

# K-Group A\* for Multiple Sequence Alignment with Quasi-Natural Gap Costs

Rong Zhou and Eric A. Hansen

Department of Computer Science and Engineering  
Mississippi State University, Mississippi State, MS 39762  
{rzhou,hansen}@cse.msstate.edu

## Abstract

*Alignment of multiple protein or DNA sequences is an important problem in Bioinformatics. Previous work has shown that the A\* search algorithm can find optimal alignments for up to several sequences, and that a K-group generalization of A\* can find approximate alignments for much larger numbers of sequences [6]. In this paper, we describe the first implementation of K-group A\* that uses quasi-natural gap costs, the cost model used in practice by biologists. We also introduce a new method for computing gap-opening costs in profile alignment. Our results show that K-group A\* can efficiently find optimal or close-to-optimal alignments for small groups of sequences, and, for large numbers of sequences, it can find higher-quality alignments than the widely-used CLUSTAL family of approximate alignment tools. This demonstrates the benefits of A\* in aligning large numbers of sequences, as typically compared by biologists, and suggests that K-group A\* could become a practical tool for multiple sequence alignment.*

## 1. Introduction

The multiple alignment of DNA or protein sequences is an important problem in Bioinformatics, with applications ranging from the discovery of evolutionary homologies among different species to predicting secondary and tertiary structure of proteins.

It is well known that multiple alignment problem is equivalent to finding a least-cost path in an  $N$ -dimensional lattice, where  $N$  is equal to the number of sequences being aligned. Although dynamic programming is a traditional approach to solving this problem, much improved performance has been achieved by using the A\* algorithm, which uses a lower-bound function (i.e., an admissible heuristic) to reduce the search state space for finding an optimal alignment [6, 9, 14, 8, 10, 15, 16].

Since the multiple sequence alignment problem is NP-hard [7], there is little hope that exact methods can scale up

to find optimal alignments of more than a dozen sequences. Most widely-used tools for multiple alignment rely on approximation and do not guarantee a mathematically optimal alignment. For example, CLUSTAL V [5] and CLUSTAL W [12], the most popular multiple alignment tools among biologists, use a variation of the progressive alignment strategy. In progressive alignment, a multiple alignment is constructed by performing a succession of pairwise alignments, which involve aligning a pair of sequences, a sequence to a profile, or a pair of profiles. (A profile is a pre-existing alignment of a subset of sequences, and aligning profiles is sometimes referred to as “aligning alignments.”) Progressive alignment is a heuristic method, and the most important heuristic is to align the most similar pairs of sequences first, where similarity is determined by first generating a binary phylogenetic tree.

Progressive alignment has the advantage of being very fast, and in many cases the resulting alignments are reasonable. But as with other approximation methods, it has drawbacks. It performs a kind of greedy search that may get stuck in local minima. In particular, the phrase “once a gap, always a gap” refers to the problem that gaps introduced in early pairwise alignments remain in the final alignment, even when keeping these gaps results in a worse alignment.

Among methods developed to alleviate this problem, some of the most successful use an iterative refinement strategy [2]. Iterative-refinement algorithms begin with an initial alignment, and then try to improve the alignment each iteration. They do so by creating a modified version of the profile alignment problem, where the modification is different each iteration, and solving it using a computationally tractable algorithm (e.g., progressive alignment). The hope is that controlled perturbation in the problem’s state space will eventually take the search out of local minima.

Iterative-refinement algorithms have proved successful in producing high-quality multiple alignments. For example, in a comparison study of multiple sequence alignment programs, Thompson *et al.* [13] found that three of the four best-performing programs employ iterative-refinement strategies.

The iterative-refinement approach is traditionally used with dynamic programming. However, Ikeda and Imai [6] point out that it can also be used with the search algorithm  $A^*$ , and describe a  $K$ -group iterative-refinement generalization of  $A^*$  that uses linear gap costs. In this paper, we describe a  $K$ -group iterative-refinement generalization of  $A^*$  that uses quasi-natural gap costs, which is the cost metric used in practice by biologists. This allows us to compare the performance of our implementation of  $K$ -group  $A^*$  to multiple sequence alignment tools used in the biological community. We compare it to OMA (Optimal Multiple Alignment) [11], a state-of-the-art exact alignment tool, and the popular CLUSTAL family of approximate alignment tools. We show that it finds better alignments without much additional computational effort.

The rest of the paper is organized as follows. In Section 2, we review the literature on profile alignment and its generalization called  $K$ -group alignment. In Section 3, we describe our main contribution – a  $K$ -group iterative-refinement generalization of  $A^*$  with quasi-natural gap costs. Section 4 compares the performance of our new algorithm to OMA, CLUSTAL V, and CLUSTAL W.

## 2. Background

In this section, we review sum-of-pairs multiple sequence alignment, profile alignment and its generalization to  $K$ -group alignment, and iterative refinement strategies.

### 2.1. Sum-of-pairs multiple sequence alignment

Formally, a sequence  $S$  of length  $L$  is an ordered list of characters  $(S[1], S[2], \dots, S[L])$  from a finite alphabet  $\Sigma$  that does not include the reserved gap character  $(-)$ . Let  $S_1, S_2, \dots, S_N$  denote  $N$  sequences of length  $L_1, L_2, \dots, L_N$ . An alignment of these sequences is a  $N \times J$  matrix  $A = (a_{nj})$ , such that

1.  $a_{nj} \in \Sigma' = \Sigma \cup \{-\}$  (that is, the matrix is “padded” with gap characters).
2. Ignoring gap characters in the  $n$ -th row of  $A$  will reproduce sequence  $S_n$ .
3.  $A$  has no column that consists only of gaps.

A distance matrix  $d : \Sigma' \times \Sigma' \rightarrow \mathfrak{R}$  describes the evolutionary distance between each pair of characters in  $\Sigma'$ . Note that  $d(-, -) = 0$  since matching gaps are ignored.

The gap cost function  $g(r)$  describes the log-likelihood of observing a run of gaps of length  $r$ . By treating the opening of a gap differently from the extension of an existing gap, affine gap costs are known to be effective in producing biologically plausible alignments. The general form of an affine gap cost function is as follows,

$$g(r) = v + ur$$

where  $v$  is the gap-opening penalty and  $u$  is the gap-extension penalty. Note that the distance matrix  $d$  can absorb the linear coefficient  $u$  of the gap cost function by defining  $d(x, -) = u \forall x \in \Sigma$ .

The *sum-of-pairs* alignment cost of  $A$  is defined as

$$SP(A) = \sum_{1 \leq l < n \leq N} S_{ln}$$

$$S_{ln} = \sum_{j=1}^J d(a_{lj}, a_{nj}) + v g_{ln}$$

where  $g_{ln}$  is the number of gap openings between sequences  $l$  and  $n$  in the final alignment. The problem of multiple sequence alignment is to find an alignment  $A^{\text{opt}}$  which minimizes the sum-of-pairs cost.

It is well known that multiple sequence alignment can be solved optimally by dynamic programming in a hypercube with dimensionality equal to the number of sequences  $N$ . However, both the time and space needed to solve the problem grow exponentially in the number of sequences, leaving exact methods impractical to all but the smallest instances.

### 2.2. Profile alignment

Profile alignment is a widely-used approximation technique to reduce the dimensionality of the hypercube for which dynamic programming must be carried out. The basic idea of profile alignment is to “freeze” the alignment *within* each profile in order to optimize the alignment *between* profiles.

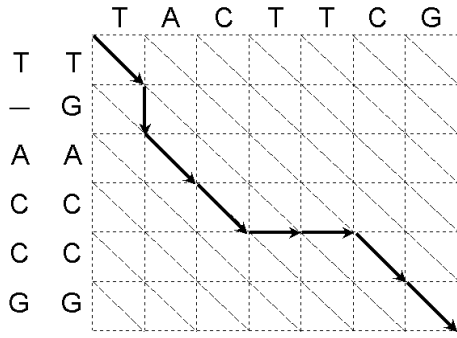
Let  $X = (x_{mi})$  and  $Y = (y_{nj})$  be two groups of pre-aligned sequences, with  $M$  and  $N$  being the number of sequences in each respective group. Let  $A$  be the profile alignment between profiles  $X$  and  $Y$ . Because alignment inside a group is not allowed to change before and after a profile alignment, the sum-of-pairs cost of  $A$  can be decomposed into three parts as follows,

$$SP(A) = SP(X) + SP(Y) + SP(X \cdot Y) \quad (1)$$

where  $SP(X \cdot Y) = \sum_{m=1}^M \sum_{n=1}^N S_{mn}$ . Note that minimizing  $SP(X \cdot Y)$  is equivalent to minimizing  $SP(A)$ , because both  $SP(X)$  and  $SP(Y)$  are constants under the assumptions of profile alignment.

The above formulation of profile alignment takes care of the case when a sequence is aligned against a profile, because one of the two groups may consist only of a single sequence.

To help the reader understand the concept of profile alignment, especially sequence-to-profile alignment, which plays an important role in our algorithm, we use the following illustrative example.



(a)

T	-	A	C	T	T	C	G
T	-	A	C	-	-	C	G
T	G	A	C	-	-	C	G

(b)

**Figure 1. Example of sequence-to-profile alignment. Panel (a) shows the state space of the sequence-to-profile alignment together with a least-cost path marked by a chain of solid arrows. Panel (b) shows the alignment that corresponds to the least-cost path shown in panel (a).**

*Example of profile alignment* Consider the problem of aligning a profile

$$X = \begin{array}{cccccc} T & - & A & C & C & G \\ T & G & A & C & C & G \end{array}$$

against a sequence  $Y = \text{TACTTCG}$  using a simple cost function: zero for a match, one for a substitution or an extension of an existing gap, and two for opening a new gap.

Figure 1(a) shows the state space of this problem. A least-cost path is marked by a chain of solid arrows in the same figure. A horizontal arrow corresponds to the insertion of a gap into both sequences of profile  $X$  shown on the rows; whereas a vertical arrow corresponds to the insertion of a gap into the single sequence  $Y$  shown on the columns.

All columns in profile  $X$  are matches except for the second column, which opens a new gap with the length equal to 1. Therefore, the affine gap cost of the second column in profile  $X$  is  $g(1) = v + ur = 2 + 1 \times 1 = 3$ . Thus,  $SP(X) = 3$ . Because  $Y$  is a single sequence, which does not have a sum-of-pairs cost by itself, the term  $SP(Y)$  in Equation (1) can be ignored.

According to Figure 1(b),  $SP(X \cdot Y)$  is found to be 11. Thus the cost of the multiple alignment  $A$  is  $SP(A) =$

$SP(X) + SP(X \cdot Y) = 3 + 11 = 14$ , which happens to be the optimal alignment cost for this example.  $\square$

### 2.3. $K$ -group alignment

Indeed, the profile alignment described in the previous subsection should be called *pairwise profile alignment*, because it only aligns a pair of profiles at a time. Similar to the way pairwise sequence alignment is generalized to multiple sequence alignment, pairwise profile alignment can also be generalized to multiple profile alignment.

Let  $G_1, G_2, \dots, G_K$  be  $K$  profiles. Let  $A$  be the multiple profile alignment of these  $K$  profiles. The alignment cost of  $A$  can be written as,

$$SP(A) = \sum_{k=1}^K SP(G_k) + \sum_{1 \leq l < k \leq K} SP(G_l \cdot G_k)$$

and minimizing the term  $\sum_{1 \leq l < k \leq K} SP(G_l \cdot G_k)$  is equivalent to minimizing  $SP(A)$  in a multiple profile alignment with  $K$  profiles.

The above formulation of multiple profile alignment is called  *$K$ -group alignment* in the literature [6]. By allowing  $K > 2$  groups of sequences to be aligned simultaneously,  $K$ -group alignment creates a finer-grained approximation of the original multiple alignment problem than pairwise profile alignment does. Furthermore, the degree of approximation can be controlled by varying the number of groups in a  $K$ -group alignment. In the extreme case where  $K$  is equal to the total number of sequences being aligned,  $K$ -group alignment solves the original multiple alignment problem and the resulting alignment is guaranteed to be optimal. From this perspective,  $K$ -group alignment unifies the classic pairwise profile alignment with optimal multiple alignment in the same algorithm, creating a spectrum of approximation methods with pairwise profile alignment and optimal multiple alignment at the two ends of the spectrum.

### 2.4. Iterative refinement

To further reduce the cost of an alignment, an iterative-refinement strategy [2] is often used in combination with  $K$ -group alignment. It works as follows.

1. Create initial alignment using any approximation method.
2. Randomly divide  $N$  sequences into  $K \geq 2$  groups.
3. Remove trivial gaps in each group.
4. Solve the  $K$ -group alignment problem.
5. If lower cost, change the current alignment to the new one.
6. If stopping criterion is met, stop; otherwise go to step 2.

There is obviously more than one way to divide  $N$  sequences into  $K < N$  groups, in step 2 above. We focus on the special case of  $K$ -group alignment in which the first  $K - 1$  groups consist of only a single sequence each, and

the  $K$ -th group contains the other  $N - K + 1$  sequences. The reasons are explained below.

First, this strategy has been found to produce better alignments than a random-grouping strategy in which  $N$  sequences are randomly divided into  $K < N$  groups such that no group is empty [6]. Second, when this strategy is used with A\*, effort to solve  $\binom{K-1}{2}$  out of  $\binom{K}{2}$  sub-alignment problems (for computing a pairwise alignment heuristic function) in *each* iteration can be spared if a sequence-to-sequence preprocessing step is performed upfront. Last but not least, the overhead of  $K$ -group alignment can be substantially reduced using this strategy, since most of the extra work compared to ordinary  $K$ -sequence alignment is confined to the last group, since it is the only group with more than one sequence. The above grouping strategy is referred to as *K-restricted-grouping* by Ikeda and Imai [6].

As for the stopping criterion in step 6, one possibility is to stop when the algorithm cannot decrease the cost of the current alignment after trying all possible ways of dividing  $N$  sequences into  $K$  groups. For *K-restricted-grouping*, the number of ways is  $\binom{N}{K-1}$ , which grows quickly with the number of sequences  $N$ . Thus, a more practical stopping criterion is to terminate when the algorithm has performed consecutively a certain number of iterations (called *max.trials* in our implementation) without finding a lower-cost alignment than the best alignment found so far. Note that every time a lower-cost alignment is found, the algorithm is allowed to run another *max.trials* number of iterations, in order to reduce the alignment cost even further.

### 3. K-group A\* with quasi-natural gap costs

In iterative refinement,  $K$ -group alignment is performed repeatedly until the alignment is good enough, as determined by some stopping criterion. We use A\* to perform  $K$ -group alignment, since A\* is more efficient than dynamic programming.

In this section, we describe how to compute quasi-natural gap costs in  $K$ -group alignment. The dynamic programming recursion we develop can be used by either A\* or dynamic programming. Previous work [6] on  $K$ -group alignment only supports linear gap costs, which are not fully justified by theories of evolution. In practice, all sequence alignment tools used by biologists support some form of affine gap cost.

As reviewed in Section 2.1, an affine gap cost function consists of a gap-opening cost and a gap-extension penalty. Since the gap-extension penalty can be represented in the distance matrix, the key issue is the gap-opening cost. Use of an affine gap cost in multiple sequence alignment presents a particular challenge because of the complexity of identifying the opening of a gap. Altschul [1] considers an affine gap cost function for multiple sequence align-

ment called *natural gap costs*, in which the number of gap openings in a multiple alignment is defined as the sum of the number of gap openings in all pairwise projections. But the number of relevant histories that need to be maintained for each state, in order to compute natural gap costs, grows prohibitively with the number of sequences, and makes this definition impractical. (For natural gap costs, the number of relevant histories grows faster than the factorial of  $N$ , or  $N!$ , where  $N$  is the number of sequences.) This led Altschul to propose a simplified version called *quasi-natural gap costs*, which slightly overcount the number of gap openings, in order to reduce the number of histories that must be maintained. (For quasi-natural gap costs, the number of relevant histories is only  $2^N - 1$ .) A nice property of quasi-natural gap costs is that the incoming edge of a state completely determines the number of gap openings for any of its outgoing edges. We refer to the cited reference for a detailed discussion of quasi-natural gap costs.

Quasi-natural gap costs are employed in multiple sequence alignment programs used in practice by biologists, including MSA [4] and OMA [11] for exact alignment of a small number of sequences. However, quasi-natural gap costs have not been used before in a  $K$ -group alignment algorithm. We now discuss how to do this.

#### 3.1. Quasi-natural gap costs in profile alignment

To support quasi-natural gap costs in  $K$ -group alignment, it helps to first consider how to compute quasi-natural gap-opening costs in pairwise profile alignment. Below we describe a naive extension of the classic dynamic programming algorithm for pairwise profile alignment. Then we present our new algorithm.

Let  $\vec{x}_i$  and  $\vec{y}_j$  be the  $i$ -th and  $j$ -th column in profiles  $X$  (with  $M$  sequences) and  $Y$  (with  $N$  sequences), respectively. Let  $D^\alpha(i, j)$  denote the cost of the sub-alignment between the first  $i$  columns of  $X$  and the first  $j$  columns of  $Y$ . The superscript  $\alpha$  denotes the direction from which the final segment of the alignment path came. To be more specific,  $\alpha = 1, 2$  and  $3$  indicate that the path arrives from an upper node, a left node and a diagonally upper-left node, respectively. The dynamic programming equations for a naive algorithm that supports quasi-natural gap costs for pairwise profile alignment can be written as follows,

$$\begin{aligned} D^1(i, j) &= \min\{D^1(i-1, j), D^2(i-1, j) + V, \\ &\quad D^3(i-1, j) + V\} + d(\vec{x}_i, -) \\ D^2(i, j) &= \min\{D^1(i, j-1) + V, D^2(i, j-1), \\ &\quad D^3(i, j-1) + V\} + d(-, \vec{y}_j) \\ D^3(i, j) &= \min_{\alpha=1}^3\{D^\alpha(i-1, j-1)\} + d(\vec{x}_i, \vec{y}_j) \end{aligned}$$

where  $V = MNv$ ,  $d(\vec{x}_i, -) = N \sum_{m=1}^M d(x_{mi}, -)$ ,

$$d(-, \vec{y}_j) = M \sum_{n=1}^N d(-, y_{nj}), \text{ and } d(\vec{x}_i, \vec{y}_j) = \sum_{m=1}^M \sum_{n=1}^N d(x_{mi}, y_{nj}).$$

The problem with the above dynamic programming algorithm is that the calculation of  $V$ , the quasi-natural gap-opening cost, does not take into account the presence of pre-existing gaps. For example, consider the extreme case where there is only a single non-gap position in column  $\vec{x}$ . According to the above equation, the gap-opening cost is still  $MNv$ , which is  $M$  times greater than the real gap-opening cost, which is only  $Nv$ . As a result, the algorithm described above fails to minimize  $SP(X \cdot Y)$  in most cases. This drawback was first noticed by Gotoh [3], who describes a way to do pairwise profile alignment that correctly minimizes the sum-of-pairs cost. However, he considers natural gap costs and pairwise profile alignment only.

To account for internal gaps in each profile, we present a new algorithm that uses extra information to compute the quasi-natural gap-opening costs. Let  $\tau(\vec{x})$  be the function that counts the number of non-gap positions in column  $\vec{x}$ . We modify the dynamic programming equations as follows,

$$\begin{aligned} D^1(i, j) &= \min\{D^1(i-1, j), D^2(i-1, j) + V^1 \\ &\quad D^3(i-1, j) + V^1\} + d(\vec{x}_i, -) \\ D^2(i, j) &= \min\{D^1(i, j-1) + V^2, D^2(i, j-1), \\ &\quad D^3(i, j-1) + V^2\} + d(-, \vec{y}_j) \\ D^3(i, j) &= \min_{\alpha=1}^3 \{D^\alpha(i-1, j-1)\} + d(\vec{x}_i, \vec{y}_j) \end{aligned}$$

where  $V^1 = N\tau(\vec{x}_i)v$  and  $V^2 = M\tau(\vec{y}_j)v$ .

In the above equations,  $V^1$  and  $V^2$  are what we call *vertical* and *horizontal gap-opening costs*, respectively. Because our new algorithm takes into account the orientation of gaps as well as the number of non-gap positions in the affected column of a profile, gap-opening costs in profile alignment can be evaluated with much greater accuracy.

To illustrate this, we use the same example shown in Figure 1. Recall that a vertical move corresponds to the insertion of a gap to the single sequence  $Y$  shown on the columns. According to the naive algorithm, the gap-opening cost of the only vertical move in the alignment path shown in Figure 1(a) will be calculated as  $V = MNv = 2 \times 1 \times 2 = 4$ , which is incorrect, because there is in fact only one new gap opening and the internal gap in the second column of profile  $X$  should not contribute any gap-opening cost to  $SP(X \cdot Y)$ . According to our new algorithm, the gap-opening cost of a vertical move is determined by  $V^1 = N\tau(\vec{x}_i)v$ . Note that the vertical move shown in Figure 1(a) causes a gap to match against the second column of profile  $X$ . Since  $\tau(\vec{x}_2) = \tau(\begin{smallmatrix} - \\ \mathbf{G} \end{smallmatrix}) = 1$ , the gap-opening cost is therefore correctly computed as  $V^1 = N\tau(\vec{x}_2)v = 1 \times 1 \times 2 = 2$ .

Note that the number of non-gap positions in each column of the profiles can be pre-computed and stored in a

lookup table, in order to accelerate the calculation of gap-opening costs. The extra time and space overhead for this is negligible in a multiple alignment.

It is easy to show that when each profile consists of a single sequence, our new algorithm computes quasi-natural gap-opening costs in exactly the same way as in optimal multiple sequence alignment.

## 4. Computational results

We tested the performance of  $K$ -group A\* in aligning real protein sequences and compared the results to those of the exact alignment tool, OMA [11], and the approximate alignment tools, CLUSTAL V [5] and CLUSTAL W [12]. The protein sequences used in our experiments are from BALiBASE, which is publicly available at <http://www-igbmc.u-strasbg.fr/BioInfo/BALiBASE>.  $K$ -group A\* ran on a 300Mhz Sun UltraSparc II workstation with two gigabytes of RAM. The *max\_trials* parameter described in Section 2.4 was set to 20, in order to limit the number of refinement iterations. The cost function is Dayhoff's PAM-250 matrix with quasi-natural gap costs. In all experiments, we used a gap-opening cost of 20 and a gap-extension cost of 8, and terminal gaps (i.e., gaps at the beginning or end of an aligned sequence) are penalized the same as non-terminal gaps.

Like any iterative-refinement algorithm,  $K$ -group A\* starts with an initial alignment that is created using any approximation method. In our implementation, an initial alignment is created using progressive profile alignment. Unlike CLUSTAL W, which can add only one sequence or one profile at a time to the progressive alignment, our approach to progressive alignment uses  $K$ -group alignment, which is able to add a maximum of  $K - 1$  sequences at a time. As a result, our initial alignment is often better, which makes it easier for the iterative refinement process to converge.

### 4.1. Comparison to OMA

OMA (Optimal Multiple Alignment) is a multiple sequence alignment tool developed by Reinert et al. [11] that uses A\* inside a divide-and-conquer approach in order to find optimal or close-to-optimal alignments for small numbers of sequences. We use it as a benchmark to assess the quality of the alignments produced by  $K$ -group A\*.

The cost-function settings above are exactly the same as those used in Reinert *et al.*'s experiments with OMA, except that the machine used in their study is a Sun Ultra Enterprise 450 with 400 MHz processors [11], which is faster than the processor in our machine. Since detailed computational results for OMA are available online at <http://bibiserv.techfak.uni-bielefeld.de/oma/>, we use these published results in our comparison.

Name	$L$	% id	OMA		K = 3		K = 4		K = 5	
			Cost	Secs	Cost	Secs	Cost	Secs	Cost	Secs
1aboA	80	15	10,674	973.64	10,680	0.34	10,674	9.67	10,674	199.67
1aho	67	44	9,807	6.06	9,816	0.32	9,829	0.91	9,807	0.82
1hfh	132	31	19,208	23.64	19,238	2.47	19,208	4.95	19,208	3.23
1idy	58	14	9,542	3.97	9,520	0.47	9,508	2.48	9,508	45.3
1krn	82	45	11,409	3.31	11,409	0.61	11,409	0.63	11,409	0.05
1pfc	117	28	17,708	19.96	17,711	1.13	17,708	1.95	17,708	3.81
1plc	99	46	14,205	4.96	14,195	0.64	14,195	0.95	14,195	0.11
2mhr	118	45	16,687	4.08	16,692	0.83	16,687	1.5	16,687	0.09
451c	87	27	13,364	200.28	13,365	1.82	13,378	15.14	13,364	74.69

**Table 1. Comparison of OMA and  $K$ -group A\* with  $K = 3, 4,$  and  $5$  on test sets of 5-sequence alignment from reference 1 of BALiBASE. Columns show the name of the test set (Name), the maximum length ( $L$ ) and the average percentage identity (% id) of the sequences, sum-of-pairs alignment cost (Cost), and CPU seconds (Secs).**

We used  $K$ -group A\* to align sets of 5 protein sequences from *reference 1* of BALiBASE, with different values of  $K$ . Our test goal was to evaluate the quality of approximation achieved by iterative  $K$ -group alignment with various numbers of groups, and to compare these results to results achieved by OMA, a state-of-the-art optimal alignment tool. Note that when  $K = N = 5$ , our algorithm finds an alignment with the minimum cost, i.e., an optimal alignment.

The results are shown in Table 1. One can see that 5-group A\* finds an optimal alignment for each and every test problem. For two problems (1idy and 1plc), it finds optimal alignments whose costs are even lower than the ones found by OMA.<sup>1</sup> For seven of the nine test problems, 4-group A\* finds optimal alignments. Although 3-group A\* only finds optimal alignments for two test problems, the sub-optimal alignments it finds are very close to optimal, and it finds two alignments with costs lower than those found by OMA.

As shown in the table, increasing the value of  $K$  can increase the running time of  $K$ -group A\* significantly, because the size of the state space increases exponentially with  $K$ , the number of groups. But even with  $K = 5$ , our  $K$ -group A\* runs much faster than OMA on all but one problem (1idy), for which OMA fails to find an optimal alignment. Note also that the timing results for OMA are on a machine with faster processors than ours. One of the reasons our  $K$ -group A\* algorithm runs faster is the restricted grouping strategy (described in Section 2.4), which lends itself easily to efficient implementation.

<sup>1</sup> The reason that OMA sometimes finds sub-optimal alignments is due to its use of a divide-and-conquer alignment strategy that does not necessarily preserve optimality. We also wrote a separate program that reads in these sub-optimal alignments produced by OMA and found that the cost of these alignments can indeed be reduced further.

## 4.2. Comparison to CLUSTAL V & W

CLUSTAL W is one of the most popular alignment tools available. It improves on its predecessor, CLUSTAL V, by using sequence weighting, position-specific gap penalties and weight matrix choice. As a result, the cost function of CLUSTAL W is very complex, which makes meaningful comparisons based on alignment cost difficult. For this reason, we compare  $K$ -group A\* to both CLUSTAL V and CLUSTAL W. Unlike CLUSTAL W, CLUSTAL V uses Dayhoff’s PAM-250 matrix.

As another way to minimize the issue of different cost functions, we compare the biological quality of the alignments found by CLUSTAL V, CLUSTAL W, and  $K$ -group A\*, using a standard test program called *bali\_score* that comes with BALiBASE. *Bali\_score* computes the percentage of correctly aligned residue pairs within core blocks of regions defined in an alignment annotation file created by the authors of BALiBASE. This provides a standard of comparison that is independent of any particular cost function.

We first compare the algorithms on 4- and 5-sequence test sets with different average percentage identities (% id) from reference 1 of BALiBASE. This makes it possible to include OMA in the comparison. The percentages shown in Table 2 are the sum-of-pairs (SP) scores computed by *bali\_score*. As we can see, all programs except CLUSTAL V find alignments with average SP scores above 90%. Surprisingly, and interestingly, 4-group A\* does not outperform 3-group A\* on average, although we earlier saw that 4-group A\* usually finds lower-cost alignments than 3-group A\*. This reflects the fact that finding lower-cost alignments does not necessarily translate into better (e.g., more biologically plausible) alignments. In many cases, the cause of a worse alignment has more to do with an inappropriate cost function than with a less efficient algorithm that fails to find a

Name	$N$	$L$	% id	OMA	CL. V	CL. W	K = 3	K = 4
laboA	5	80	15	56.4%	43.0%	77.2%	66.4%	56.4%
lad2	4	213	30	96.0%	96.0%	94.9%	96.0%	96.0%
laho	5	67	44	96.6%	97.7%	92.0%	96.6%	94.9%
lamk	5	254	49	99.5%	99.5%	99.6%	99.3%	99.5%
laym3	4	244	32	96.7%	96.4%	95.7%	96.7%	96.7%
lcsp	5	70	51	97.8%	92.0%	99.3%	97.8%	97.8%
lcsy	5	104	30	98.6%	98.6%	93.3%	97.9%	98.6%
lezm	5	308	60	93.1%	94.4%	95.0%	92.3%	93.1%
lfkj	5	110	44	98.2%	94.7%	97.1%	98.2%	98.2%
lgdoA	4	265	30	93.4%	87.0%	90.8%	93.4%	93.4%
lhfh	5	132	31	94.1%	92.7%	88.6%	94.1%	94.1%
lidy	5	58	14	59.2%	31.9%	56.9%	70.8%	56.9%
lkrn	5	82	45	99.2%	99.2%	99.2%	99.2%	99.2%
ldg	4	315	27	97.4%	94.3%	96.7%	97.4%	97.4%
lmrj	4	266	33	100.0%	99.4%	99.3%	100.0%	100.0%
lpfc	5	117	28	97.2%	97.2%	89.7%	97.2%	97.2%
lpgtA	4	212	26	49.5%	94.7%	100.0%	98.9%	98.9%
lpil	4	259	32	88.1%	82.8%	84.1%	88.1%	88.1%
lplc	5	99	46	94.1%	97.6%	95.3%	97.6%	97.6%
ltis	5	295	50	98.0%	94.2%	97.3%	98.0%	98.0%
lton	5	244	30	97.7%	84.1%	80.1%	95.7%	97.7%
2cba	5	259	26	92.9%	83.3%	87.5%	93.0%	92.9%
2fxb	5	63	51	94.3%	92.1%	95.0%	94.3%	94.3%
2mhr	5	118	45	99.0%	95.5%	99.6%	98.4%	99.0%
451c	5	87	27	66.7%	66.4%	64.9%	66.7%	69.2%
Average				90.1%	88.2%	90.8%	93.0%	92.2%

**Table 2. Sum-of-pairs score comparison of OMA, CLUSTAL V (CL. V), CLUSTAL W (CL. W), and K-group A\* with K = 3 and 4 on test sets from reference 1 of BALiBASE. Columns show the name of the test set (Name), the number ( $N$ ), the maximum length ( $L$ ), the average percentage identity (% id) of the sequences, and sum-of-pairs score in percentages.**

lower-cost alignment. This also provides a justification for using  $K$ -group A\* with small values of  $K$ , such as 3 or 4.

In this experiment, we found that 3-group A\* outperforms CLUSTAL V and CLUSTAL W by 4.8% and 2.2% on average SP score, respectively. The improvements are appreciable given the fact that the accuracy of CLUSTAL V (88.2%) and CLUSTAL W (90.8%) is already quite high.

Next, we compare  $K$ -group A\* and the CLUSTAL programs on more challenging instances of multiple alignment with larger numbers of sequences. The test sets are from reference 5 of BALiBASE, and Table 3 gives the results. Again, 3-group A\* outperforms CLUSTAL V and CLUSTAL W in terms of the average SP score; the degree of improvement is 9.8% and 2.1%, respectively.  $K$ -group A\* does particularly well on test sets with more than 10 sequences, such as S51, kinase2, and kinase3. We attribute this to  $K$ -group A\*'s improved ability to overcome CLUSTAL V/W's "once a gap, always a gap" problem, which becomes more pronounced as the number of sequences being aligned increases. Note that we cannot run OMA on most test sets of reference 5, because OMA has difficulties aligning more

than six or seven sequences with low similarities.

The running time of  $K$ -group A\* on reference 5 of BALiBASE is very fast. For example, it took only 159.4 seconds to solve kinase3, the largest problem in the reference set. Recall that  $K$ -group A\* uses iterative refinement to improve the final alignment. In all of our experiments,  $K$ -group A\* ran at least 20 iterations before termination. This means it took, at most, an average of 7.97 seconds to perform a single iteration for kinase3. For comparison, CLUSTAL W took 29.3 seconds to solve kinase3 on the same machine. In other words, CLUSTAL W is at least 3.68 times slower than a single iteration of 3-group A\* on this problem.

### 4.3. Discussion of results

Because CLUSTAL W adaptively chooses the distance matrix based on the average similarity of the sequences and penalizes gaps according to their positions in an alignment, its cost function is believed to be the best among all programs tested. Since OMA, CLUSTAL V, and  $K$ -group A\* all use a *fixed* cost function in all experiments, this gives

Name	$N$	$L$	% id	CL. V	CL. W	$K = 3$
left	8	314	19	34.8%	39.5%	32.0%
livy	7	441	36	100.0%	100.0%	87.4%
lpysA	10	313	25	63.4%	68.3%	63.2%
lqpg	5	498	35	95.0%	100.0%	100.0%
lthm1	11	231	32	58.9%	76.9%	88.9%
lthm2	7	229	38	93.5%	86.2%	93.5%
2cba	8	328	29	87.5%	91.3%	95.4%
S51	15	301	21	53.2%	93.1%	100.0%
S52	5	334	29	100.0%	100.0%	100.0%
kinase1	5	358	26	92.3%	92.3%	100.0%
kinase2	12	400	29	82.6%	89.4%	93.6%
kinase3	19	384	29	66.9%	83.4%	91.0%
Average				77.3%	85.0%	87.1%

**Table 3. Sum-of-pairs score comparison of CLUSTAL V (CL. V), CLUSTAL W (CL. W) and K-group A\* with  $K = 3$  on test sets from reference 5 of BAliBASE.**

CLUSTAL W a significant advantage. It also means the comparison results must be interpreted with care.

We believe that the comparisons of  $K$ -group A\* with OMA and CLUSTAL V are more objective, because one can be reasonably sure that most, if not all, of the improvement achieved by  $K$ -group A\* is due to its improved ability to find lower-cost alignments, which is the main focus of our paper. But to be a useful tool for biologists,  $K$ -group A\* must show its advantage over other popular alignment tools. This is the reason we chose to include CLUSTAL W in the comparison study. Algorithmically speaking, CLUSTAL W is no different from CLUSTAL V (they both use progressive pairwise profile alignment), and any improvements in profile alignment that CLUSTAL W introduces can also be incorporated into  $K$ -group A\*. That is, the performance of  $K$ -group A\* can be further improved, by incorporating any improvements that CLUSTAL W adds to CLUSTAL V. This is left for future work.

## 5. Conclusion

Alignment of multiple protein or DNA sequences is an important problem that presents many challenges for both computer scientists and biologists. This paper introduces a new method of computing gap-opening costs in profile alignment that makes it possible to implement a  $K$ -group generalization of A\* that uses quasi-natural gap costs. Computational results show that our implementation of the  $K$ -group A\* algorithm finds high-quality alignments in a reasonable amount of time, and that it outperforms OMA, a state-of-the-art optimal alignment tool, as well as the CLUSTAL family of approximate alignment tools.

**Acknowledgements** This work was supported in part by NSF grant IIS-9984952 and NASA grant NAG-2-1463.

## References

- [1] S. F. Altschul. Gap costs for multiple sequence alignment. *Journal of Theoretical Biology*, 138:297–309, 1989.
- [2] M. P. Berger and P. J. Munson. A novel randomized iterative strategy for aligning multiple protein sequences. *Computer Applications in the Biosciences*, 7:479–484, 1991.
- [3] O. Gotoh. Optimal alignment between groups of sequences and its application to multiple sequence alignment. *Computer Applications in the Biosciences*, 9:361–370, 1993.
- [4] S. K. Gupta, J. D. Kececioğlu, and A. A. Schaffer. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *Journal of Computational Biology*, 2(3):459–472, 1995.
- [5] D. G. Higgins, A. J. Bleasby, and R. Fuchs. CLUSTAL V: improved software for multiple sequence alignment. *Computer Applications in the Biosciences*, 9:189–191, 1992.
- [6] T. Ikeda and H. Imai. Enhanced A\* algorithms for multiple alignments: optimal alignments for several sequences and k-opt approximate alignments for large cases. *Theoretical Computer Science*, (2):341–374, 1999.
- [7] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8:615–623, 2001.
- [8] R. Korf and W. Zhang. Divide-and-conquer frontier search applied to optimal sequence alignment. In *17th National Conference on Artificial Intelligence*, pages 910–916, 2000.
- [9] M. Lermen and K. Reinert. The practical use of the A\* algorithm for exact multiple sequence alignment. *Journal of Computational Biology*, 7(5):655–671, 2000.
- [10] M. McNaughton, P. Lu, J. Schaeffer, and D. Szafron. Memory-efficient A\* heuristics for multiple sequence alignment. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-02)*, pages 737–743, 2002.
- [11] K. Reinert, J. Stoye, and T. Will. An iterative method for faster sum-of-pairs multiple sequence alignment. *Bioinformatics*, 16(9):808–814, 2000.
- [12] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.
- [13] J. D. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–2690, 1999.
- [14] T. Yoshizumi, T. Miura, and T. Ishida. A\* with partial expansion for large branching factor problems. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, pages 923–929, 2000.
- [15] R. Zhou and E. A. Hansen. Sparse-memory graph search. In *International Joint Conference on Artificial Intelligence (IJCAI-2003)*, pages 1259–1266, 2003.
- [16] R. Zhou and E. A. Hansen. Space-efficient memory-based heuristics. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-2004)*, pages 808–814, 2004.