An Empirical Study of Two Existing Global Iterative Multiple Sequence Alignment Methods: PRRP and *A**-DCA

Heidi Lam
University of British Columbia, Canada
Department of Computer Science
hllam@cs.ubc.ca

Ivan Minevskiy
University of British Columbia, Canada
Department of Computer Science
ivan@cs.ubc.ca

January 19, 2005

Abstract

Multiple sequence alignment of proteins or nucleic acids is an important aspect of many bioinformatics applications, but it is a difficult problem. In this paper, we report our evaluations of two known global iterative refinement methods from the literature using sum-of-pairs score as our cost function: PRRP and *A**-DCA. We evaluated our implementations and the programs by comparing our alignments with those provided in the BAliBASE reference sets. We found that PRRP and *A**-DCA were similar in their ability to align equidistance and equi-homology sequence sets, and to handle extension/insertion of blocks of sequences. Using simulated evolution, we also found that the performance of the two programs were similar in handling evolutionary changes. In terms of time and memory requirements, we found that PRRP was more time intensive, while *A**-DCA required more memory. These results are encouraging for the relatively new *A**-DCA, since PRRP has been found to be one of the best iterative algorithms.

1 Introduction

Multiple sequence alignment is the arrangement of several sequences such that homologous residues are aligned in the same column. The aim of the arrangement is to produce a representation that reflects on the relationship of the sequences. This technique can be applied to nucleic acids or proteins. The definition of "homology" depends on the context and criteria for the alignment. In the structural sense, the aligned residues should ideally occupy similar three-dimensional positions in the structures. In the phylogenetic

sense, the aligned residues should originate from the same residue in the common ancestor.

Multiple sequence alignment is an important tool in bioinformatics and molecular biology. It can be used to improve predictions of secondary and tertiary structures of new sequences by predicting the role a residue can play. In phylogenetic analyses, multiple alignments are used to construct the evolutionary history based on a homologous set of biological sequences, and to identify evolutionally conserved motifs and domains. This technique also plays an important role in the demonstration of homology between new sequences and existing families; identification of diagnostic patterns for families; and in polyermerase chain reaction primer design.

Despite the many applications, automatic multiple alignment is a difficult task. To align two sequences, the Needleman-Wunsch algorithm produces optimal global alignment with gaps [Needleman & Wunsch, 1970], and the Smith-Waterman algorithm locally align subsequences [Smith & Waterman, 1981]. Both of these algorithm use dynamic programming to optimize the scores based on substitution matrices. Unfortunately, when these algorithms are generalized to multiple sequences, the time complexity increases from $O(L^2)$ to $O(2^N L^N)$, and the memory complexity increases from $O(L^2)$ to $O(L^N)$[1]. Even though the algorithm guarantees a mathematically optimal alignment, it is obvious that it would be unsuitable for aligning long and/or large sequence sets for practical uses. Since the problem of computing optimal multiple sequence using the sum-of-pair scores (SPS) (see §2.2) has been demonstrated to be NP-complete [Wang & Jiang, 1994], various heuristic strategies have been developed to produce reasonable alignments in reasonable time.

In our work, we implemented two iterative refinement methods for multiple sequence alignment using the SPS based on our literature review: PRRP and *A**-DCA. Since PRRP has been well characterized, we used it as a benchmark to evaluate the performance of *A**-DCA, another global iterative algorithm. To test our implementations and the algorithms, we empirically studied our programs with the BAliBASE reference set where we aligned the reference sequence sets and compared our scores with published results. We also carried out two tests using simulated evolution as an attempt to further characterize the two programs' ability to handle evolutionary changes.

The report is arranged as follows. We first review the related work of multiple sequence alignment in the areas of optimization strategies and cost functions. We then describe in detail the two selected algorithms based on literature, and our implementation of the two algorithms. This is followed by our empirical study section where we report and analyze our alignment results using the BAliBASE reference set, and simulated evolution. We then conclude with a discussion of our results, and point out some directions for further investigation.

---

[1] Assuming the sequences are of comparable length L

2 Related Work

In this section, we survey related literature in the field of multiple sequence alignment. Basically, the alignment method has two components: (1) the optimization strategy, to align the input sequences, and (2) the cost function, to determine the quality of the resultant alignment. We will first look at optimization strategies in the literature, followed by those of the cost functions.

2.1 Optimization Strategies

In our discussion of optimization strategies, we follow the grouping scheme proposed by [Thompson et al., 1999], where two main groups of strategies are identified: progressive and iterative (Figure 1).

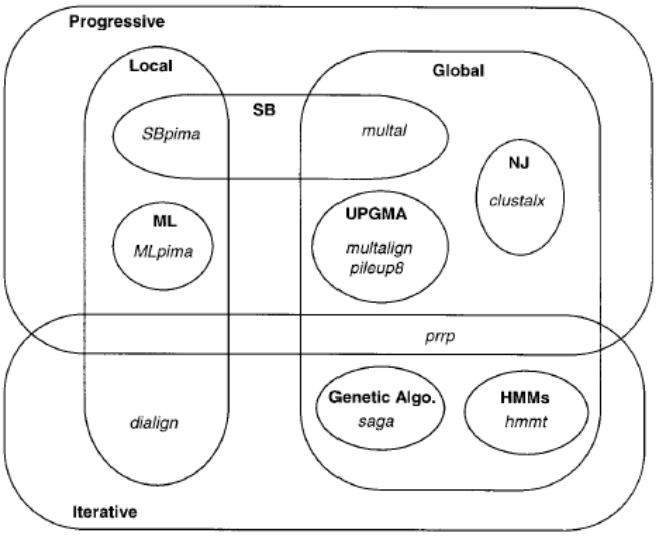

Figure 1. Schematic showing the relation between the different alignment programs and algorithms, from [Thompson et al, 1999].

2.1.1 Progressive methods

Our review here is based on [Durbin et. al, 1998, p. 143-149, Notredame, 2001 and Thompson, 1999]. The progressive alignment approach has been the most popular approach traditionally. The basic idea is to construct the alignment two sequences at a time using standard pairwise alignment techniques. Once aligned, the alignment is fixed and consolidated to a single entity. A number of variations in the algorithm have been proposed after the first step where the first two sequences are aligned. The simplest way to align the rest of the sequences is to add individual sequences one-by-one to the intermediate alignment successively in some pre-defined order. A slightly more complex algorithm aligns sequences that are closer together first to build subfamilies, which are eventually aligned until all the sequences are aligned. The distances between sequences are estimated based on a guide tree whose leaves represent sequences, and whose nodes represent alignments. The root of the tree thus represents a complete multiple alignment. Guide trees are constructed based on distances calculated from pairwise alignments

between all pairs of sequences.

Despite being simplistic and low-cost, progressive methods do not offer any guarantee with respect to the optimality of the resulting alignments. Since once made, the alignment cannot be changed, the performance of progressive methods depends heavily on the choice of sequence alignment order. Intermediate alignments may not necessarily be correct since they were made with partial information of the sequences. Early mistakes are therefore not corrected, and may propagate by inducing more mistakes in further intermediate alignments. Progressive methods are limited by short-term constraints that may lead to a non-global solution.

2.1.2 Iterative Methods

To overcome this problem, a more global approach is required. Iterative methods are heuristic methods that refine an initial suboptimal alignment through a series of cycles until no more improvements can be made. We follow Notredame's classification scheme in the following discussion, where the more traditional stochastic iterative methods (e.g., simulated annealing and genetic algorithm) are considered as "stochastic", and those using dynamic programming as "non-stochastic", even though the latter may contain elements of randomness in their algorithms [Notredame, 2001].

Non-stochastic iterative methods use dynamic programming to modify intermediate suboptimal alignments. The idea is to correct the mistakes that may arise in progressive methods. Starting with an initial alignment, one sequence (or a set of sequences) is taken out to be realigned to a profile of the remaining aligned sequences. This step is repeated until the alignment is constant. Different algorithms select the sequence(s) to be realigned differently. In AMPS, the sequences are chosen according to their input order, and are realigned one-by-one [Barton & Sternberg, 1987]. In Berger & Munson's algorithm, the sequences are chosen randomly, and realigned to a profile of the other aligned sequences by profile-sequence alignment [Berger & Munson, 1987]. Gotoh proposed a doubly nested iterative algorithm that optimizes weighted SPS with affine gap penalties [Gotoh, 1996]. The weights (outer loop) and the alignments (inner loop) are optimized simultaneously. The iteration stops when the weights converge.

Stochastic methods modify the suboptimal alignment randomly. Simulated annealing (SA) is based on statistical mechanics of the physical annealing process to solids (e.g., [Kim et al, 1994] and [Ishikawa, 1993]). Modifications to the initial solution are made randomly, and the change is accepted if the new solution is superior to the old. Otherwise, it is accepted or rejected based on a probability that depends on the "temperature", and the level of disimprovement. As the process progresses, the "temperature" reduces, and the acceptance constraint tightens. Genetic algorithms (GAs) are another type of stochastic method. They are based on the theory of evolution where mutation and cross-over events are introduced to a population randomly. The population is then subjected to selective pressure, and alignments will die or survive depending on their "fitness". An example of MSA GA is SAGA [Notredame & Higgins, 1996].

2.2 Cost Functions

Cost functions are mathematical functions that define the quality of the alignment. It is sometimes called "Objective Function" since it defines the objective of the optimization. Ideally, the optima of the mathematical function should correspond to the biologically optimal, but that is rarely the case [Notredame, 2002]. In order to define an ideal cost function, the designer would need to incorporate the sequences' structure, function and evolution history. Since this information is rarely available, optimization is often based on sequence similarity.

Cost functions can be roughly divided into two categories: those that do not assume a phylogenetic relationship among the sequences to be aligned, and those that do. Examples of the first group include sum-of-pair, star, consensus and minimum entropy scores, while the latter group includes tree, weight sum-of-pair and maximum likelihood scores. Newer cost functions include a statistically based Gibbs sampler (measures *p*-values) [Lawrence et al., 1993], and a consistency-based function called COFFEE (measures consistency between sequence and library) [Notredame et al, 1998]. In this review, we will focus on the sum-of-pair score (SPS).

SPS is defined as the sum of all pairwise scores between all pairs of residues in the columns of the multiple alignment. The cost functions rely on the substitution matrix that gives a score to each substitution of residue. Gap penalties incur costs to deletions and insertions. The solution with the lowest cost for substitution, and deletion/insertion is then the optimal solution. The SPS of alignment *A* of length *I* with *N* nucleotide or amino acid sequence, SP(*A*), is defined as,

$$SP(A) = \sum_{j=2}^{N} \sum_{k=1}^{j-1} S_{j,k} \qquad \textbf{[Eqn 1]}$$

where $S_{j,k}$ is the score associated with the pairwise alignment between the *j*th and the *k*th sequences within *A*. Gap penalty scores can either be linear,

$$\gamma(g) = -gd \qquad \textbf{[Eqn 2]}$$

or accounts for gap-opening (*d*) and gap-extension (*e*),

$$\gamma(g) = -d - (g-1)e \qquad \textbf{[Eqn 3]}$$

These values (*d* and *e*) are set empirically, and may vary from one set of sequences to another. Since substitution matrices are adapted from a large number of sequences, they may not be specific enough for the problem at hand, especially when the evolutionary distances between the sequences are not evenly distributed. To correct the biased contributions of individual members of a family of sequences, weighting techniques are used to modify SPS [Altschul et al., 1989].

3 Selected Algorithms

In this paper, we report our evolutions of two iterative refinement methods using the SPS

function for further study: PRRP and *A**-DCA. Our rationale for our choice of algorithm is as follows. *A**-DCA is the first iterative alignment algorithm that provably improves its result up to optimality while being able to quickly compute good intermediate alignments. PRRP was evaluated using BAliBASE in [Thompson et al, 1999], and performed very well in most of the reference sets except for N/C-terminal extensions. In contrast, *A**-DCA has only been tested against BAliBASE reference 1 in [Reinert, 2000]. PRRP would thus be a good benchmark for *A**-DCA.

3.1 PRRP

Our implementation of PRRP is based on [Gotoh, 1996, 1995]. As mentioned in §2.1.2, PRRP is a doubly nested algorithm that optimizes weighted SPS with affine gap penalties where the weights and the alignments are optimized simultaneously (Figure 2).

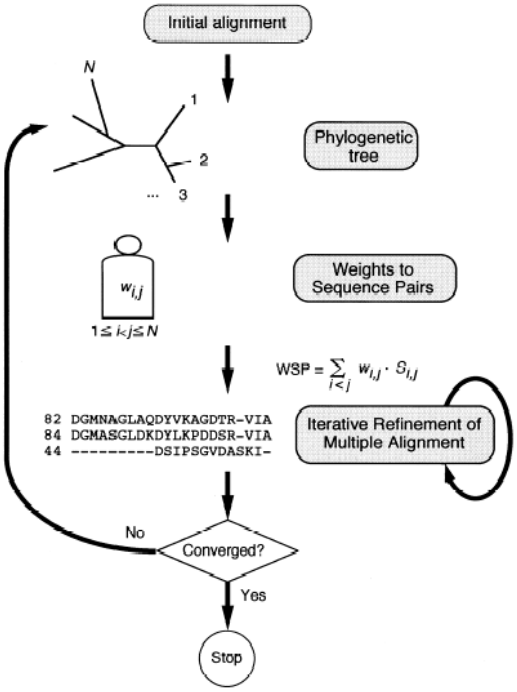

Figure 2. Schematic diagram of PRRP. From [Gotoh, 1996].

There are four basic steps in this algorithm: (1) phylogenetic tree construction, (2) weight calculations, (3) iterative multiple sequence alignment, and (4) weight comparison. The algorithm stops when the total weight of the tree converges.

3.1.1 Phylogenetic Tree Construction

Starting with an initial suboptimal alignment, the first step in the algorithm is to construct a phylogenetic tree. As in [Gotoh, 1996]'s version, we constructed the trees using UPGMA.

UPGMA (unweighted pair group method using arithmetic averages) is a clustering

procedure proposed by [Sokal & Michener, 1985]. We implemented the algorithm based on the outline in [Durbin et al, 1998, pp. 166-168]. Briefly, sequences are initially put into individual clusters. To form a node in the tree, the two closest clusters are merged together based on their relative distances. This process is repeated until only two clusters remains, where the last two clusters will form the root of the tree. Since the next stage (weight calculation) requires an unrooted tree, we terminate the algorithm at this stage instead of amalgamating the last two clusters to form the root of the tree. The modified algorithm we adapted is then as follows (based on [Durbin et al, 1998], pp. 166):

Initialization

Assign each sequence i to its own cluster C_i.

Define one leaf of tree T for each sequence, and place at height zero.

Iteration

Determine the two clusters i, j for which d_{ij} is minimal (If there are several equidistant minimal pairs, pick one randomly). The distance d_{ij} between two clusters C_i and C_j is defined as,

$$d_{ij} = \frac{1}{|C_i||C_i|} \sum_{p \text{ in } C_i, q \text{ in } C_j} d_{pq}$$

and the pair-wise distance d_{pq} is based on the is estimated by Kimura distance $(-1)*log(1 - d - 0.2*d^2)$, where d is the fraction of different amino acids. d is defined as the number of positions in the sequence that are difference, divided by the length of the sequence.

Define a new cluster k by $C_k = C_i \cup C_j$, and define d_{kl} for all l based on

$$d_{kl} = \frac{d_{il}|C_i| + d_{jl}|C_j|}{|C_i| + |C_j|}$$

Define a node k with daughter nodes i and j, and place it at height $d_{ij}/2$.

Add k to the current clusters and remove i and j.

Termination

When only two clusters i and j remain.

3.1.2 Weight calculations

Weight calculations are based on [Gotoh, 1995] and [Gotoh, 1994]. There are two components in this step: calculate pair-wise weights, and calculate the weight of the entire tree.

3.1.2.1 Pair-wise weight calculation

Pairwise weight calculation is based on the three-way method described in [Gotoh, 1995]. In cases where there are more than three nodes, the unrooted phylogenetic tree obtained by UPGMA in §3.1.1 is considered as a collection of partly overlapping three-way tress having the same internal node as the original tree (Figure 3). The branch lengths of the tree are determined in a manner similar to determining the 'synthetic resistance' of an

electric circuit.

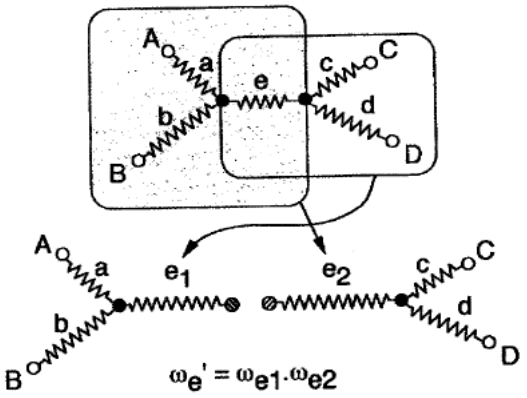

Figure 3. Schematic presentation of procedures used in the three-way method. The original tree is decomposed into overlapping three-way trees (in boxes), where the circles represent the nodes of the tree, and the zigzag lines are edges between nodes. Taken from [Gotoh, 1995].

The w_e value obtained from the three-way tree is directly assigned to that of the original tree if one of the termini of *e* is a leaf of the tree. When both termini of *e* are internal nodes of the tree, the weight of the edge *e* shared by the two trees is estimated by the product of the weights of the flanking imaginary three-way trees, w_{e1} and w_{e2}. w_{e1} and w_{e2} can be calculated as,

$$w_{e1} = \sqrt{ab(e_1 + a)(e_1 + b)/e_1(a + b)FS}$$

$$w_{e2} = \sqrt{cd(e_2 + c)(e_2 + d)/e_2(c + d)FS}$$

where *F* is the normalization factor, and is found to be 1.1 experimentally for better estimations of the true weights of the tree (for further discussion of weight estimation, see [Gotoh, 1995]), and *S* is defined as $(ae_1 + e_1b + ab)$ in w_{e1} and $(ce_2 + e_2d + cd)$ in w_{e2} calculations.

The three-way method is therefore as follows:

```
Double three_way (e){
    If one of the termini of e is a leaf of T
        return √(cd(e2 + c)(e2 + d)/e2(c + d)FS) ;
    else return three_way(e.left) * three_way(e.right);
}
```

3.1.2.2 Tree weight calculation

Once the pair-wise weights are determined, the total weight of the tree can be calculated using an algorithm proposed in [Gotoh, 1995]. Before we can discuss the algorithm in detail, we need to define a few parameters.

A is a multiple sequence alignment of length *I*, composed of *N* residues. The sequences composed of residues that can either be nucleic acids or proteins. Here we use the set ***X*** to represent the set of possible residue type. In the case of nucleic acids, the set ***X*** = {A, T, G, C, -} where "-" represents a gap. *A* is the same alignment used to construct the phylogenetic tree *T*. A_v the sequence in the alignment *A* corresponding to node *v*.

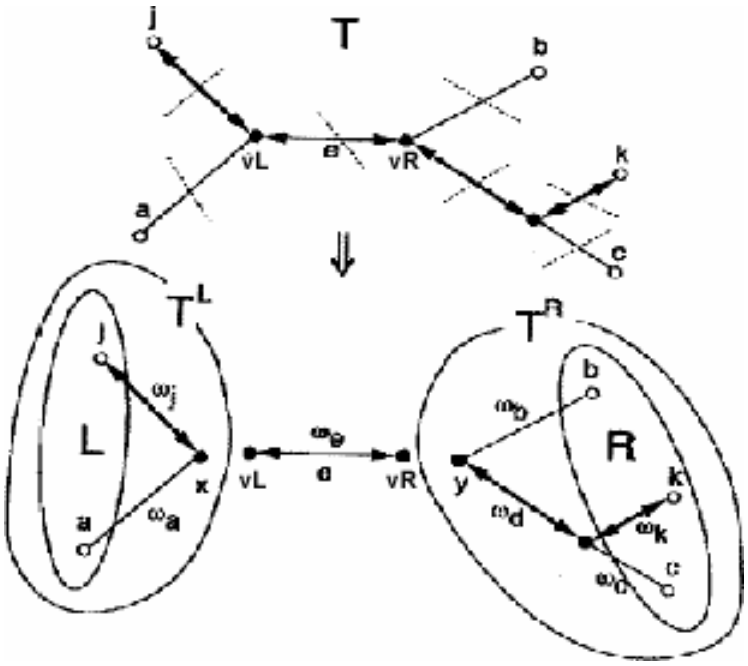

Figure 4. Partition of tree *T* into two sub-trees, T^L and T^R, and an edge *e*. This partitions the tree into two groups, *L* and *R*. Taken from [Gotoh, 1995].

First, an edge between two nodes is chosen in the unrooted tree obtained by UPGMA in §3.1.1, and is denoted as $e = (v_L, v_R)$, where v_L is the node to the left of *e*, and v_R is the node to the right of *e* (Figure 4). A temporary root *v* is created, with v_L and v_R as its sons. The wps2(*v*, ***P***, ***Q***) function is called recursively to calculate the total weight of the tree:

```
Double wsp(v, P, Q)
{
    if v is a leaf of the tree
        P = w_v * Profile_of_type_P(A_v);
        Q = w_v * Profile_of_type_Q(A_v);
        return (0);

    Score_left = wsp(v.left, P_left, Q_left);
    Score_right = wsp(v.right, P_right, Q_right);
    Score_here = w_v * P_left · Q_right;
    P = w_v * (P_left + P_right);
    Q = w_v * (Q_left + Q_right);
    return (Score_left + Score_right + Score_here);
}
```

In the function, w_v is the weight assigned to the edge between *v* and its parent. Initially, since the node *v* is the root of the tree, w_v is defined as unity. The algorithm divides the

tree into two parts, the left (L), and the right (R) with respect to node *v*. ***P*** and ***Q*** are profile vectors that are defined at each position *i* of the sequence with length *I*. For example, for the left partition,

$$P_i^L = W^L \cdot p_i^L$$

where $\boldsymbol{W^L}$ is a list of weights between all the nodes in the left side of *v*, to v_L, the left side of the edge *e*, and p_i^L is a list of profiles of all the residue types in the left side of the tree. For residue type *x* at position *i*, this profile is defined as,

$$p_{x,i}^L = \sum_{y \in X} d(x,y) f(y,i)$$

where *d(x, y)* is the dissimilarity between the two residues *x* and *y*, and *f(y, i)* is the frequency of residue *y* at position *i*.

As for ***Q*** (for the right partition as an example),

$$Q_i^R = f_i^R \cdot W^R$$

where $f_{x,i}^R$ is the frequency vector for all the frequencies of all the residues at position *i* in the right side of *v*, and W^R is the weights between v_R and all the nodes in the right of *v*. The derivation of ***P*** and ***Q*** is enclosed in the Appendix.

3.1.3 Iterative multiple sequence alignment

We used the algorithm by Berger and Munson to align the sequences [Berger & Munson, 1991]. Briefly, the *n* sequences are randomly partitioned into two subsets (S_1 and S_2), and they are aligned as a group to each other using an extended Needleman-Wunsch algorithm. The alignment is terminated when the SPS does not change in one search step.

The algorithm we used is as follows,

```
iter_align(Sequences S)
{
    Sequences S_new = S;

    // see if the scores converges
    while (Score(S) < Score(S_new))
    {
        // select partition to divide the n initially aligned sequences
        // into two groups chosen among (2^n-1)-1 possibilities
        n_partition = rand()*(2^n-1)-1

        // partition S into two groups of sequences
        S1 = S(1.. n_partition);
        S2 = S(n_partition+1..n);
```

```
            // remove global gaps
            global_gap_remove(S1);
            global_gap_remove(S2);
            // align subgroups
            S_new = ext_NW_align(S1, S2);
            S = S_new;
        }
    }
```

Two extensions are applied to the Needleman-Wunsch algorithm:

1. Similarity scores *s(i, j)* are defined only for those cells in the matrix representing fully occupied alignment positions. They are calculated as the sum of the pairwise scores between all residues aligned at position *i* in group S_1, and *j* in group S_2. In positions where gaps are present, the score is 0.

2. Any transition in the matrix from a non-gap region (unshaded areas in Figure 5) into, or beyond a region with gaps in the prealigned sequences (shaded areas in Figure 5) opens a gap in the global alignment, and a gap penalty of is added to the score (e.g. arrows 1 and 2 in Figure 5). If the transition is from inside a gap region into, or beyond a gap region (e.g. arrows 3 and 4 in Figure 5), then no penalty is incurred for the transition.

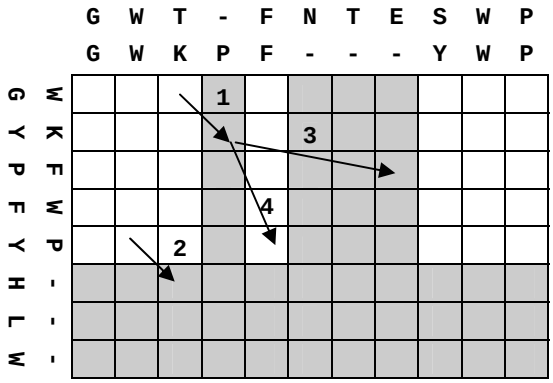

Figure 5. Sample matrix used by the extended Needleman-Wunsch algorithm.

3.1.4 Checking for weight convergence

The last step in the algorithm compare the total weight of the newly constructed tree with the previous one. If the new score is not an improvement over the old, then the algorithm terminates; if it is, it returns to step 1 and construct a phylogenetic tree based on the newly aligned sequences.

3.2 *A**-DCA

Our implementation of *A**-DCA is based on [Reinert, 2000, 2004] where the A^* algorithm is used to produce a more efficient solution than previous exact algorithms for finding the shortest path in a graph. In our case, an alignment of the *N* sequences is interpreted as a path in a *N*-dimensional grid graph. The *A** algorithm computes a

shortest path in the graph. Obviously, it is not feasible to compute such path in the full grid graph. Thus, the *A** algorithm uses a priority queue in which new edges are only inserted if a potentially optimal path might pass through them. Whether to insert an edge into the priority queue can be determined by using an upper bound on the cost of an optimal alignment, which is computed with the Divide-and-Conquer Alignment algorithm (DCA) [Stoyle, 1998]. The *A** algorithm and the DCA thus form an iterative combination: the DCA is successively called with increasing value of a special stop length parameter, where, at each step, the values of the corresponding partial alignments from the previous step are used to compute an upper bound for the computation of an optimal alignment using *A**.

3.2.1 The *A** algorithm

An alignment of the N sequences of length I is interpreted as a path in a N-dimensional grid graph G. The graph has nodes V,

$$V = \{v = (v[1], v[2], \ldots, v[N]) : v \in \mathrm{N}^N, 0 \le v[i] \le I, 1 \le i \le N\},$$

and edges E,

$$E = \{(p,q) : p,q \in V,\ p \ne q,\ q - p \in \{0,1\}^N \setminus \{0\}^N\},$$

where "–" is the element-wise subtraction, "$\{\}^N$" and "\" are usual set operations

Each node holds information about the edge through which an optimal path from source arrives to the node and about the cost (distance) of this path. By default a node has infinite distance to the source. A node also stores a position per every sequence (Figure 6).

Each edge stores its cost and the information about two nodes at its ends. An edge corresponds to a column in a multiple alignment, so its cost is a sum-of-pairs score of the corresponding column.

The purpose of this algorithm is to find the shortest path in G, starting from a source $s = (0,0,\ldots,0)$, and ending in a sink $t = (I,I,\ldots,I)$. Each path starting in s and ending with an edge e connecting the nodes (i-1) and i corresponds to one possible alignment of the sequences from the beginning to position i, and each path starting from the same node to the last node (i.e. between (i, t)) corresponds to an alignment from positions i+1 to I. The cost of an edge is the SP cost of the alignment column corresponding to the edge. Figure 6 shows a graph corresponding to an alignment of 2 sequences of length 2 each.

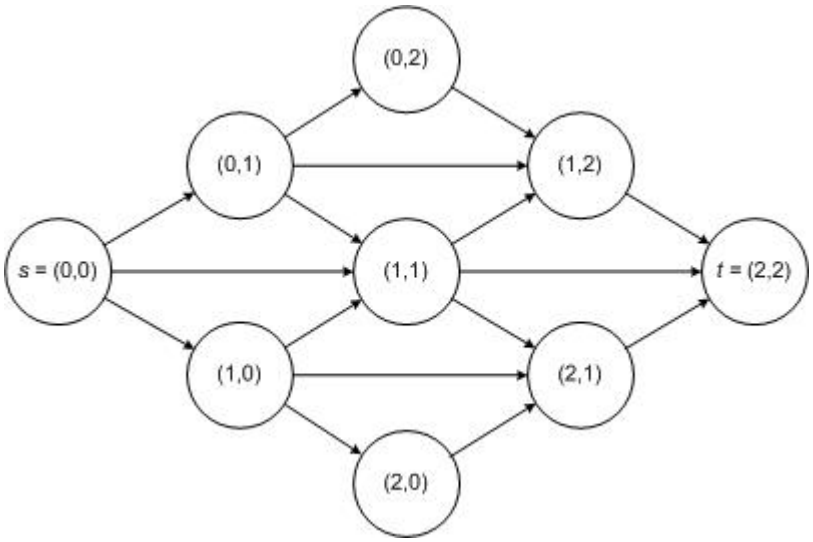

Figure 6. Graph corresponding to an alignment of 2 sequences of length 2 each. Values in brackets are positions in the corresponding sequences.

To calculate the shortest path, edges with costs below a certain upper limit *U*, which is obtained from a DCA alignment score, are added to a priority queue *Q* using the following pseudo-code:

```
MSA() {
    Node u,v;
    Edge e;
    int c;
    PQueue Q = new PQueue();  // priority queue stores references to Node objects

    s.dist = 0;    // s = source node
    Q.insert(s);
    while (not Q.isEmpty()) {
        u = Q.dequeueMin();
        for (all adjacent edges e of u) {   // 2^N-1 edges
                v = e.getTarget();
                c = u.getDistanceToSource() + e.getCost() + L(v);
                if (c < v.getDistanceToSource()) {
                        v.setDistanceToSource(c); // set distance of the path from v to source s
                        v.setOptimalEdge(e);  // set edge which connects v to the optimal path
                }
                if (c <= U) Q.enqueue(v);
        }
    }
}
```

where

L(Node v) = $\sum_{1 \le i < j \le k} D^{r}_{i,j}\left[v[i], v[j]\right]$, and

N is the number of sequences

v[*i*] is position in sequence s_i at node *v*

$D^{r}_{i,j}$ is the reverse distance matrix for sequences s_i and s_j. It can be computed with the Needleman-Wunsch algorithm which starts in the lower right corner of the matrix (the

"usual" forward matrix is computed starting in the upper left corner)

3.2.2 The DCA

DCA is used to determine whether to insert an edge into *Q*. In the DCA, the sequences are recursively cut at certain positions near to their center until they are of a length short enough for the exact alignment procedure. The DCA has a special stop length parameter *Z*, such that the recursion stops if the maximal length of a sequence drops below it. An alignment of the complete sequence is then the concatenation of the prefix alignment with the suffix alignment.

The pseudo-code is as follows,

```
DCA (s1, s2, …, sk, Z){
    if max{l1, …, lk} ≤ Z
        return MSA()
    else return (concatenate( DCA(α1^ĉ1, α2^c2, ..., αn^cn, Z), DCA(σ1^ĉ1, σ2^c2, ..., σn^cn, Z) )
}
```

where $\hat{c}_1 := \left\lceil \frac{l_1}{2} \right\rceil$

$(c_2, c_3, ..., c_n) := Copt((s_1, s_2, ..., s_n), \hat{c}_1)$

l_i := length of sequence s_i

α_i^c := prefix of sequence s_i : $s_i[1..c]$

σ_i^c := suffix of sequence s_i : $s_i[(c+1)..l_i]$

MSA() := an exact alignment algorithm defined in section 3.2.1

The *Copt* subroutine calculates the optimal cut positions in the sequences $s_2, s_3, \ldots, s_k$ and is as follows.

$C_{\min} := +\infty$

for $c_2 := \left\lceil \frac{|s_2|}{2} \right\rceil, \left\lceil \frac{|s_2|}{2} \right\rceil - 1, \left\lceil \frac{|s_2|}{2} \right\rceil + 1, \ldots, 2, I-1$ do

  if $C(\langle \hat{c}_1, c_2 \rangle) < C_{\min}$ then

    for $c_3 := \left\lceil \frac{|s_3|}{2} \right\rceil, \left\lceil \frac{|s_3|}{2} \right\rceil - 1, \left\lceil \frac{|s_3|}{2} \right\rceil + 1, \ldots, 2, I-1$ do

      if $C(\langle \hat{c}_1, c_2, c_3 \rangle) < C_{\min}$ then

        .
        .

          for $c_n := \left\lceil \frac{|s_n|}{2} \right\rceil, \left\lceil \frac{|s_n|}{2} \right\rceil - 1, \left\lceil \frac{|s_n|}{2} \right\rceil + 1, \ldots, 2, I-1$ do

            if $C(\langle \hat{c}_1, c_2, \ldots, c_k \rangle) < C_{\min}$ then

              $C_{\min} := C(\langle \hat{c}_1, c_2, \ldots, c_k \rangle)$

              $c_{\min} := \langle \hat{c}_1, c_2, \ldots, c_k \rangle$

return $c_{\min}$

Here, a set of cut positions for sequences $s_1, s_2, s_3, \ldots, s_k$ is denoted by $\langle \hat{c}_1, c_2, ..., c_k \rangle$. $\hat{c}_1$ is provided as an argument to *Copt*.

3.2.3 Putting it together

The DCA is called with a stop length *Z* at which the recursion stops. Obviously each DCA alignment is a heuristic alignment giving an upper bound *U*. The better the upper bound *U*, the better our branch-and-bound algorithm (i.e., *A**) works. On the other hand, we would like to stop the DCA recursion early, since then we can expect a near to optimal alignment while the computation time increases due to the larger optimal alignments to be computed.

This motivates an iterative combination of both DCA and the optimal alignment procedure (i.e., *A**): Call the DCA with a small value of *Z*. For the small alignments, we first compute the optimal branch-and-bound alignment using *A**. The upper bound *U* is infinite for this run, so all nodes of a graph are visited. That is why we want to cut sequences into the shortest slices, so to have the smallest graphs to work with at the beginning.

The resulting alignment is now a rather good heuristic alignment for the combined block. Hence its score can be used as an upper bound for the branch-and-bound algorithms that can correct the heuristic alignment into an optimal one.

We can continue this strategy until we reach the last division, or we can stop at any point of this procedure and have a heuristic alignment of quality that is at least as good as the original DCA. The longer we wait the better is the alignment, until we reach the optimal alignment. The way the entire algorithm works is illustrated schematically in Figure 7.

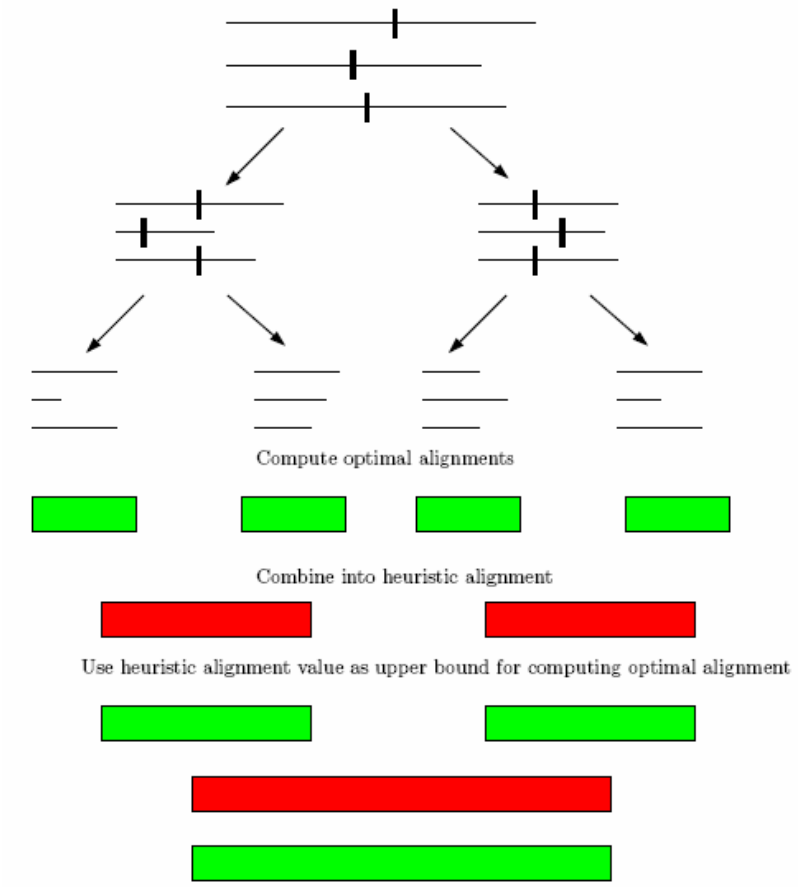

Figure 7. A schematic illustration of the *A-DCA algorithm.**

3.3 Implementation

We used Java 2 Standard Edition 5.0 and the Eclipse 3.1 development environment to implement both algorithms. For alignments of protein sequences, we used the PAM250 score tables. For DNA sequences, we used a simple score table which gives 2 for a perfect match, 1 for a transition and 0 for a transversion. The gap penalty was set to be 8, and we did not incur a gap-to-gap alignment penalty. The stop length Z was equal to 10.

4 Empirical Evaluation

All alignments were performed under SunOS 5.9 on a Solaris 9 SPARC with 8 processors and 32 GB RAM. The maximum memory size that our applications could access was 3.6 GB. Successful *A**-DCA trials (i.e., those with sufficient memory) took about 2 hours. Failed *A**-DCA trials could take up to 10 hours until memory ran out. In those cases, we had to terminate the programs before the optimal alignments were obtained due to the limitation of memory. Generally, PRRP trials required less memory than those of *A**-DCA, but were of slightly longer run-time. Trials successful in *A**-DCA took up to 2.5 hours in PRRP. Due to the time constraints, in PRRP we perform only those trials that were successful in *A**-DCA.

We used two approaches to evaluate the quality of the alignments: the BAliBASE reference sets, and simulated evolution. In this section, we will discuss both of these approaches and the results in detail.

4.1 Evaluation using the BAliBASE references

In order to objectively evaluate the quality of the alignments, we used the BAliBASE benchmark alignment database in a manner similar to [Thompson et al., 1999]. We used both PRRP and *A**-DCA to align each file in the reference sets. The test alignments were then compared to the reference alignments using two sum-of-score measurements. Since there are at least 12 files for each reference set, we believe we have enough trials for each reference set to test the algorithms, even though one component (the Berger & Munson alignment) of PRRP is non-deterministic. We therefore only ran each file once. The results of these trials were analyzed using non-parametric statistics.

The database contains 142 reference alignments divided into five reference sets:

Reference	Member	Evaluation Criteria
1	A small number of equidistant sequences of similar length	Benchmark, also effects of sequence length and % ID
2	A closely related family of sequence with >25% ID, with up to three orphans (<20% ID)	Stability of alignment when orphans are introduced; Quality of the alignment of the orphan sequences
3	Up to four families with <25% ID between any two sequences from different families	Ability to align divergent families
4	Sequences with large N/C-terminal extensions	Ability to identify the presence of extensions
5	Sequences with insertions	Ability to identify the presence of insertions

Table 1. BAliBASE references. %ID = residue identity between the sequences

Each reference set contains a number of alignments:

Reference	Short (<100 residues)	Medium (200-300 residues)	Long (> 400 residues)
1: <25 %ID 20-40 %ID >35 %ID	7 10 10	8 9 10	8 12 8
2	9	7	7
3	5	3	5
4	12		
5	12		

Table 2. BAliBASE reference sets: total number of alignments in each set

Two different scores are used to estimate the quality of the alignments: the sum-of-pairs scores (SPS), and the column score (CS).

SPS is used to determine the extent to which the programs succeed in aligning some, if not all, of the sequences when compared to the reference *r*. Given an alignment of length *I*, consisting of *N* sequences, and a reference alignment of length I_r, also of *N* sequences, SPS can be calculated column-by-column in the following manner,

$$SPS = \frac{\sum_{i=1}^{I} S_i}{\sum_{i=1}^{I_r} S_{ri}}$$

where S_i and S_{ri} are the score for the *i*th column in the test and reference alignment respectively, and is defined as,

$$S_i = \sum_{j=1, j\neq k}^{N} \sum_{k=1}^{N} p_{ijk} ,$$

where $p_{ijk} = \begin{cases} 1 & \text{if the residue is aligned with each other in the reference alignment, and} \\ 0 & \text{otherwise} \end{cases}$

CS is a binary score to test the programs' ability to align all the sequences correctly. As in SPS, it is also calculated column-by-column. For the *i*th column, the score $C_i = 1$ if all the residues in the column are aligned as in the reference alignment, and 0 otherwise. The total CS is therefore,

$$CS = \frac{\sum_{i=1}^{I} C_i}{I}$$

As seen in Table 2, each reference set contains at least 12 alignments. In order to characterize and differentiate these score populations, statistical analysis was used. The scores were analyzed using non-parametric statistics since their distribution has been reported to be non-normal [Thompson et al., 1999]. To describe a population of scores, we reported median and interquartile range as a measure of dispersion. Friedman tests were used to test systematic pattern for algorithm ranks [Friedman, 1937], and Wilcoxon signed rank tests was be used to determine if changes lead to a significant difference between paired scores [Wilcoxon, 1947].

All statistical calculations were performed using version 6.5.1 of Mathworks MatLab software.

4.2 BAliBASE Results

In this section, we present our BaliBASE reference alignment results using our implementation of PRRP and *A**-DCA. We first compare our results with published data to check the correctness of our implementations. We then proceed to references 2 to 5 of the BAliBASE reference sets.

4.2.1 Reference 1: sequence length and homology

This test is designed to study the effect of sequence length and homology on the alignment program performance. Since both PRRP and *A**-DCA reference 1 results were available, we also used this to check our implementations of the two algorithms.

4.2.1.1 Comparisons between our results and published results

For PRRP, we compared our results with the ones published at http://www-igbmc.u-strasbg.fr/BioInfo/BAliBASE/prog_scores.html. Overall, our results are lower than those of the published data, and with larger dispersion, but the trends are similar (Figure 8). We attribute the lower scores and the higher dispersion to our interpretation of Gotoh's description of the algorithm in [Gotoh, 1994, 1995, 1996]. In our implementation of PRRP, there were occasions where we need to derive details in order to implement the algorithm. One example is in calculating the weight of the entire tree. As a result, our implementation may not be as efficient, effective and robust (against different sequences) as his. For this reason, we were not surprised that our scores were generally lower and more dispersed than the published data.

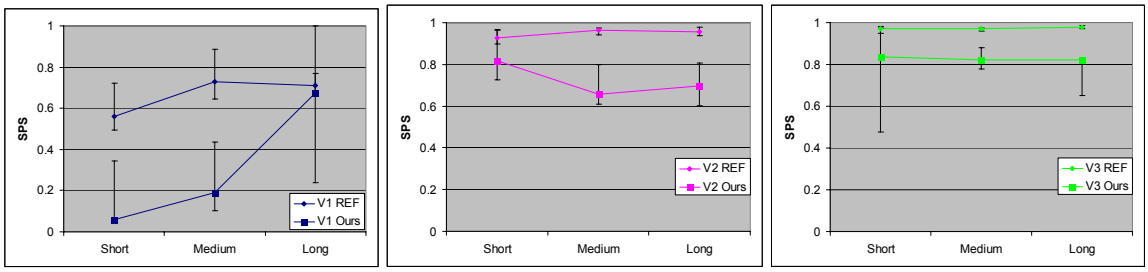

Figure 8. PRRP reference 1 results. Our alignment results versus published reference results. V1 = <25% ID, V2 = 20-40%ID, V3 = > 35% ID.

For *A**-DCA, we compared our results to those published in [Reinert, 2000]. The scores reported were alignment costs, and we took the set that best reflects our alignment conditions: limitation of memory (up to 2GB of memory) and alignment time (less than 12 hours). Since the definition of "alignment costs" is not clearly stated in the literature, we could only compare the general trend of the scores instead of their absolute values to our calculated SPS. Also, only a small subset of the reference 1 test sets was tested in the literature. As expected, our results follow the opposite trend as in the published data, since cost should be inversely related to alignment quality (Figure 9).

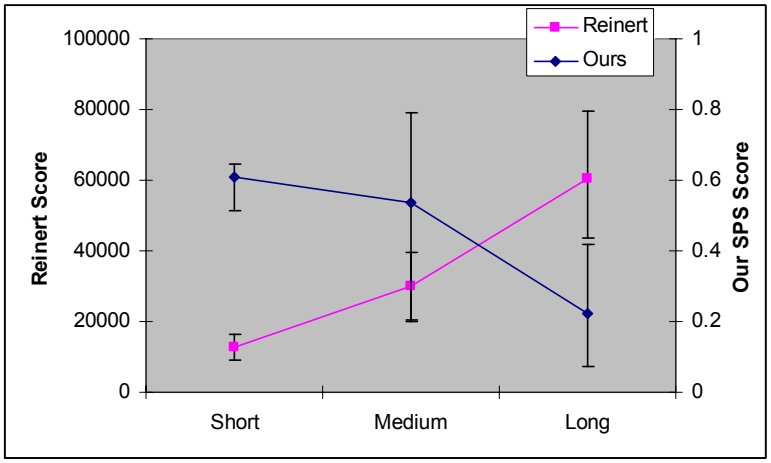

Figure 9. *A-DCA reference 1 results. Our alignment results versus published reference results from [Reinert, 2000].**

Based on these results, we believe our implementations of the algorithms are largely consistent with those of the original authors, even though our implementations are most likely to be less efficient, effective, and robust in general. Since there are more places of uncertainty in our PRRP implementation than *A**-DCA, we believe our obtained PRRP results are likely to be lower than our *A**-DCA results. Keeping this in mind, we then proceeded to compare our two programs.

4.2.1.2 Comparisons between PRRP and *A**-DCA

Overall, the two programs obtain similar SPS for the different sequence groups (short, medium and long), and homology groups (V1, <25%ID, V2, 20-40%ID and V3, >35%ID) as seen from (Figure 10), except for the "Short+V2" group.

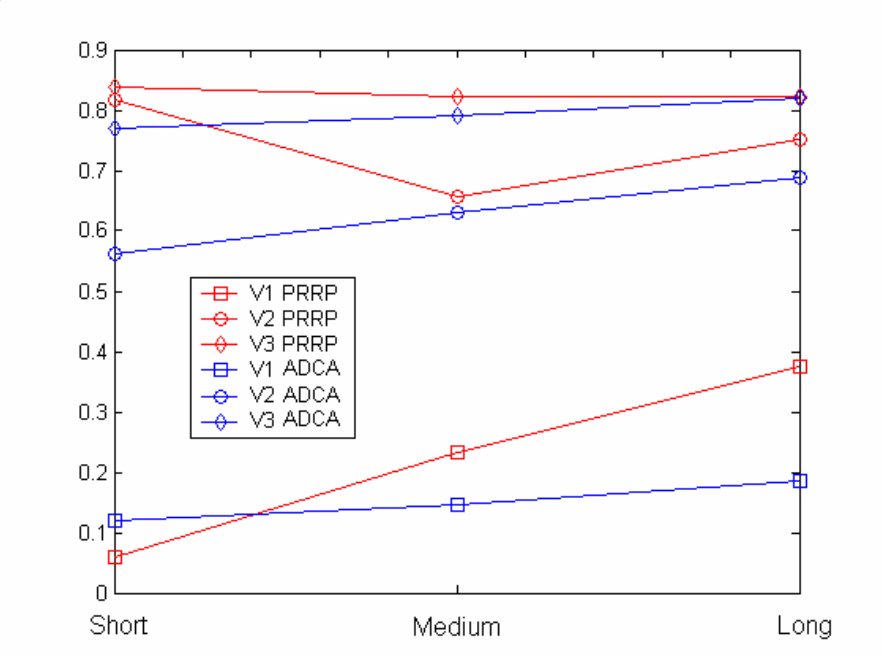

Figure 10. Reference 1 results for PRRP and *A-DCA, with three different sequence lengths (short, medium and long), and homology (V1, V2, V3).**

This observation was confirmed by the Friedman tests:

Testing Criteria	Chi-Square Value	*p*-value
Short	$\chi^2(1, N=54) = 3.0$	0.08
Medium	$\chi^2(1, N=54) = 2.5$	0.12
Long	$\chi^2(1, N=56) = 3.6$	0.06
V1	$\chi^2(1, N=46) = 0.7$	0.39
V2	**$\chi^2(1, N=62) = 14.2$**	**0.0002**
V3	$\chi^2(1, N=56) = 0.14$	0.71

Table 3. Friedman test results for reference 1, comparing the two program alignment scores.

Even though most of the Friedman tests results indicate no significant difference between the sets of results from the two programs, PRRP results were better numerically than those of *A**-DCA except for the "Short + V1" case.

The general trend of the results corroborates with those reported by [Thompson, 1999], where the high homology sets were better aligned than the lower homology set (i.e., V1), and the longer sequences are better aligned than the shorter ones.

4.2.2 Reference 2: a related family with divergent, orphan sequences

This reference set looks at how the programs handle orphan sequences amongst a related family when the family size is small (4 sequences), or large (14-22 sequences). For PRRP, we could align 13 out of the 23 large family sequences, while for *A**-DCA, we

could only align the small family sets.

The presence of orphans (up to three) disrupts the family alignments for small families for both programs (Figure 11). For PRRP, the decrease in alignment score with the introduction of orphans is 56%, and is found to be significant using the Wilcoxon signed rank tests ($p<0.001$). For *A**-DCA, the effect is less striking: a decrease of 15% in score (statistically significant at $p = 0.001$). When comparing between the two programs, *A**-DCA produces a family alignments of higher quality with the introduction of orphans ($\chi^2(1, N=46) = 5.3$, $p = 0.02$).

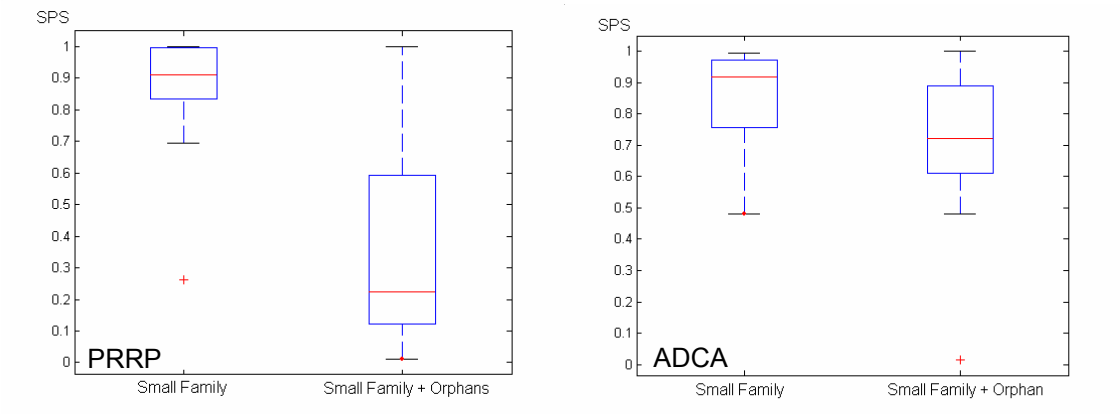

Figure 11. The effect of orphans on small family of sequences (reference 2) for the two programs. All the scores are those of the small families only, in the presence (right), or absence (left) of the orphans.

However, if we look at the alignments (family and orphans) as a whole, then both programs are similar in the alignment scores (Figure 12, $\chi^2(1, N=46) = 0.4$, $p = 0.5$).

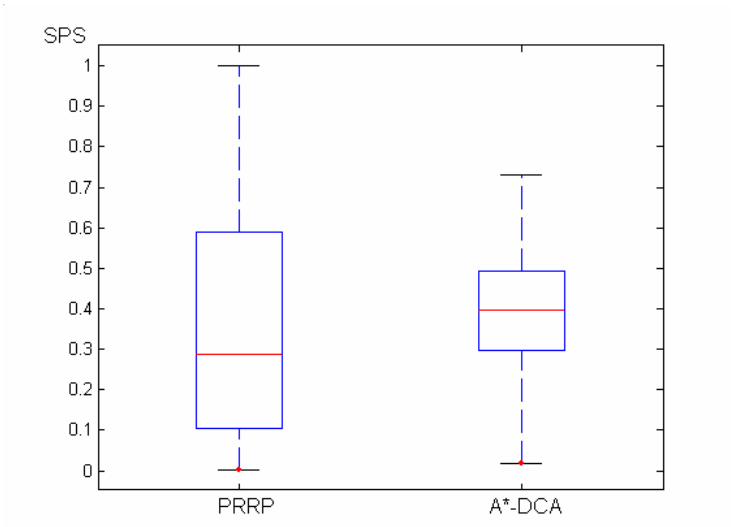

Figure 12. SPS for alignment of small family and orphans.

The corresponding results for large family are shown in (Figure 13). For PRRP, the

effect of orphans is much less pronounced in large family of sequences. The reduction in score is about 21%, but is not found to be significant ($p = 0.2$). Unfortunately, we could not align this set of references for *A**-DCA.

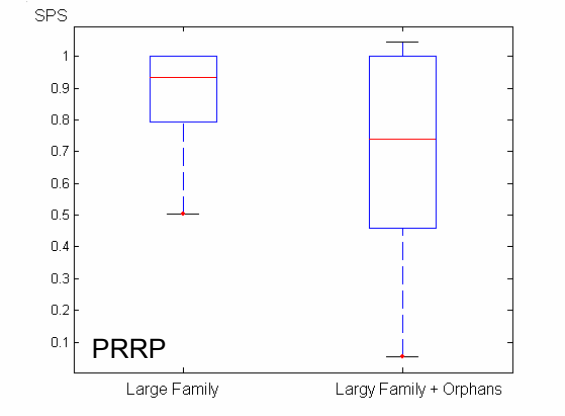

Figure 13. The effect of orphans on large family of sequences (reference 2) for PRRP.

4.2.3 Reference 3: families of related sequences

This reference tests the programs' ability to align families of sequences. We performed two types of tests here: aligning one family member of each family, and aligning the whole set of families. The CS is used in these tests, as SPS used previously are more influenced by the quality of the alignment within the family.

For PRRP, the effect of aligning with the whole set of family sequence and only one member from each family is striking: on average, the improvement of column score is 80% (Figure 14). This difference is found to be significant using the Wilcoxon signed rank tests with $p = 0.004$. This corroborates with the [Thompson, 1999] results, and is expected: With more family members, it should be easier to identify the conserved blocks in the sequences.

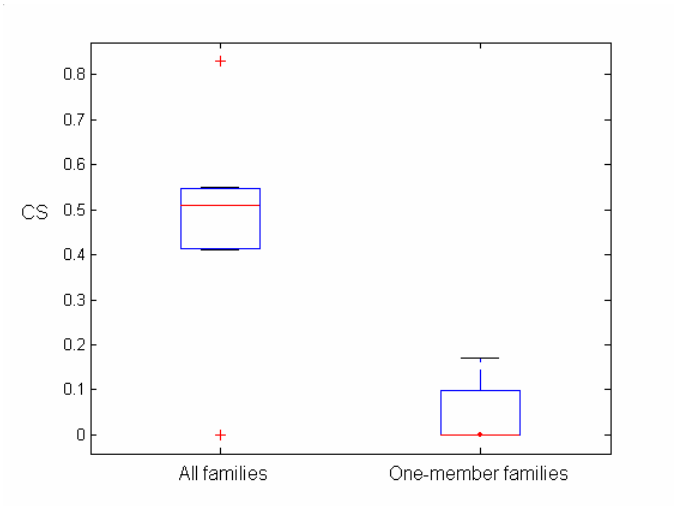

Figure 14. Boxplot for PRRP aligning families of sequences using all family members, and one member from each family. The values plotted are CS.

Unfortunately, we could not align the large family sequence sets using *A**-DCA due to memory limitations, and could not evaluate the algorithm in this reference set.

4.2.4 Reference 4: N/C-terminal extensions

This reference set tests insertion of sequences to the N/C terminal of the original sequences. In the [Thompson, 1999] paper, column scores were used to evaluate the program's ability to align all of the sequences correctly. However, in our implementations, the CS are too low to be useful (almost all zeroes for both PRRP and *A**-DCA). Instead, we used the SPS to look at the extent to which the program could align these sequences. Based on SPS, the quality of the alignments by PRRP and *A**-DCA are comparable, with *A**-DCA alignment being slightly better (Figure 15). This difference is not found to be significant ($\chi^2(1, N=20) = 1.6$, $p = 0.2$). This result is not surprising since PRRP has been reported to perform poorly with this test. In fact, it came second-last amongst all the programs tested in [Thompson, 1999].

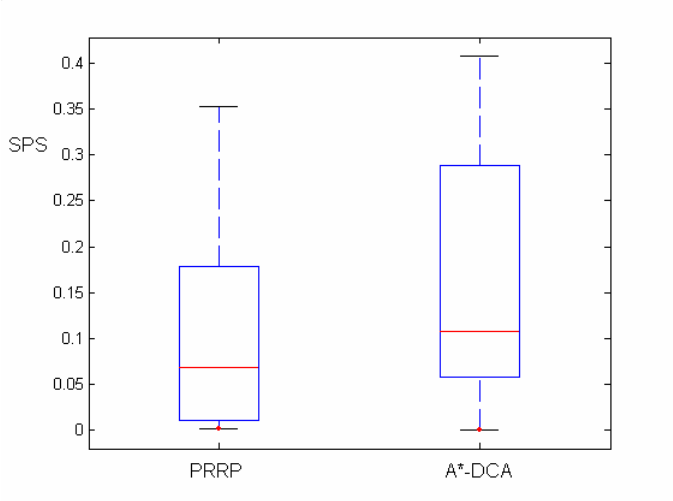

Figure 15. SPS for reference 4 (N/C-terminal extensions).

4.2.5 Reference 5: internal insertions

This reference set contains sequences of unequal length with insertions at internal sites of the homologous domains. The CS are shown in a boxplot (Figure 16). The Friedman test indicated that the two results are barely discernable ($\chi^2(1, N=20) = 3.57$, $p = 0.059$), with *A**-DCA producing slightly more optimal alignments.

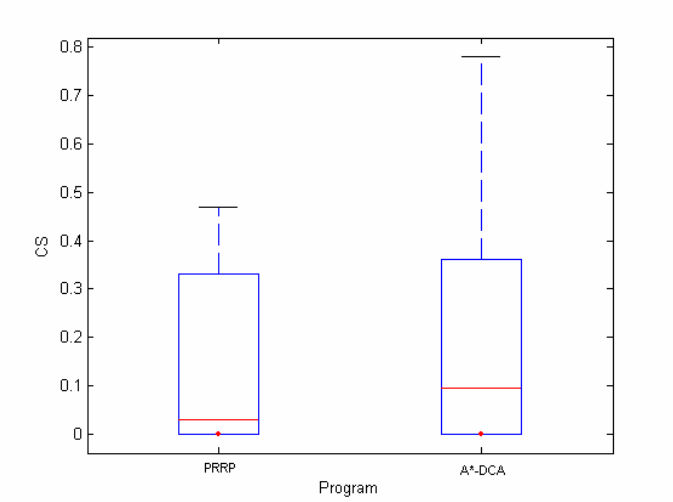

Figure 16. Boxplot for BAliBASE reference 5 (internal insertions) using CS.

4.3 Sequence Generator Tests

Since the rationale behind using weighted SPS in PRRP is to better model the uneven evolutionary distances between residues, it will be interesting to look at the effect of evolution on the performance of PRRP as compared to the second algorithm. In the BAliBASE database, references 1 contain sets of equidistance and homology, which are useful in this regard. Another approach in studying the effect of evolution on the

performance of our implementations is to simulate evolution. For this, we used a program called Sequence Generator (Seq-Gen) that simulates the evolution of nucleotide sequences along a phylogeny [Rambaut & Grassly, 1997], and the Hasegawa, Kishino and Yano (HKY) model of substitution [Hasegawa et al., 1985]. HKY assumes that evolution is independent and identical at each site and along each lineage. It allows for a different rate of transitions and transversions as well as unequal frequencies of the four nucleotides (i.e., base frequencies). This is a more general case than the Kimura model, where the base frequencies are equal, or the Jukes-Cantor model, where transition-to-transversion ratio is required to be 0.5 in addition to the equal base frequencies. We selected this model since it is a more realistic portray of evolution than the Kimura or the Jukes-Cantor models.

4.4 Sequence Generator Results

We selected the transition-transversion ratio of 3.0, unequal base frequencies (A=0.3, C=0.2, G=0.2, and T=0.3), and the following phylogenetic tree (Figure 17):

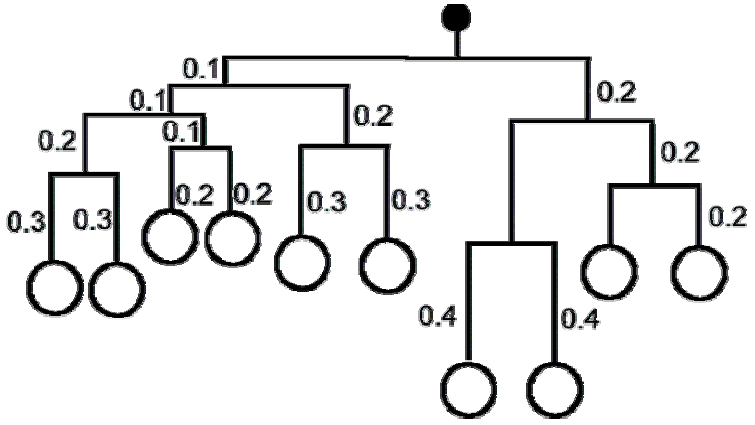

Figure 17. Phylogenetic tree used in Seq-Gen to simulate a set of related sequences. Open circles are individual sequences, and the closed circle is the root of the tree. Edge lengths are indicated by the numbers beside each edge.

These values and the tree are based on a documentation provided by the creators of Seq-Gen, which we assumed to be reasonably realistic. Using Seq-Gen, we created a set of 10 sequences of length 200 nucleotides. We then used our programs to align these generated sequences.

The results of the alignment are surprising: Both PRRP and *A**-DCA aligned all the sequences correctly, except for one case in PRRP, where the SPS was 0.95. Not surprisingly, the difference between these scores are found to insignificant ($\chi^2(1, N=20) = 1$, $p = 0.3$).

We then increase the degree of mutation in our parameters, with a transition-transversion ratio of 5. Despite the change, the results are still the same: PRRP and *A**-DCA aligned most of the sequences correctly, and the differences are found to be insignificant.

4.5 Discussion

The objective of our work was to study the performance of PRRP and *A**-DCA. More specifically, since the performance of PRRP has been compared to other alignment programs using the BAliBASE reference sets [Thompson, 1999], we could use the PRRP results to evaluate the performance of *A**-DCA.

We found that *A**-DCA's performance for sequences with various lengths and homology was slightly lower, but overall comparable to that of PRRP. This is very encouraging for *A**-DCA, since PRRP was ranked amongst the best alignment programs in this reference set (BAliBASE: reference 1). For both programs in general, alignment accuracy increased with homology and sequence length. The decrease in alignment accuracy with decreasing homology is expected since the identity of the sequence set becomes less distinct with low homology. Indeed, the greatest loss of performance occurred with V1 (<25% ID), which corresponds to the "twilight zone" of evolutionary relatedness. For sequence length, it was surprisingly to find that longer sequences are better aligned. [Thompson, 1999] observed the same trend in all the programs they tested, and attributed this unexpected behaviour to the nature of the BAliBASE reference set. Even thought the overall homology are the same for the difference sequence lengths, the core blocks in short sequences are less well-preserved that those in longer sequences, making it more difficult to align short sequences.

In areas where PRRP was not found to be optimal (extension and insertion, references 4 and 5), *A**-DCA alignment scores are slightly better than those of PRRP, but the differences are not found to be significant. This could be because both programs are global in nature. Since global alignments attempt to align the sequences over the whole length, it may have difficulty in locating locally conserved motifs. As a result, these algorithms may produce a collinear alignment of the entire lengths of the sequences, and result in less optimal scores for insertion/extension alignments. In contrast, local algorithms like dialign (iterative), MLpima (progressive), and SBpima (progressive) have been found to be much more effective in these reference sets.

Also, iterative algorithms may sometimes be unstable in the presence of biases in the sequence set, and diverge away from the correct alignment. As a result, iterative alignment algorithms tend not to perform well with reference 2, where the presence of orphans may cause instability in the algorithms. Initially, we believed *A**-DCA would not perform as well as PRRP in references 2, since PRRP takes the evolutionary distances between members into account when calculating the SPS, and obtained higher scores than most iterative algorithms like hmmt and dialing (Figure 1). However, we found that in reference 2, where orphans are introduced to small families of sequences, PRRP produces more disorganized family alignment than *A**-DCA, even though the overall alignment are comparable in quality. For large families, the family disruption is insignificant in PRRP. Unfortunately, we could not align any of the sequence sets with *A**-DCA to compare the two programs, which indicates to us that *A**-DCA would have more difficulty in aligning these reference sets than PRRP.

We obtained similar results with reference 3, where we aligned one member from each

family, and then the whole family. PRRP shows a striking improvement in the case where all the family members were included in the alignment set when compared to the one-member family case. Again, we could not align the sequences using *A**-DCA to compare the results and came to the same conclusion that *A**-DCA would have more difficulty with these reference sets.

Our simulated evolution results corroborate with our BAliBASE reference 1 results that the two programs are comparable in their ability to handle evolutionary changes. This was surprising to us since PRRP uses weighted SPS, which we believed would be helpful in these cases. However, both programs aligned most of the sequence sets perfectly in our testing, which may indicate the degree of mutation we have chosen may not have been sufficient to differentiate the two programs.

According to [Thompson, 1999], a big disadvantage of the current iterative methods is the heavy time penalty incurred. Based on our experience with these programs, we can attest and add to this statement: our programs also incurred a severe memory limitation. In fact, we could not align very long sequences (> 200 residues), or very large sets of sequences (> 20 sequences) given our memory limitations for *A**-DCA, and at times, PRRP as well. This led to our inability to test fully test reference 2 for both PRRP and *A**-DCA, reference 3 for *A**-DCA in the BAliBASE sets. In addition to our less efficient and effective implementations, we believe our choice of programming language may have led to our difficulties in aligning the large/long sequences. Java is known to be a slow language and perhaps with suboptimal memory management, and would not normally be suited for development of algorithms that are expensive in terms of processing time and memory. Nonetheless, Java allowed us to perform a preliminary evaluation of the two algorithms, and to focus on algorithmic details.

5 Conclusion

In our work, we selected and implemented two iterative algorithms using sum-of-pair scores for multiple sequence analysis based on a literature survey: PRRP and *A**-DCA. We tested our programs using two approaches. First, the BAliBASE reference sets, and second, simulated evolution with the Seq-Gen program. Before comparing between the two programs, we first compared our results with available data using the first reference set in BAliBASE. We found that the trends we observed with our results are comparable to those in literature, but our results are generally lower (from 10-20%), and more dispersed. We attribute these differences to our less efficient, effective and robust implementation of the algorithms. This is particularly true in the case of PRRP, where we needed to derive many details based on the author's descriptions of the algorithm. Also, we believe our choice of Java may have imposed a more severe memory limitation to our tests than a lower-level language (e.g., C).

Our results indicate that *A**-DCA performed similarly as PRRP. This is not surprising since both algorithms are iterative and global in nature. In terms of sequence sets of different lengths and homology (BAliBASE: reference 1), shorter and lower homoglogy sequences are generally more difficult. In terms of extension/insertion (BAliBASE: references 4/5), both programs had difficulty, as reported in literature due to their global

alignment nature. Originally, we hypothesized that PRRP would perform better with evolutionary changes (BAliBASE: references 2 and 3). Unfortunately, we could not align those reference sets with *A**-DCA to compare the two programs, which we see as an indicator that *A**-DCA had more difficulties with these reference sets. Our attempts to test the program based on simulated evolution failed to show a distinction between the two for the parameter of mutation we have chosen. Overall, our results are encouraging for *A**-DCA, since PRRP was known to be amongst the best programs in handling evolutionary changes (BAliBASE: reference 1, low homology set), and *A**-DCA had demonstrated a similar ability in the same reference set.

In short, *A**-DCA shows promising results in our preliminary evaluation despite the more demanding memory requirement. It would be interesting to further investigate the memory issue to see if it is due to our implementation, or is inherent in the algorithm itself. Since *A**-DCA is adaptive, we plan to construct its time and memory profiles to better characterize its run-time requirements. If we can overcome the memory requirements, we plan to evaluate *A**-DCA more thoroughly using the BAliBASE reference set and simulated evolution. Also, it will be interesting to compare *A**-DCA with other types of MSA (e.g., local, stochastic or progressive) to better characterize the strengths and weaknesses of *A**-DCA.

6 References

Altschul, S.F., Carroll, R.J., and Lipman, D.J. (1989) Weights for data related by a tree. *Journal of Molecular Biology* 20, 647-653.

Barton, G.J., and Sternberg, M.J.E. (1987) A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons. Journal of Molecular Biology 264(4), 823-838.

Berger, M.P., and Munson, P.J. (1991) A novel randomized iterative strategy for aligning multiple protein sequences. Computer Applications in the Biosciences 7, 479-484.

Durbin, Eddy, S.R., Krogh, A., Mitchison (1998). Biological Sequence Analysis. Cambridge University Press; Ch.6.

Freidman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of American Statistics Association 32, 675-701.

Gotoh, O. (1994). Further improvement in methods of group-to-group sequence alignment with generalized profile operations. Computer Applications in the Biosciences 9, 361-370.

Gotoh, O. (1995). A weighting system and algorithm for aligning many phylogenetically related sequences. Computer Applications in the Biosciences 11, 543-551.

Gotoh, O. (1996). Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments.

Journal of molecular biology 264, 823-838.

Hasegawa, M., Kishino, H. and Yano, T. (1985) Dating the human-ape splitting by a molecular clock of mitochondrial DNA. Journal of Molecular Evolution 22, 160-174.

Ishikawa, M., Toya, T., Hoshida, M. Nitta, K, Ogiwara, A., Kanehisa, M. (1993) Multiple sequence alignment by parallel simulated annealing. Computer Applications in the Biosciences 9, 267-273.

Kim, J., Pramanik, S., Chung, MJ. (1994) Multiple Sequence Alignment using Stimulated Annealing. Computer Applications in the Biosciences 10(4), 419-426.

Lawrence, C.E., Altschul, S.F., Boguski, M.S., Liu, J.S., Neuwald, A.F., Wootton, J.C. (1993) Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. Science 262, 208-214.

Needleman, S.B. and Wunsch, C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48, 443-453.

Notredame, C. (2002) Recent progresses in multiple sequence alignment; a survey. Pharamcogenomics 3(1), 142-2416.

Notredame, C., Higgins, D.G. (1996). SAGA: sequence alignment by genetic algorithm. Nuclear acid research 24, 1515-1524.

Notredame, C., Holm, L. Higgins, D.G. (1998). COFFEE: an objective function for multiple sequence alignment. Bioinformatics 14(5), 407-422.

Rambaut, A., and Grassly, G. C. (1997) Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. Computer Applications in the Biosciences 13 (3), 235-238.

Reinert, K., Stoye, J., Will, T. (2000) An iterative method for faster sum-of-pairs multiple sequence alignment. Bioinformatics 16(9), 808-814.

Reinert, K. (2004) Multiple Sequence Alignment. In Advanced Aspects in Sequence Analysis, SS 04.

Smith, T.F. and Waterman, M.S. (1981). Identification of common molecular subsequences. Journal of Molecular Biology 147, 195-197.

Sokal, R. R. and Michener, C.D. (1985) A statistical method for evaluating systematic relationships. University of Kansas Scientific Bulletin 28: 1409-1438.

Stoye, Jens Multiple sequence alignment with divide-and-conquer method. *Gene*, 211,

GC45-GC56, 1998

Tönges, U., Perrey, S.W., Stoye, J., Dress, A.W.M. (1996) A General Method for Fast Multiple Sequence Alignment; Preprint, Universität Bielefeld.

Thompson, J.D., Plewniak, F., and Poch, O. (1999) A comprehensive comparison of multiple sequence alignment programs. Nucleic Acid Research 27(13), 2682-2690.

Wang, L. and Jiang, T. (1994) On the complexity of multiple sequence alignment. Journal of Computational Biology 1(4), 337-348.

Wilcoxon, F. (1947) Probability tables for individual comparisons by ranking methods. Biometrics 3, 119-122.

Appendix

In PRRP, the total weight calculation is described by [Gotoh, 1995]. In order to implement the pseudo-code, we need to identify the two vector profiles P and Q. We derived the values of P and Q based on the Gotoh 1995 paper, and one of his previous publications [Gotoh, 1994]:

$$WSP(L,R) = \sum_{j \in L} \sum_{k \in R} w_{j,k} s_{j,k}$$

$$= \sum_{i=1}^{I} \sum_{j \in L} \sum_{k \in R} w_{j,k} d(j_{x,i}, k_{y,i})$$

where ($d(x,y)$ is the dissimilarity of residues x and y)

$$= w_e \sum_{i=1}^{I} \sum_{j \in L} \sum_{k \in R} w_{j,e_R} w_{j,e_L} d(j_{x,i}, k_{y,i})$$

where w_e is the weight of the edge $e = (e_L, e_R)$

$$= w_e \sum_{i=1}^{I} \sum_{j \in L} \sum_{k \in R} w_{j,e_R} w_{j,e_L} \sum_{x \in X} p_{x,i}^{L} \cdot f_{x,i}^{R}$$

given $\sum_{j \in L} \sum_{k \in R} d(j_{x,i}, k_{y,i}) = \sum_{x \in X} p_{x,i}^{L} \cdot f_{x,i}^{R}$ [Gotoh, 1994]

$$= w_e \sum_{i=1}^{I} W^L \cdot p_i^L \cdot f_i^R \cdot W^R$$

$$= w_e \sum_{i=1}^{I} P_i^L \cdot Q_i^R$$