An Evolutionary Algorithm for the Maximum Weight Trace Formulation of the Multiple Sequence Alignment Problem

Gabriele Koller and Günther R. Raidl

Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Vienna, Austria
{koller|raidl}@ads.tuwien.ac.at

Abstract. The multiple sequence alignment problem (MSA) can be reformulated as the problem of finding a maximum weight trace in an alignment graph, which is derived from all pairwise alignments. We improve the alignment graph by adding more global information. A new construction heuristic and an evolutionary algorithm with specialized operators are proposed and compared to three other algorithms for the MSA, indicating advantages of the new approaches.

1 Introduction

Multiple sequence alignments are of great practical interest in the area of molecular biology. For example, they are essential for the prediction of secondary and tertiary structures of protein sequences and for finding conserved motifs in a group of related proteins [2]. As a DNA or protein sequence can be represented by a string over a finite alphabet Σ, the multiple sequence alignment problem (MSA) can be defined as follows.

Given a set of $k > 2$ strings $S = \{S_1, \ldots, S_k\}$, where each string S_a, $a = 1, \ldots, k$, consists of characters $s_{a,1}, \ldots, s_{a,l_a} \in \Sigma$, find a multiple sequence alignment $\hat{S} = \{\hat{S}_1, \ldots, \hat{S}_k\}$ over the alphabet $\hat{\Sigma} = \Sigma \cup \{\text{"-"}\}$ which minimizes a given scoring function and has the following properties: (1) all strings in $\hat{S}$ have the same length l with $\max_{a=1\ldots k}(l_a) \leq l \leq \sum_{a=1}^{k} l_a$, (2) ignoring dashes, $\hat{S}_a$ is identical with $S_a, \forall a = 1, \ldots, k$, (3) no column of $\hat{S}$ only contains dashes. In the alignment, a dash represents a space between characters of the original sequences. Fig. 1a gives an example for the MSA.

To score an alignment, the weighted sum-of-pairs (SP) function with affine gap penalties is most widely used [2]. Optimal pairwise alignments ($k = 2$) can be found in time $O(l^2)$ by dynamic programming [6]. For the general case $k > 2$ the time complexity of exact methods with SP-score increases exponentially with k. Thus, exact methods can in practice only be applied to instances with few, rather short sequences. For larger instances, various heuristics have been developed.

Heuristic methods for the MSA fall into different categories, see [7] for an extensive survey. The majority of them follow the progressive approach, in which

This work is supported by the Austrian Science Fund (FWF) under grant P16263-N04.

an alignment is built by repeatedly aligning two sequences or multiple alignments of subsets of the sequences by a pairwise alignment method. A widely used algorithm of this category is ClustalW [12]. While it is usually fast and yields acceptable solutions, it may produce sub-optimal alignments due to the preservation of gaps of the pairwise alignments. T-Coffee [9] is also based on the progressive method, but tries to avoid such drawbacks by using a tree-based consistency objective function called COFFEE [10]. A library containing all pairwise alignments replaces the usual substitution matrix for the evaluation of aligned characters. To incorporate more global information, the library is extended by transitive relationships for all character triples from different sequences.

Iterative algorithms try to overcome the drawbacks of progressive methods by refining an initial alignment, generated randomly or by some other method, via local improvement and/or random variation. Evolutionary algorithms such as SAGA [8] fall into this category. SAGA was one of the first attempts to apply a evolutionary algorithm (EA) to the MSA with SP-score. It uses a 2D array to represent the alignment and includes a variety of crossover and mutation operators. The EA of Zhang et al. [13] focuses on identifying fully matched columns. The evolutionary programming approach by [1] uses a different objective function; it maximizes a score for matched characters minus a gap penalty.

In our work we focus on the maximum weight trace formulation, which maps the MSA to a graph problem [3]. This formulation is described in Sect. 2. Two novel strategies for improving the information in the underlying alignment graph are presented in Sect. 3. In Sect. 4, a construction heuristic for deriving a trace from an alignment graph is described. Sect. 5 presents an EA with specific operators based on the construction heuristic. The algorithms are experimentally compared to ClustalW, T-Coffee, and SAGA in Sect. 6.

2 The Maximum Weight Trace Formulation of the MSA

The problem of aligning multiple sequences can be transformed into the maximum weight trace problem (MWT), which has been introduced by Kececioglu [3] as a natural formulation of merging partial alignments to form multiple alignments. Like MSA, MWT is NP-hard. It is based on the concept of the *alignment graph* $G = (V, E)$, whose nodes V correspond to the characters of the k input sequences and are labeled correspondingly, and whose edges E represent desirable alignments of character pairs. Each edge $e \in E$ is assigned a weight $w(e) > 0$ corresponding to the desirability of the pairwise character alignment. For convenience, G is extended to a mixed graph $\hat{G} = (V, E, H)$ by adding directed arcs H connecting all successive nodes of each sequence, see Fig. 1b. A *trace* $T \subseteq E$ is a subset of edges that can be realized in a multiple alignment, i.e., each pair of characters connected by an edge appears in the multiple alignment in the same column. A set of edges forms a trace iff the arc-extended subgraph $\hat{G}_T = (V, T, H)$ does not contain any cycle including an arc, see Fig. 2. Given a weighted alignment graph G, the MWT is to find a trace T with maximum total weight $w(T) = \sum_{e \in T} w(e)$.

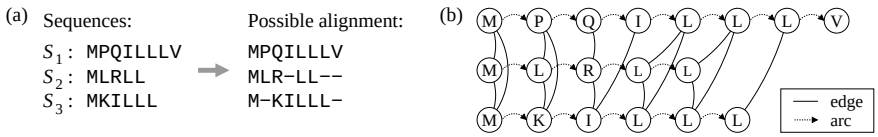

Fig. 1. Examples for (a) a multiple sequence alignment and (b) an arc-extended alignment graph for three sequences (edge weights are omitted).

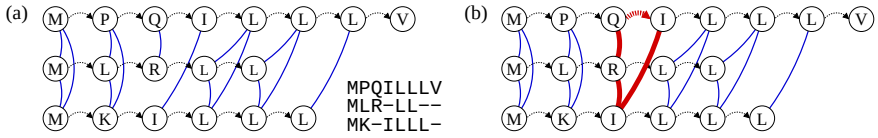

Fig. 2. Examples for traces for the alignment graph in Fig. 1b: (a) a trace and a corresponding alignment (note that edge (R,I) is not in the trace), (b) an edge set violating the trace property since it contains a cycle including an arc: characters Q and I of row 1 cannot simultaneously be aligned with character I of row 3.

A shortest possible multiple alignment can be derived from a given trace column by column by aligning characters connected by an edge in the trace. For each column, the algorithm considers in each sequence the left-most character not yet included in the alignment—the *front*—and adds it to the current column if it is not connected by an edge to a character to the right of the front.

A branch-and-bound algorithm [3] and a polyhedral approach [4] have been proposed to solve small to moderately sized instances of the MWT exactly. In these approaches, the alignment graph is constructed by calculating all pairwise alignments and including corresponding edges. Edge weights are simply assigned according to an appropriate substitution matrix.

3 Improvements on the Alignment Graph

Obviously, the edges of the alignment graph and their weights are crucial for the success of any method applied to the MWT formulation of the MSA. So far, finding appropriate edges and weights has received only little attention. We propose two concepts to better reflect some of the global information contained in all pairwise alignments: a neighborhood-dependant weighting scheme for calculating edge weights and an edge extension.

We first obtain a preliminary edge set E from all pairwise sequence alignments which are computed using dynamic programming. For each pair of aligned characters $s_{a,i}$ and $s_{b,j}$, we add an edge $e = (s_{a,i}, s_{b,j})$. The purpose of the new weighting scheme is to take the neighborhood of an edge into consideration and to reward edges in regions of higher similarity with higher weights. The edge weight $w(e)$ is calculated as a weighted sum of the value from ClustalW's substitution matrix for the pairwise alignment of $s_{a,i}$ and $s_{b,j}$ and the values for up

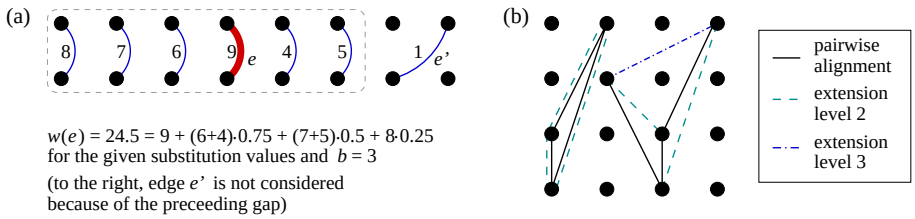

Fig. 3. Examples for (a) the computation of an edge weight and (b) the alignment graph extension by adding transitive edges up to level $d_{\max=3}$.

to b consecutively aligned character pairs to the right and to the left of e. The value of a neighbor with distance Δ to e in terms of columns is scaled by the factor $1 - \frac{\Delta}{b+1}$. The block of considered character pairs is limited by the first gap or by b. An example is given in Fig. 3a.

Furthermore, we extend the edge set E by transitive edges up to a level $d_{\max}$. This extension can be viewed as a generalization of the concept of the extended library [9] where only transitive edges of level 2 are considered. An edge $e_t = (s_{a,i}, s_{b,j})$ is called a *transitive edge* of level $d \geq 2$ if there exists a path of d edges between $s_{a,i}$ and $s_{b,j}$, where only one character per sequence may be involved, see Fig. 3b. A transitive edge's weight depends on the minimum edge weight $w_{\min}$ on the path and the level: $w(e_t) = w_{\min}/(d-1)$. If e_t already exists, then its weight is increased by $w(e_t)$. The addition or update of all transitive edges of level $d = 2, \ldots, d_{\max}$ for k sequences of maximum length l can be performed in time $O(l \cdot k^{d_{\max}+1})$ using a recursive depth-first search algorithm.

4 The Repeated Edge-Insertion Construction Heuristic

This section describes a new heuristic for deriving a feasible trace from a given weighted alignment graph, which we call *repeated edge-insertion construction heuristic* (REICH). The basic idea follows Kruskal's greedy algorithm for finding a minimum spanning tree of an undirected graph. Initially, trace T is the empty set, and all edges are sorted according to decreasing weight; ties are broken randomly. In its main part, the algorithm considers each edge in the predetermined order and adds it to the trace if this does not violate the feasibility of the trace. Otherwise, the edge is discarded.

As discussed in Sect. 2, a set of edges represents a trace as long as the arc-extended subgraph does not contain any cycle including an arc. When we consider an edge $(s_{a,i}, s_{b,j}) \in E$ for inclusion in the trace, we therefore check if a path from $s_{a,i}$ to $s_{b,j}$ or a path from $s_{b,j}$ to $s_{a,i}$ already exists in the arc-extended trace. We can safely add edge $(s_{a,i}, s_{b,j})$ to T if no such path exists or the found path does not contain any arcs.

Since the alignment graph usually contains many edges, the running time of the path-search procedure is critical. We make the following observations and later exploit them in an efficient algorithm. For convenience, we say *a path*

enters/leaves a sequence if it contains an edge leading to a node of this sequence or an edge leading away from a node of this sequence, respectively.

1. If we find a path from a node $s_{a,i}$ to a node $s_{b,j}$, we have implicitly found all the paths from any node $s_{a,i'}$ with $i' \leq i$ to any node $s_{b,j'}$ with $j' \geq j$.
2. We only have to consider paths which enter/leave any sequence at most once. Furthermore, the path must not enter the sequence containing the source node or leave the sequence containing the destination node. All paths visiting a sequence more than once have corresponding simpler paths that use arcs of this sequence as shortcuts.
3. It follows that we only have to consider paths containing at most $k-1$ edges (plus the arcs, whose number is not limited by k).
4. Let $s_{a,i}$ be the source node. If we have identified the edge $(s_{a,m}, s_{c,n})$ with minimum $m \geq i$ from all the edges connecting nodes of sequences a and c, we do not have to consider any further edge inbetween these two sequences. For any subsequent edge $(s_{a,m'}, s_{c,n'})$ with $m' > m$, $n' > n$ must hold (otherwise, the trace would be infeasible), and, due to observation (1), only a subset of the nodes already reachable via edge $(s_{a,m}, s_{c,n})$ is reachable via the edge $(s_{a,m'}, s_{c,n'})$. More generally, there even cannot exist any path from $s_{a,m'}$, $m' > m$, to any node $s_{c,n''}$ with $n'' \leq n$. However, a path from $s_{a,m''}$ with $m'' < m$ to a node $s_{c,n''}$, $n'' < n$, may exist.

A simple depth-first search in the arc-extended trace for identifying a path from $s_{a,i}$ to $s_{b,j}$ would clearly be inefficient. Instead, we perform a breadth-first search starting from $s_{a,i}$ and consider the above observations for pruning the search tree. Global variables $M_1, \ldots, M_k$ are used to store for each sequence the minimum indices of the nodes which are reachable via the already considered edges/paths; initially, $M_a = i$ and $M_c = \infty$, $\forall c \neq a$. At any iteration of the breadth-first search, at most $k-1$ different *active paths* exist, which have last improved the values M_c, $\forall c \neq a$. The path we consider next for an extension with a new edge is always the one with the minimum number of edges; if two or more paths with the same minimum number of edges exist, we choose the path involving the smallest number of arcs.

To optimally support this breadth-first search, the trace is implemented as a special data structure. For each pair of sequences $c = 1, \ldots, k$ and $d = 1, \ldots, k$, $c \neq d$, we maintain a balanced tree $A_{c,d}$ which stores references to all edges connecting nodes of sequences c and d, sorted by the index of the node in sequence c. In this way, we can efficiently identify the "next" edge leading from a certain position in sequence c to sequence d, i.e., the edge $(s_{c,m'}, s_{d,n})$ with minimum $m' \geq m$ for given m, c, and d. The balanced trees allow this operation in time $O(\log \hat{l})$, with $\hat{l} = \max\{l_1, \ldots, l_k\}$, and need $O(k^2 + |T|)$ space.

When using this data structure, the total running time of REICH is bounded above by $O(|E| \log |E| + |E| k^3 \log \hat{l})$, where the first term represents the initial

sorting of edges and the factor $O(k^3 \log \hat{l})$ of the second term comes from the breadth-first search[1].

The running time of the described algorithm can be further improved to a worst case complexity of $O(|E| \log |E| + |T|k^2 \log \hat{l} + |E| \log \hat{l})$. For this purpose, we additionally store two kinds of auxiliary edges in the trace data structure: (a) An *alias-edge* $(s_{c,m}, s_{d,n})$ is stored if the two incident nodes are already connected by a path containing no arcs. (b) A *directed path-edge* $(s_{c,m}, s_{d,n})$ is stored if a path including at least one arc exists from node $s_{c,m}$ to node $s_{d,n}$, and we are so far not aware of any node $s_{d,n''}$ with $n'' < n$ that is reachable from $s_{c,m}$. Keeping the trace data structure up-to-date with these auxiliary edges needs additional bookkeeping, but it turns out to be worthwhile, since some effective additional conditions for further pruning the breadth-first search can now be efficiently checked.

5 The Maximum Trace Evolutionary Algorithm (MTEA)

The evolutionary algorithm described in this section is the first one that makes use of the MWT formulation of the MSA problem. In combines and enhances ideas of REICH and previous genetic algorithms for MSA.

5.1 Representation and Initialization

Candidate alignments are primarily represented in the most direct form as two-dimensional array $\hat{S}$. When needed, the trace data structure is additionally created and stored.

To create promising initial candidate solutions and to ensure diversity at the same time, a randomized version of REICH is used: A random number $|D| \in \{\lfloor k \cdot \hat{l}/2 \rfloor, \ldots, \lfloor k \cdot (k-1) \cdot \hat{l}/2 \rfloor\}$ is picked; a list $D \subseteq E$ of the $|D|$ edges with highest weights is determined and randomly permuted. Then, REICH is performed on this list to obtain a trace, which is finally decoded into a multiple alignment. To achieve higher diversity, we keep track of the edges from D that could not be included in the trace. When creating the next solution, these edges are forced to be included in D before all other edges. Additionally, REICH is applied once to the whole list of edges.

5.2 Variation Operators

We apply the following five variation operators. They were carefully chosen or designed with their specific strengths and computational efforts in mind in order to achieve highest possible synergy. The first three operators are relatively simple

[1] The depth of the breath-first search is bounded by $k-1$. In depth t, $0 < t < k$, a maximum of $k-1-t$ possible extensions must be further considered for each of the $\leq k-1$ active paths. Considering the time $O(\log \hat{l})$ for finding an edge for a possible extension, we get the total time complexity $O(k^3 \log \hat{l})$ for one call of the search.

and fast; they are traditionally working directly on the multiple alignments. In contrast, the latter two operators focus on the MWT formulation and make active use of the trace data structure and the alignment graph; they are more powerful, however, computationally also more expensive.

One-point crossover has been used in several previous EAs for MSA including [8, 13]. The first parent alignment is cut straight after a randomly chosen column. The whole left part is copied to the offspring. The right part is adopted from the second parent alignment which is tailored accordingly so that it can be appended while keeping the characters' order of the original sequences. Any void space appearing at the junction point is filled with gaps.

Block shuffling mutation has also previously been used [8]. From one sequence a block of consecutive characters being bounded left or right by one or more gaps is randomly chosen. This block is then shifted over the neighboring gaps. When there is more than one neighboring gap, we either check them all and keep the best position or determine the target position randomly; the actually applied strategy is decided at random. To limit the changes and the computational effort, we apply this operator only to blocks of a maximum length of 9 and consider at most 9 neighboring new positions.

Best consistent cut crossover. Columns of two alignments are called *consistent*, when for each sequence their characters (or gaps) are identical with respect to the corresponding positions in the original sequence. Blocks inbetween consistent columns can be directly exchanged among parent alignments [8]. For the alignment of many protein sequences, however, such consistent columns are rare, and the operator can therefore not often be applied. We relax the requirements by introducing *consistent cuts*: A cut of the first parent alignment after some column is said to be consistent if no right-most symbol of the left part is aligned with a symbol of the right part in the second parent alignment. In this case, the second parent can also be cut straight at the corresponding position and the right part of the second parent can simply be concatenated to the left part of the first parent. In our crossover, we determine all possible consistent cuts and choose the one yielding the highest total weight. Since the weights can be calculated incrementally, only one sweep over the parent alignments is necessary, and the operator can be implemented in time $O(k^2\hat{l})$.

Edge insertion mutation selects an edge from the alignment graph and includes it in the alignment. The selection is performed by determining a uniform random value $u \in [0, 1)$, calculating an exponentially distributed rank $r = \lfloor \rho \log(1-u) \rfloor \mod |E| + 1$ from it, and using this rank as an index into an array containing references to all edges E sorted according to decreasing weight. The strategy parameter ρ is set according to theoretical and experimental investigations described in [5]. If an edge is selected that already exists in the trace of the current alignment, the edge-selection is repeated.

In order to feasibly include the selected edge e, it is usually necessary to remove several other edges from the alignment's trace. All edges of the original

alignment's trace realized in the alignment in columns to the left or the right of both end-nodes of edge e will remain unchanged. All other edges of the trace, thus, the edges representing alignments of symbols in the alignment-block $\hat{B}$ delimited to the left and right by the columns containing the characters to be aligned when including e, are temporarily removed and stored in a list R. The new edge e can now safely be added to the trace. Then, REICH is used to process R in order of decreasing weight and re-insert as many of the edges as possible. Finally, as local improvement, we determine all edges from E being incident to a character in block $\hat{B}$ and not contained in R. These edges are also considered in decreasing weight order for insertion in the trace by applying REICH.

Path relinking uses two parent alignments $\hat{A}$ and $\hat{B}$ and transforms $\hat{A}$ step by step into $\hat{B}$. Thus, we consider all solutions lying in the search space on a path connecting the two parents. The best intermediate solution is kept as offspring.

We transform $\hat{A}$ into $\hat{B}$ column by column from left to right. Let $\hat{I}_i$ be the ith intermediate solution, with $\hat{I}_0 = \hat{A}$ and $\hat{I}_z = \hat{B}$. The first i columns of alignment $\hat{I}_i$ always correspond to those in parent $\hat{B}$. We determine $\hat{I}_{i+1}$ from $\hat{I}_i$ for $i = 0, \ldots, z-1$ as follows. Let X be the characters appearing in column $i+1$ of $\hat{B}$. This column will be realized in $\hat{I}_{i+1}$ in a locally optimal way.

Let Y denote the set of all characters appearing in $\hat{B}$ to the right of X, and let $Y^C \subseteq Y$ be the subset of characters connected with a character of X in the trace of I_i. In order to align the characters of X, the edges connecting X and Y^C cannot be retained, and they are therefore deleted from the trace. All edges from E connecting two nodes from X are then realized and added to the trace; the characters in X are actually aligned. Finally, we check if the performed removal of edges allows for an inclusion of new edges from E at the right side of X: All edges from $(a, b) \in E$ with $a \in Y^C$ and $b \in Y$ are considered, and if they do not yet appear in the trace they are tried to be included by using the edge-insertion algorithm of REICH. The corresponding alignment I_{i+1} is updated accordingly. In this way, all intermediate solutions determined by path relinking are locally optimal in the sense that no further edges from E can be realized in the trace without removing others.

5.3 General Structure of the Algorithm

We use an island model as basic structure for the EA. Each island follows the steady-state concept by always only creating a single new solution candidate per iteration. If this offspring is a duplicate of another solution of the island, it is discarded. Otherwise, it replaces the worst solution. Within each island, an offspring is always created by performing either one-point crossover or best consistent cut crossover (each with 50% probability) followed by the block shuffling mutation. The more complex and time-demanding path relinking and edge insertion mutation are applied with relatively low probabilities as inter-island operators. Parents are randomly chosen among the best solutions of all islands, and the offspring is incorporated into another randomly chosen island. Additionally, traditional migration is performed: With a certain probability, the best solution of a randomly chosen island is copied into another island.

Table 1. Average results of ClustalW, T-Coffee, SAGA, REICH, and MTEA on the BAliBASE instances: annotated SPS-values and total edge weights $w(T)$.

Instance	ClustalW	T-Coffee	SAGA	REICH		MTEA	
Class	SPS	SPS	SPS	$w(T)$	SPS	$w(T)$	SPS
Ref 1	0.859	0.865	0.789	1107	0.877	1114	**0.888**
Ref 2	0.847	0.854	0.800	19027	**0.861**	19047	**0.861**
Ref 3	0.749	0.773	0.672	18282	0.796	18347	**0.804**
Ref 4	0.770	0.835	0.564	5086	0.862	5101	**0.863**
Ref 5	0.850	0.956	0.752	5140	0.960	5149	**0.965**
Total avg.	0.837	0.860	0.753	6144	0.873	6158	**0.880**

6 Experimental Results

We experimentally compare REICH and MTEA to ClustalW 1.82, T-Coffee, and SAGA using the 145 instances of the BAliBASE 1.0 benchmark library [11] for protein sequence alignment. These instances are grouped into five classes: equidistant sequences of similar length (Ref 1), a related family with divergent, orphan sequences (Ref 2), families of related distances (Ref 3), sequences with N/C-terminal extensions (Ref 4), and sequences with internal insertions (Ref 5).

For REICH and MTEA, the alignment graphs were built using ClustalW to calculate global pairwise alignments and additionally—as suggested in [9]—LALIGN to consider local pairwise alignments. The neighborhood-dependent calculation of edge-weights was applied with $b = 10$, and edge extension was performed to level $d_{\max} = 2$. Appropriate parameters for MTEA were obtained in preliminary tests: The population size was 100, divided into four equally sized islands. Migration was performed with probability 0.1%, path relinking with 2%, and insert edge mutation with 10% per iteration. A run was terminated when no new best solution had been found for 1 000 iterations or after a total of 10 000 iterations. ClustalW, T-Coffee, and SAGA were performed using default parameter settings. All experiments were done on a Pentium 4/1.9 GHz PC.

The resulting alignments were evaluated using the annotated SPS-values of [11], which indicate the proportion of correctly aligned character pairs of an alignment with respect to the manually refined BAliBASE reference alignment considering only trusted blocks. A value of 1 means that the computed alignment is identical to the reference alignment, whereas 0 signifies that no character pair has been correctly aligned. Table 1 shows average results for each algorithm on each of the five classes of instances. MTEA yielded for all five instance classes the best average results; however, absolute differences of the SPS-values to the results of REICH are generally small. A Wilcoxon signed rank test nevertheless indicates the significance of the difference with an error probability less than 0.1%. On instance class 2, MTEA could not improve on the average SPS-value, although slightly better solutions were obtained with respect to the average edge weights $w(T)$. REICH and MTEA performed on nearly all test instances as good as or better than ClustalW, T-Coffee, and SAGA. In detail, the best known solutions were obtained by MTEA for 57% of all instances compared to 40% by REICH, 37% by T-Coffee, 24% by ClustalW and 15% by SAGA.

Running times strongly depend on the individual instances. In general, ClustalW was fastest, but also T-Coffee and REICH took only about one minute on the largest instances. SAGA needed up to 214 min and MTEA up to 413 min.

7 Conclusion

The maximum weight trace formulation of the MSA is a well-suited model for finding accurate multiple alignments. The proposed calculation of edge weights and the extension of the alignment graph by transitive edges enhance the model by the inclusion of more global information. The construction heuristic REICH is fast and yields high quality alignments which are usually superior to those obtained by ClustalW, T-Coffee, and SAGA. If longer running times are acceptable, the proposed EA often finds even significantly better solutions.

References

1. K. Chellapilla and G. B. Fogel. Multiple sequence alignment using evolutionary programming. In P. J. Angeline et al., editors, *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pages 445–452. IEEE Press, 1999.
2. D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.
3. J. D. Kececioglu. The maximum weight trace problem in multiple sequence alignment. In *Proceedings of the 4th Symposium on Combinatorial Pattern Matching*, number 684 in LNCS, pages 106–119. Springer, 1993.
4. J. D. Kececioglu, H.-P. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, and M. Vingron. A polyhedral approach to sequence alignment problems. *Discrete Applied Mathematics*, 104:143–186, 2000.
5. S. Leopold. An alignment graph based evolutionary algorithm for the multiple sequence alignment problem. Master's thesis, Vienna University of Technology, Vienna, Austria, February 2004.
6. S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.
7. C. Notredame. Recent progresses in multiple sequence alignment: A survey. *Pharmacogenomics*, 3(1):131–144, 2002.
8. C. Notredame and D. G. Higgins. SAGA: Sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24(8):1515–1524, 1996.
9. C. Notredame, D. G. Higgins, and J. Heringa. T-COFFEE: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, 392:205–217, 2000.
10. C. Notredame, L. Holm, and D. G. Higgins. COFFEE: An objective function for multiple sequence alignment. *Bioinformatics*, 14(5):407–422, 1998.
11. J. Thompson, F. Plewniak, and O. Poch. BAliBASE: A benchmark alignments database for the evaluation of multiple sequence alignment programs. *Bioinformatics*, 15:87–88, 1999.
12. J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.
13. C. Zhang and A. K. C. Wong. A genetic algorithm for multiple molecular sequence alignment. *CABIOS*, 13(6):565–581, 1997.