```
  g a c t            g a c t
  g a c t            g a c t
  g a c t            g a c t
  g c - t            g - c t
  g c - t            g - c t
  g c - t            g - c t

 Score = 6          Score = 3
```

**Fig. 4.** An alignment (left) not improved to its optimum (right) by round-robin with $\delta_a$.

In using $\delta_a$ we lose some of the properties of $\delta_c$ discussed above. Namely, the consensus scores of the multi-alignments obtained through each iteration of the algorithm do not necessarily form a nondecreasing sequence, and the fragments are not necessarily optimaly aligned with the consensus that results. Empirical evidence indicates that it is extremely rare for the consensus score to increase but it can happen, thus the termination condition in Step 7 is carefully phrased as "$\delta(A)$ does not decrease". One method of correcting the problem of the fragments not being optimally aligned to the consensus is to do the iterative realignment process twice. First using the $\delta_a$ scoring scheme and then using $\delta_c$. While this works well the time required to do the iterative process twice isn't clearly worth the resulting improvement in the alignment. In the end, we settled on a scoring scheme that is an evenly weighted combination of $\delta_c$ and $\delta_a$. Formally we used, $\delta_{a+c}$, defined as:

$$\delta_{a+c}(x, \mathcal{X}) = .5 \cdot \delta_c(x, \mathcal{X}) + .5 \cdot \delta_a(x, \mathcal{X}) \tag{6}$$

Tests show that using $\delta_{a+c}$ gives results identical to those obtained from applying the iterative algorithm twice, once using $\delta_a$ and once using $\delta_c$, in almost every case. Unfortunately even this choice is still a heuristic and thus an optimal layout alignment cannot be guaranteed. For example, Figure 5 shows an alignment at left which is not realigned to its optimum at right for any weighted combination of $\delta_c$ and $\delta_a$.

```
c - - t g - g g c          c t - g - g g - c
c a - t g - g a c          c a - - t g g a c
c a - g g t g g c          c a g g t g g - c
c a g t g - g g c          c a - g t g g g c
c - - g g t g g c          c g - g t g g - c
   Score = 8                  Score = 7
```

**Fig. 5.** An alignment (left) not improved to its optimum (right) by any choice of $\delta_2$.

## 3.3 Speeding Up Realignment

Step 4 of the round-robin algorithm computes the best alignment with respect to $\delta_2$ between a selected sequence $S$ and the rest of the multi-alignment $B = A_Q$. In order to preserve the layout, not all alignments are permitted, but only those positioning $S$ with respect to $B$ so that the first and last symbols of $S$ are within $O(\epsilon)$ of the columns they were originally aligned against. Thus the columns of $B$ that may be realigned to match the first symbol of $S$ are in a small "band" about the column initially aligned to this symbol. The same is true about what may be aligned to the last symbol of $S$. Figure 6 illustrates this for a band of width 3.

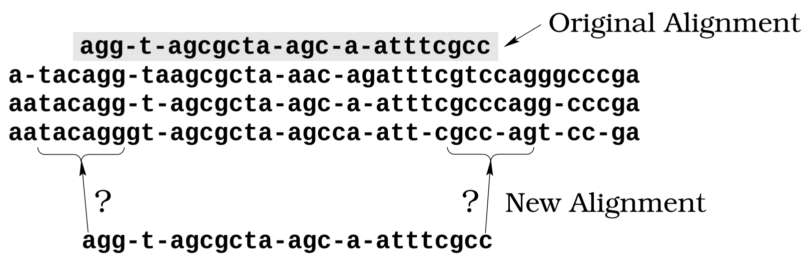

Original Alignment
agg-t-agcgcta-agc-a-atttcgcc
a-tacagg-taagcgcta-aac-agatttcgtccagggcccga
aatacagg-t-agcgcta-agc-a-atttcgcccagg-cccga
aatacagggt-agcgcta-agcca-att-cgcc-agt-cc-ga
?
?
New Alignment
agg-t-agcgcta-agc-a-atttcgcc

**Fig. 6.** Possible realignment positions for a sequence.

Recall that the assumption in a realignment scheme such as ours is that the initial layout alignment is globally correct but locally non-optimal due to an error rate of up to, say 10%, in the sequences. This, combined with the restriction on the start and end of alignments noted above, implies that the final alignment of $S$ will not be radically different from its original alignment. Realignment is expected to shift each symbol a few column left or right of its original position. This naturally gives rise to the idea of performing the standard dynamic programming comparison of $S$ and $B$ within a narrow *band* of the current alignment of $S$ and $B$. The banding approach provides efficiency, but equally importantly, guarantees that the layout is preserved.

Suppose that we have determined to realign within a band of fixed radius $r$. Further suppose $C$ is the dynamic programming matrix for the unrestricted comparison of $S$ and $B$ and that prior to realignment $a$ is the alignment between $S$ and $B$, i.e., $A = S \parallel^a B$. Now $a$ corresponds to a path in the dynamic programming matrix as illustrated in Figure 7. The *band of radius r about a* is the set of all entries $C[i, j]$ such that there exists $C[x, j] \in a$ such that $i \in [x - b, x + b]$ or there exists $C[i, y] \in a$ such that $j \in [y - b, y + b]$. Figure 7 illustrates a band of radius 3. Note that the band is a connected region of the matrix delimited by an upper and lower boundary. Computing the boundaries of a band as the banded alignment proceeds is a simple exercise. For a fixed radius, computing the best alignment between $S$ and $B$ within the band takes $O(|S|)$ time. Thus with this optimization realigning a layout takes $O(CN)$ time where $C$ is the number of complete iterations through the sequences required for convergence. As we will see $C$ is effectively bounded in practice giving us $O(N)$ expected performance.

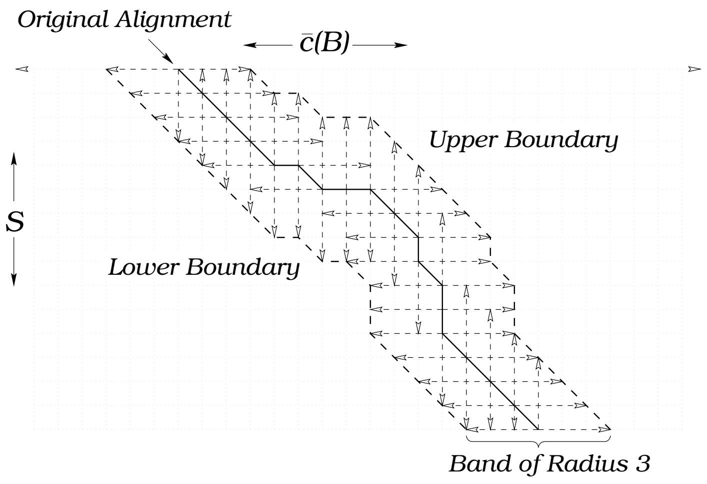


**Fig. 7.** Band of radius 3 around a hypothetical original alignment.

The critical empirical issue for our banding scheme is the choice of radius $r$ to use. A large $r$ ensures that $S$ has sufficient room to realign with $B$ in an optimum way. A small $r$ results in faster realignment of individual sequences. However, increasing $r$ can occasionally decrease the total time taken because allowing the sequences more room to realign per iteration results in the multi-alignment stabilizing in fewer cycles $C$. We ran a series of trials to determine the smallest choice of $r$ that was large enough to ensure that the results are as good as those obtained without banding.

The trials involved a series of simulated data sets where we varied the average column height or coverage $\overline{C}$ of the multi-alignments and the average amount of error $\overline{\epsilon}$ present in the data. A data set for a given trial was generated as follows. First a 100,000 basepair DNA sequence was generated by selecting each of the four bases with probability $\frac{1}{4}$. Fragments of length chosen uniformly between 300 and 500 bases were selected from the source sequence with a starting base selected uniformly from the positions available for the chosen length. Fragments were collected until the total number of bases in all the fragments $N$ became greater than $\overline{C} \cdot 100,000$. In expectation this amounted to selecting approximately $\overline{C} \cdot 250$ fragments. Each fragment was reverse complemented with probability $\frac{1}{2}$ and then errors were introduced into the fragment according to a linear ramp $[\frac{2}{3}\overline{\epsilon}, \frac{4}{3}\overline{\epsilon}]$. That is, for a fragment of length $n$, an error was introduced at position $j$ with probability $\frac{2}{3}\overline{\epsilon} + \frac{2j}{3n}\overline{\epsilon}$. The error chosen was a substitution, insertion, or deletion with equal probability. This data set of fragments was then assembled into a layout by our FAKII software suite and an initial crude layout alignment was produced as detailed in the third paragraph of the background section. This initial, unrefined layout alignment, possibly comprising several contigs, constituted the input for a given trial.

Tables 1 through 3 present the results of our experiments. Each table is for trials with $\overline{\epsilon}$ equal to 2.5%, 5.0%, and 7.5%, respectively. For each of these choices of $\overline{\epsilon}$, we ran trials with the coverage, $\overline{C}$, set to 3, 6, 9, and 12. For each choice of $\overline{\epsilon}$ and $\overline{C}$, we generated an initial alignment and for band radius from 2 to 9, we report the time taken, the average number, $\overline{I}$ of round-robin iterations required over all contigs, and the consensus score of the resulting multi-alignment(s). As $\overline{\epsilon}$ and $\overline{C}$ increase the size of the band required to ensure results as good as those without banding increases. In all cases a band of size 8 was sufficient and we have chosen this band radius as our standard for *ReAligner*.

| Coverage | 3 | | | 6 | | | 9 | | | 12 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Band Radius | Time | $\overline{I}$ | Score | Time | $\overline{I}$ | Score | Time | $\overline{I}$ | Score | Time | $\overline{I}$ | Score |
| 2 | 9.3 | 1.91 | 6,981 | 21.9 | 3.25 | 14,863 | 38.4 | 3.50 | 22,258 | 52.1 | 3.50 | 29,740 |
| 3 | 9.6 | 1.86 | 6,980 | 22.2 | 3.00 | 14,863 | 33.9 | 3.00 | 22,232 | 45.0 | 3.00 | 29,735 |
| 4 | 9.9 | 1.86 | 6,980 | 23.5 | 3.00 | 14,863 | 35.6 | 3.00 | 22,232 | 47.2 | 3.00 | 29,735 |
| 5 | 10.4 | 1.86 | 6,980 | 24.7 | 3.00 | 14,863 | 37.0 | 3.00 | 22,232 | 49.5 | 3.00 | 29,735 |
| 6 | 10.8 | 1.86 | 6,980 | 25.6 | 3.00 | 14,863 | 39.0 | 3.00 | 22,232 | 51.9 | 3.00 | 29,735 |
| 7 | 11.3 | 1.86 | 6,980 | 26.6 | 3.00 | 14,863 | 40.3 | 3.00 | 22,232 | 54.2 | 3.00 | 29,735 |
| 8 | 11.7 | 1.86 | 6,980 | 27.7 | 3.00 | 14,863 | 41.9 | 3.00 | 22,232 | 56.5 | 3.00 | 29,735 |
| 9 | 12.3 | 1.86 | 6,980 | 28.7 | 3.00 | 14,863 | 43.8 | 3.00 | 22,232 | 58.7 | 3.00 | 29,735 |

**Table 1.** Band radius trials for $\overline{\epsilon} = 2.5\%$

.

| Coverage | 3 | | | 6 | | | 9 | | | 12 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Band Radius | Time | $\overline{I}$ | Score | Time | $\overline{I}$ | Score | Time | $\overline{I}$ | Score | Time | $\overline{I}$ | Score |
| 2 | 9.7 | 2.25 | 14,052 | 25.7 | 3.10 | 29,384 | 50.6 | 3.33 | 44,192 | 68.2 | 5.00 | 59,181 |
| 3 | 10.0 | 2.19 | 14,049 | 25.6 | 2.80 | 29,368 | 40.5 | 2.67 | 44,176 | 61.0 | 4.00 | 59,124 |
| 4 | 10.5 | 2.15 | 14,049 | 26.6 | 2.70 | 29,364 | 42.7 | 2.67 | 44,173 | 63.9 | 4.00 | 59,117 |
| 5 | 11.0 | 2.15 | 14,049 | 27.9 | 2.70 | 29,364 | 48.6 | 2.83 | 44,170 | 67.7 | 4.00 | 59,115 |
| 6 | 11.6 | 2.15 | 14,049 | 29.2 | 2.70 | 29,364 | 47.0 | 2.67 | 44,170 | 71.4 | 4.00 | 59,115 |
| 7 | 12.1 | 2.15 | 14,049 | 30.5 | 2.70 | 29,364 | 49.4 | 2.67 | 44,170 | 74.4 | 4.00 | 59,115 |
| 8 | 12.6 | 2.15 | 14,049 | 32.3 | 2.70 | 29,364 | 51.3 | 2.67 | 44,170 | 77.5 | 4.00 | 59,115 |
| 9 | 13.3 | 2.15 | 14,049 | 33.2 | 2.70 | 29,364 | 53.4 | 2.67 | 44,170 | 81.3 | 4.00 | 59,115 |

**Table 2.** Band radius trials for $\overline{\epsilon} = 5.0\%$

.

| Coverage | 3 | | | 6 | | | 9 | | | 12 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Band Radius | Time | $\overline{I}$ | Score | Time | $\overline{I}$ | Score | Time | $\overline{I}$ | Score | Time | $\overline{I}$ | Score |
| 2 | 11.2 | 2.68 | 20,444 | 31.4 | 3.86 | 43,975 | 58.8 | 4.50 | 66,132 | 72.1 | 4.00 | 88,839 |
| 3 | 11.7 | 2.66 | 20,387 | 32.7 | 3.71 | 43,881 | 45.9 | 4.00 | 65,979 | 78.3 | 4.00 | 88,630 |
| 4 | 12.2 | 2.64 | 20,387 | 34.5 | 3.71 | 43,878 | 49.5 | 4.00 | 65,965 | 69.7 | 3.50 | 88,576 |
| 5 | 12.8 | 2.64 | 20,387 | 36.2 | 3.71 | 43,877 | 52.2 | 4.00 | 65,965 | 86.8 | 4.00 | 88,568 |
| 6 | 13.4 | 2.64 | 20,387 | 38.1 | 3.71 | 43,875 | 54.7 | 4.00 | 65,964 | 76.6 | 3.50 | 88,566 |
| 7 | 14.1 | 2.64 | 20,387 | 39.9 | 3.71 | 43,875 | 57.1 | 4.00 | 65,959 | 80.7 | 3.50 | 88,566 |
| 8 | 14.6 | 2.64 | 20,387 | 41.6 | 3.71 | 43,875 | 59.4 | 4.00 | 65,959 | 83.7 | 3.50 | 88,496 |
| 9 | 15.4 | 2.64 | 20,387 | 43.4 | 3.71 | 43,875 | 62.2 | 4.00 | 65,959 | 86.9 | 3.50 | 88,496 |

**Table 3.** Band radius trials for $\overline{\epsilon} = 7.5\%$

.

## 4 Results

We ran several tests giving as input to *ReAligner* the multi-alignments produced by other fragment assembly programs to measure how much *ReAligner* improved their results. We used the simulated data sets described in the previous section as the source of input to each assembler. Recall that we considered twelve data sets characterized by coverage $\overline{C}$ and average error rate $\overline{\epsilon}$.

The experiments involve three assemblers:

*CAP2*:

CAP2 by Xiaoqui Huang of Michigan Technological University [7].

*FAK3.12*:

FAKII Version 3.12, an earlier version of our assembly suite using a window-sweep heuristic to refine its layout alignment (as described in Section 2).

*FAK.MST*:

FAK II Version 4.1, our current assembler, with round-robin refinement turned off. That is, the multi-alignment is just the one produced by merging pairwise alignments as described earlier.

The last case is used as a baseline reference and the output of *ReAligner* for this case is exactly the output of FAK II Version 4.1. Because different overlap and layout algorithms are used by each assembler, the layout alignments produced by each are generally different, at least in the details of their multi-alignments. The point of the experiments here is not to compare assemblers, but to show the improvements that result from applying *ReAligner* to the output of any assembler. Indeed in several cases applying *ReAligner* to *CAP2* multi-alignments produced slightly better scoring results than applying *ReAligner* to *FAK.MST* multi-alignments.

The results of our experiments are shown in Tables 4-6, one table for each assembler. In each table there is a row for each of the twelve combinations of $\overline{\epsilon}$ and $\overline{C}$. We report the number of contigs in the assembly produced by the program as an indication of the degree of fragmentation of the solution. The two columns labeled $<X>$ *Input* give information about the multi-alignment produced by program $X$, and the two columns labeled *ReAligner Output* give the same type of information for the multi-alignment produced by *ReAligner* given the multi-alignment produced by $X$ as input. The column labeled *Score* gives the consensus alignment score of the specified multi-alignment. The column labeled *Miscalls* gives the number of differences between the consensus sequence of the specified multi-alignment and the original sequence from which the fragments were produced. Note that the the original sequence is known because we are using simulated data.

The primary thing to notice in all the tables, is that *ReAligner* improves both the consensus scores and the number of miscalls for all programs under all conditions. The improvement is the least when coverage and error rate are low, but as either one or both of these are increased the difference begins to become substantial. The results further show that both the *FAK3.12* and *CAP2* programs could have produced better multi-alignments. At an error rate of 2.5%, *CAP2* produces multi-alignments that are improved on just slightly, but when error rates are high the change in the number of miscalls is as high as 75%. While the change in the number of miscalls clearly demonstrates that an alignment is superior, the change in consensus scores is harder to interpret. In some cases this score improves while the number of miscalls remains roughly the same. What is happening in these case is that the alignment improves the evidence that the given consensus symbol is indeed correct, in say, the sense of the Church-Waterman statistic [3]. For example, in Figure 8 a portion of a *CAP2* multi-alignment at left was improved by ReAligner to the portion at right. While

| | | | FAK.MST Input | | ReAligner Output | |
|---|---|---|---|---|---|---|
| Error Rate | Coverage | Contigs | Score | Miscalls | Score | Miscalls |
| 2.5 | 3 | 43 | 7,913 | 1,944 | 6,980 | 1,664 |
| | 6 | 4 | 19,063 | 684 | 14,863 | 238 |
| | 9 | 2 | 30,593 | 387 | 22,232 | 38 |
| | 12 | 2 | 44,806 | 409 | 29,735 | 14 |
| 5.0 | 3 | 53 | 16,281 | 4,143 | 14,049 | 3,457 |
| | 6 | 10 | 40,198 | 1,985 | 29,364 | 798 |
| | 9 | 6 | 67,468 | 1,427 | 44,170 | 233 |
| | 12 | 1 | 98,851 | 1,176 | 59,115 | 29 |
| 7.5 | 3 | 44 | 24,332 | 6,055 | 20,387 | 4,890 |
| | 6 | 7 | 62,519 | 3,399 | 43,875 | 1,294 |
| | 9 | 2 | 107,397 | 2,687 | 65,959 | 269 |
| | 12 | 2 | 159,199 | 2,453 | 88,496 | 50 |

**Table 4.** Effect of *ReAligner* on *FAK.MST* Layout Alignments.

| | | | FAK3.12 Input | | ReAligner Output | |
|---|---|---|---|---|---|---|
| Error Rate | Coverage | Contigs | Score | Miscalls | Score | Miscalls |
| 2.5 | 3 | 43 | 7,087 | 1,682 | 6,979 | 1,663 |
| | 6 | 4 | 15,263 | 270 | 14,864 | 238 |
| | 9 | 2 | 22,041 | 45 | 22,233 | 39 |
| | 12 | 1 | 23,041 | 45 | 22,233 | 11 |
| 5.0 | 3 | 53 | 14,336 | 3,553 | 14,045 | 3,457 |
| | 6 | 10 | 30,686 | 904 | 29,365 | 799 |
| | 9 | 2 | 46,770 | 180 | 44,262 | 123 |
| | 12 | 1 | 63,453 | 45 | 59,111 | 26 |
| 7.5 | 3 | 44 | 21,125 | 5,131 | 20,386 | 4,884 |
| | 6 | 7 | 46,548 | 1,594 | 43,880 | 1,295 |
| | 9 | 2 | 70,885 | 420 | 65,958 | 268 |
| | 12 | 1 | 96,585 | 99 | 88,570 | 38 |

**Table 5.** Effect of *ReAligner* on *FAK3.12* Layout Alignments.

| | | | CAP2 Input | | ReAligner Output | |
|---|---|---|---|---|---|---|
| Error Rate | Coverage | Contigs | Score | Miscalls | Score | Miscalls |
| 2.5 | 3 | 49 | 7,030 | 1,668 | 6,967 | 1,658 |
| | 6 | 4 | 15,164 | 254 | 14,861 | 232 |
| | 9 | 2 | 22,882 | 41 | 22,232 | 38 |
| | 12 | 1 | 30,889 | 11 | 29,737 | 11 |
| 5.0 | 3 | 54 | 14,389 | 3,457 | 14,077 | 3,380 |
| | 6 | 8 | 30,771 | 802 | 29,431 | 716 |
| | 9 | 2 | 47,167 | 153 | 44,256 | 123 |
| | 12 | 1 | 64,126 | 43 | 59,121 | 28 |
| 7.5 | 3 | 56 | 21,068 | 5,046 | 20,323 | 4,894 |
| | 6 | 6 | 47,040 | 1,454 | 43,877 | 1,263 |
| | 9 | 3 | 72,838 | 382 | 65,949 | 260 |
| | 12 | 1 | 100,697 | 119 | 88,560 | 31 |

**Table 6.** Effect of *ReAligner* on *CAP2* Layout Alignments.

the consensus **GCACG[TC]TGTT[G-]AAA** versus **GCACGTTGTAAA** changes by only 1-3 miscalls (depending on how the tied consensus classes are resolved), the change in consensus score is dramatic and one much more clearly sees the evidence for the consensus at right.

```
a-cac--c--ttgtaaa            a-caccttgt-aaa
g-cacg-ttgtt--aa-            g-cacgttgt-taa
ggcacg-ttgt---aaa            ggcacgttgt-aaa
g-c---ac-gttgtaaa            g-cacgttgt-aaa
     gcacgttgtaaa            g-cacgttgt-aaa
g-c---ac-gttgtaac            g-cacgttgt-aac
g-c---ac-gttgtaaa            g-cacgttgt-aaa
g-c---ac-gttgtaaa            g-cacgttgt-aaa
gg-acg-ttgt---aaa            g-gacgttgt-aaa
g-c--g-c-gtcggaaa            g-cgcgtcgg-aaa
g-cacg-ctgtc--aaa            g-cacgctgtcaaa
ggcacg-ttgt---aaa            ggcacgttgt-aaa
g-cacg-ttgt---aaa            g-cacgttgt-aaa
g-cacg-ttgt---aaa            g-cacgttgt-aaa
g-cacg-ttgt---aa-            g-cacgttgt--aa
        tgttgtaaa                tgttgt-aaa

   Score = 68                  Score = 14
```

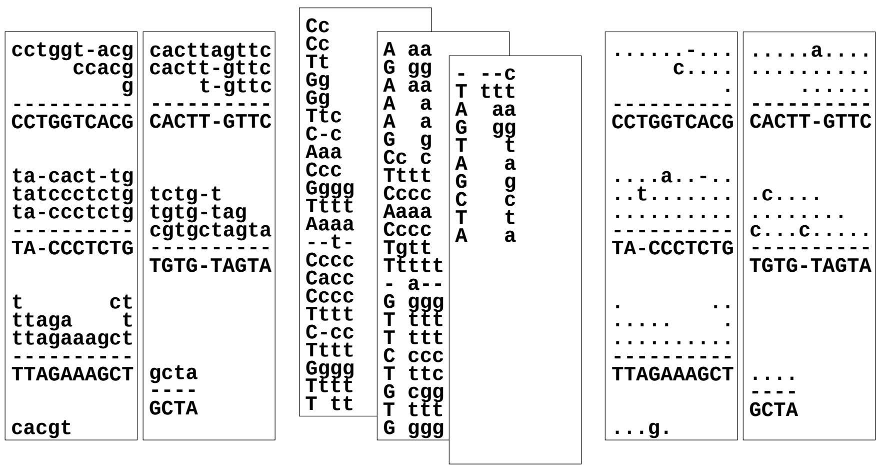
**Fig. 8.** A portion of a *CAP2* multi-alignment (left) and after given to *ReAligner* (right).

## 5 The Software

From the perspective of building software tools, the most problematic issue in dealing with layout alignment is the variety of possible input and output formats. To resolve this issue, we built two separate programs, *ReAligner* and *Converter* both freely available at `ftp://ftp.cs.arizona.edu/realigner`. *Converter* translates between various formats allowing *Realigner* to handle the single task of realigning a multi-alignment according to our round-robin algorithm. Thus if one wants to produce a realigned multi-alignment in format $Y$, given the initial multi-alignment in format $X$, one simply applies *Converter* to translate from format $X$ to the unique form accepted by *Realigner*; then applies *Realigner* to refine the multi-alignment; and finally uses *Converter* to translate the refined alignment to format $Y$. On machines using the UNIX operating system, this may be written as a simple "pipe": `Converter <X | ReAligner | Converter >Y`.

There are two distinctly different encodings of multi-alignments: *horizontal* and *vertical*. Typically a human reader prefers to see a layout with each fragment written horizontally across the page. On the other hand, it is much simpler for a program to input and parse a layout in which fragments are written vertically down the page. To describe each format concisely, begin by considering the $k$ by $l$ matrix $A$ used to formally describe a layout alignment. The number of rows $k$ may be reduced significantly by placing several fragments in a row when they are mutually non-overlapping with at least one blank symbol between them. Formally, an assignment $row(i)$ of each fragment $S_i$ is *legal* if $row(i) = row(j)$ implies $[beg_A(i)-1, end_A(i)+1] \cap [beg_A(j), end_A(j)] = \emptyset$. Greedily allocating each fragment to the next available row in a left-to-right sweep gives a legal assignment involving the fewest rows, say $c$. Let $A'$ be the resulting $c$ by $l$ matrix.

To display $A'$ in horizontal mode at *width* $w$ and *separation* $s$, one first divides $A'$ into $c$ by $w$ blocks, save for the last which may have $l(mod\ w)$ columns. Each block is displayed in order with each row of a block occupying a line of the output. If the last few rows in a block consist solely of blank symbols, then these rows are not output. The last printed row of each block is followed by a line of $w$ dashes and then $s$ blank lines. If there is more than one contig, then each contig is output as above and another line of $w$ dashes separates it from the next contig. The key motivation for the format above is that it has the property that one can unambiguously tell when a fragment has been broken across a line, and thus one can unambiguously reconstruct $A'$ given such a representation as input. Figure 9.a gives an example.



```
cctggt-acg
     ccacg
         g
----------
CCTGGTCACG


ta-cact-tg
tatccctctg
ta-ccctctg
----------
TA-CCCTCTG


t       ct
ttaga    t
ttagaaagct
----------
TTAGAAAGCT


cacgt
```

```
cacttagttc
cactt-gttc
     t-gttc
----------
CACTT-GTTC



tctg-t
tgtg-tag
cgtgctagta
----------
TGTG-TAGTA






gcta
----
GCTA
```

```
Cc
Cc
Tt
Gg
Gg
Ttc
C-c
Aaa
Ccc
Gggg
Tttt
Aaaa
--t-
Cccc
Cacc
Cccc
Tttt
C-cc
Tttt
Gggg
Tttt
T tt
```

```
A aa
G gg
A aa
A  a
A  a
G  g
Cc c
Tttt
Cccc
Aaaa
Cccc
Tgtt
Ttttt
- a--
G ggg
T ttt
T ttt
C ccc
T ttc
G cgg
T ttt
G ggg
```

```
-  --c
T  ttt
A   aa
G   gg
T    t
A    a
G    g
C    c
T    t
A    a
```

```
......-...
     c....
         .
----------
CCTGGTCACG


....a..-..
..t.......
..........
----------
TA-CCCTCTG


.       ..
.....    .
..........
----------
TTAGAAAGCT


...g.
```

```
.....a....
..........
    ......
----------
CACTT-GTTC


.c....
........
c...c.....
----------
TGTG-TAGTA



....
----
GCTA
```

*(a) Horizontal* *(b) Vertical* *(c) Consensus+Dots*

**Fig. 9.** Encodings of the layout alignment of Figure 1 on 10 × 22 paper.

To display $A'$ in vertical mode, one simply outputs each column per line of output in sequence. Note that each line is $c$ symbols long, save that trailing blanks may be removed from the end of each line. A blank line separates contigs. Further note that this form is especially easy to input and output, and especially suitable for any auxiliary computation that computes column-based information, e.g. a consensus symbol or quality measure for each column. Figure 9.b gives an example.

An alignment may optionally have a consensus sequence. In horizontal format this follows the line of dashes at the end of each output block. In vertical format it is assumed to be the sequence in the first column. While *Converter* will compute and output a consensus sequence as defined in this paper, this sequence can be computed by a user-supplied program. If one provides a consensus then there is another desirable representation of the alignment in which all symbols in a column equal to the consensus symbol of that column are displayed as a period symbol. This form of display makes it easy to see where fragments differ from the consensus. Figure 9.c gives an example.

Both *ReAligner* and *Converter* are designed as UNIX pipes, taking data from the standard input, `stdin`, and placing results on the standard output, `stdout`. Within an alignment, the programs recognize the characters `a, c, g, t, A, C, G, T, -` as the four bases and the intra-fragment pad character. All other characters are treated as if they were the symbol `n` which does not match anything (including itself). Both *ReAligner* and *Converter* preserve the case of input letters. Comments lines are permitted in the input to the programs and are designated by beginning a line with the character "`%`". Comment lines are copied to the output without alteration. All comments that appear within a contig in the input are printed at the beginning of that contig in the output.

*ReAligner* expects an input alignment in vertical mode, without a consensus and outputs the realignment in the same format. *ReAligner* takes one optional argument of the form `-b#` giving the size `#` of the band radius to use (the default is 8).

*Converter* accepts input in any of the formats described above, but it must be told specifically what format is forthcoming. The lower-case options below describe this expected input format while their upper-case counterparts describe the output format desired.

| | |
|---|---|
| `h/H` | horizontal format |
| `v/V` | vertical format |
| `c/C` | consensus string in input/output |
| `d/D` | dots in input/output |
| `s/S#` | use `#` lines for horizontal block separation (default 3) |
| `r/R#` | use a row length of `#` for horizontal block width (default 80) |

The use of some options is context dependent. Namely, one can elect the `-d(D)` option only if the `-c(C)` option is specified, and one must give the `-s(S)` and `-r(R)` options if the `-h(H)` option is specified and the the default value is not correct and/or desired. All options may be run together as a single string as in:

```
Converter -vHCD  # Convert from vertical without consensus to horizontal with
                   consensus, dots, and default separation and width
```

save for the separation and width options which must be specified individually as in:

```
Converter -hcVC -s2 -r50  # Convert from 2x50 horizontal format with con-
                            sensus to vertical format with consensus
```

Note carefully, that in the first example *Converter* computes the consensus as we have defined it in this paper and outputs it with its horizontal display. In the second example, since a consensus came with the input, *Converter* does not need to compute one for the output but simply passes the input consensus on to the vertical output. It is further the case that *Converter* has the side effect of printing to `stderr` a summary consisting of the consensus score and percentage of characters that do not match the consensus sequence.

We conclude by re-emphasizing that the role of *Converter* is not just so that *ReAligner* has the convenience of inputing and outputting vertical alignments, but also for the user who may wish to do their own pre- or post-processing to alignments. For example, a user can easily write a program, say *MyConsensus*, that reads a vertical alignment without consensus and outputs the vertical alignment with a consensus sequence computed according to whatever definition they prefer. One may then build the UNIX pipe:

```
converter -hV <X | ReAligner | MyConsensus | converter -vcHC >Y
```

to improve the alignment of a horizontally coded alignment and attach their concept of consensus to the result.

**Acknowledgement**
We would like to thank Webb Miller for his comments on the design of the *ReAligner* and *Converter* programs. Also we thank David Lipman for encouraging us to encapsulate this work in a fashion independent of our FAKII assembly suite.

## References

1. A. Krogh, M. Brown, I.S. Mian, K. Sjolander and D. Haussler (1994) Hidden Markov models in computational biology, *J. Mol. Biol.*, **235**, 1501-1531.
2. Barton, J.G. and Sternberg, R.F. (1987) A strategy for rapid multiple alignment of protein sequences, *J. Mol. Biol.*, **198**, 327-337.
3. G. Churchill and M. Waterman, The Accuracy of DNA Sequences: Estimating Sequence Quality. *Genomics* 14 (1992), 89-98.
4. F. Corpet, Multiple sequence alignment with hierarchical clustering. *Nucl. Acids. Res.* 16 (1988), 10881–10890.
5. D. Feng and R. Doolittle, Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.* 25 (1987), 351–360.
6. D. Higgins and P. Sharpe, CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene* 73 (1988), 237–244.
7. X. Huang, An improved sequence assembly program. *Genomics* 33 (1996), 21–31.
8. Rabiner, L. (1989) A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proc. of the IEEE*, Vol. 77, No. 2, 257-285.
9. W. Taylor, Multiple sequence alignment by a pairwise algorithm. *CABIOS* 3 (1987), 81–87.
10. M. Waterman and M. Perlwitz, Line geometries for sequence comparison. *Bull. Math. Biol.* 46 (1984), pp. 567–577.