

Monte-Carlo Tree Search for the Multiple Sequence Alignment Problem

Stefan Edelkamp and Zhihao Tang

Bremen University
{edelkamp,zhihao}@tzi.de

Abstract

The paper considers solving the multiple sequence alignment, a combinatorial challenge in computational biology, where several DNA RNA, or protein sequences are to be arranged for high similarity. The proposal applies randomized Monte-Carlo tree search with nested rollouts and is able to improve the solution quality over time. Instead of learning the position of the letters, the approach learns a policy for the position of the gaps. The Monte-Carlo beam search algorithm we have implemented has a low memory overhead and can be invoked with constructed or known initial solutions. Experiments in the BALiBASE benchmark show promising results in improving state-of-the-art alignments.

Introduction

Multiple sequence alignments (MSA) are frequently used for the analysis of DNA, RNA, or protein sequences in order to determine the evolutionary relation between species with a common ancestor, to predict the so-called secondary/tertiary structure, as well as the functional centers, in which as few possible mutations as possible occur (assuming that similar sequences inherit similar structures and function).

Computational biologists have declared the MSA problem to be a *holy grail* (Gusfield 1997). One reason is that solving this problem often leads to a high memory demands, which has been partially leveraged with frontier search (Hirschberg 1975; Korf et al. 2005), refined heuristics, and variants of memory-limited (Zhou and Hansen 2002; 2003) or iterative-deepening heuristic search (Schrödl 2005). Most of these approaches provide strategies to limit exploring the search space induced by dynamic programming (Bellman 1957). Tools like Clustal(W/Omega) and Blast compute approximate MSAs with probabilistic models.

Algorithmically, MSA boils down to the cost-optimal alignment of strings. Smaller problems can be solved optimally and the dynamic programming solution relates to

approximate string matching. Precursor work in AI showed considerable scaling but often neglects biological relevant features like the inclusion of similarity cost matrices and affine gap costs. Exceptions are iterative deepening dynamic programming (Schrödl 2005), its externalization (Edelkamp and Kissmann 2007), and a search variant using partial expansion (Hatem and Ruml 2013). Still, the memory requirements raise exponentially with the problem complexity (measured in the sum of the input sequences).

In this paper we apply fixed-memory-bound randomized search that incorporates no expert knowledge in form of refined heuristics. The algorithm that we chose has successfully been used for vehicle routing (Cazenave 2012; Edelkamp and Gath 2014). It applies a series of random walks (rollouts) and learns a mapping (policy) for sampling the search space. It is able to improve over existing solutions and incorporates initial alignments into the search. As other algorithms are memory-bound, with its low memory profile it can serve as an add-on over existing approaches.

The paper is structured as follows. First, we provide a concise formulation of the MSA problem. Next, we consider the implementation of Monte-Carlo tree search that we have adapted to MSA optimization. Experimental results in the BALiBASE benchmark show advances to the state-of-the-art.

Problem Formulation

We start with some formal definitions.

Definition 1 (Sequence Alignment) *Given a set of n sequences $S = \{s_1, s_2, \dots, s_n\}$ with $s_i \in \Sigma^*$ for all $i = 1, 2, \dots, n$, and Σ being a final alphabet. A sequence alignment (of length k) consists of a set of n sequences $A = \{a_1, a_2, \dots, a_n\}$ with $a_i \in \Sigma'^*$ for all $i = 1, 2, \dots, n$, where $\Sigma' = \Sigma \cup \{“-”\}$ and $“-” \notin \Sigma$. For each aligned sequence $a_i \in A$ we have length $|a_i| = k$. If all letters $“-”$ are removed from $a_i \in A$, we get back s_i . For $n = 2$, the alignment is pairwise, for $n > 2$ multiple.*

Definition 2 (Gap, Number, Length, Position) *A gap G consists of a single or a sequence of letters $g = “-”$.*

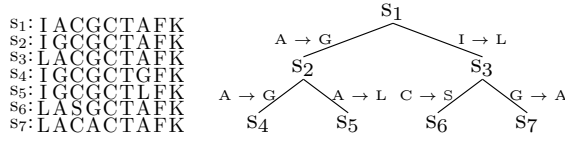


Figure 1: An MSA and its phylogenetic tree.

Moreover $\text{gaps_num}(a_i)$ is the number of empty letters in the aligned sequence $a_i \in A$ and $|G|$ the length of gap G . Particularly we have $|G| = 1$ for $G = \langle g \rangle$ and letter g is located at position $\text{gap_pos}_i(g)$ in sequence $a_i \in A$.

For DNAs the alphabet Σ_{DNA} is $\{A, G, C, T\}$ denoting the nucleo bases adenin, guanin, cytosin and thymine. For RNA the nucleo base uracil, abbreviated by U is used instead of thymine, so that $\Sigma_{RNA} = \{A, G, C, U\}$. The protein alphabet contains 20 amino acids.

In an alignment all sequences are written on top of each other such that the number of columns with matching letter is maximized. Gaps are inserted to slide letters in the alignment. A substitution occurs, if two different letters meet; a gap is a deletion and/or an insertion of a letter and called *indel*. The assumption is that the alignment with the least number of indels is biologically most plausible.

Fig. 1 shows an example of a protein MSA with $n = 7$ having no gaps, and the according phylogenetic tree where internal nodes denote the ancestor sequences, where I (Isoleucine), L (Leucine), F (Phenylalanine), K (Lysine) and S (Serine) are the one-letter abbreviations for the amino acids. To judge the quality of an MSA an evaluation function is required.

Definition 3 (Evaluation Function) An evaluation is a function $F : A \rightarrow \mathbb{R}$. For a pairwise alignment $A = \{a_1, a_2\}$ with $a_i = \langle c_{i1}c_{i2} \dots c_{ik} \rangle$ and $c_{ij} \in \Sigma'$, $i = 1, 2$ and $j = 1, 2, \dots, k$, its evaluation is the sum of similarities f of all alignment columns $F(A) = F(a_1, a_2) = \sum_{j=1}^k f(c_{1j}, c_{2j})$. For a MSA $A = \{a_1, a_2, \dots, a_n\}$ the evaluation is the sum of values for all sequence pairs $F(A) = F(a_1, a_2, \dots, a_n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n F(a_i, a_j)$.

The evaluation function by (Levenshtein 1966) is used to compute *edit distances*. For DNA alignment we support scoring matrices used in WU-BLASTN (Altschul et al. 1990) and FASTA (Pearson 1994), and for protein alignment the PAM (Point Accepted Mutation) matrix (Dayhoff, Schwartz, and Orcutt 1978; Boeckenhauer and Bongartz 2010), the PET91 matrix (Jones, Taylor, and Thornton 1992), and BLOSUM (BLOck SUBstitution Matrix) (Henikoff and Henikoff 1992).

Definition 4 (Optimal MSA) Let \mathcal{A} be the set of all MSAs that can be generated by a set of sequences $S = \{s_1, s_2, \dots, s_n\}$. The optimal MSA $A^* \in \mathcal{A}$ is an MSA with $F(A^*) = \min_{A \in \mathcal{A}} F(A)$, if the evaluation is based on distances or $F(A^*) = \max_{A \in \mathcal{A}} F(A)$.

Definition 5 (MSA Problem) Given a set of sequences $S = \{s_1, s_2, \dots, s_n\}$, the MSA problem is to find the optimal MSA for A^* for S .

For a set of sequences more than one optimal MSAs may exist (Fig.2) yielding different biological explanations. All solutions have the same edit-distance 4. $F(A)$ can calculate not only the similarities (maximum problems) but also the dissimilarities (minimum problems).

-T-CACG GTAGA-G	-TC-ACG GTAGA-G	-TCA-CG GT-AGAG	-TCAC-G GT-AGAG
-TCACG GTAGAG	TCA-CG GTAGAG	TCAC-G GTAGAG	

Figure 2: Two sequences with 7 optimal MSAs.

We consider *affine gap costs* where *gap opening* has cost op and *gap extension* cost ex (per extension), so that gap G has total cost $P(G) = op + ex \cdot |G|$. Unfortunately, for biologists the values of op and ex in this refined cost model may vary (Hodgman, French, and Westhead 2010).

For a rising number of sequences the MSA problem is NP-hard (Wang and Jiang 1994). For n sequences of maximal length q , standard *dynamic programming* (DP) computes an optimal solution with memory $O(q^n)$ and time $O(2^n \cdot q^n)$, so that alternative algorithms are required.

The algorithm iterative-deepening dynamic programming (IDDP) (Schrödl 2005) combines dynamic programming with iterative-deepening A* on the graph representation of the DP matrix. It expands edges not nodes. A lower bound $h(e)$ is devised based on precomputed pattern database of triples. We have $f(e) = g(e) + h(e)$, so that $f(e)$ for an edge e is the estimated cost of a path of the start edge to reach the end edge via the current edge e . IDDP inherits the advantages of DP and IDA*, it has a fixed ordering so that every node is visited once and includes a lower bound for guidance. A partial expansion alternative to IDDP has been proposed and parallelized by (Hatem and Ruml 2013).

Monte-Carlo Tree Search

Monte-Carlo search denotes a class of randomized tree search algorithms that has been designed for search spaces with large node branching factors and weak evaluation functions. By learning the proper choice of successors over time they can converge to the overall optimal solutions. In single-agent search, a series of optimization problems have been solved, e.g., TSPs with Time Windows (Rimmel, Teytaud, and Cazenave 2011; Cazenave and Teytaud 2012; Cazenave 2012) and Morpion Solitaire (Cazenave 2009; Rosin 2011).

Nested Monte-Carlo Search (NMCS) (Cazenave 2009) is a recursive algorithm that contributes to the fact that it is more important to erect the solution on the result of a recursive optimization process than looking at the next step only.

Nested Rollout Policy Adaptation (NRPA) (Rosin 2011) combines NMCS with policy learning. In NRPA we also apply nested search but a state-to-state policy is adapted. The branching being defined by an additional parameter called *iteration*. In every iteration a new random simulation (rollout) is conducted by sampling the policy. Improved solutions induce changes. In each level of the search an individual policy obtains a compromise between exploration and exploitation.

```

1: if  $level = 0$  then
2:    $seq \leftarrow alignment(seq, pol)$ 
3:   return  $(weight(seq), seq, pol)$ 
4: else
5:    $beam \leftarrow \{(\infty, \{\}, pol)\}$ 
6:   for  $N$  iterations do
7:      $new\_beam \leftarrow \{\}$ 
8:     for all  $(v, s, p)$  in  $beam$  do
9:       insert  $(v, s, p)$  in  $new\_beam$ 
10:     $temp\_beam \leftarrow BeamNRPA(level - 1, p)$ 
11:    for all  $(t\_v, t\_s, t\_p)$  in  $temp\_beam$  do
12:       $t\_p \leftarrow adapt(p, t\_s)$ 
13:      insert  $(t\_v, t\_s, t\_p)$  in  $new\_beam$ 
14:     $beam \leftarrow$  the  $B$  best beams in  $new\_beam$ 
15: return  $beam$ 

```

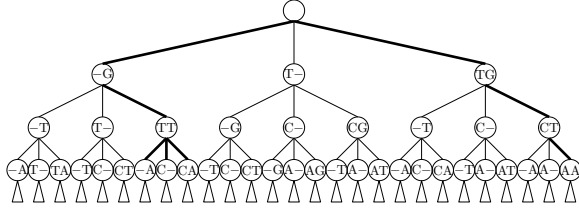


Figure 3: The search tree for a sample pairwise alignment.

Beam Nested Rollout Policy Adaptation (BeamNRPA) (Cazenave 2012) is a variant of NRPA that maintains a policy for each solution, and a set of good solutions for each search level. The size of the set in level i is called *beam* and denoted by B_i . The pseudo-code is shown in Alg. 1. For each solution in a *level* BeamNRPA is called with *level* $- 1$. At the end B_{level} best solutions are generated return, so that the policies in higher search levels can be *adapted*. The adaptation of the policy is based on Bellmann updates and the same as in NRPA. The advantage of BeamNRPA is that it generalizes NRPA and naturally supports prior knowledge in form of solutions seeds.

MCTS for MSA

The intuitive method for the MSA problem is to enumerate possible alignments and after evaluating them, to choose the best one. The search tree can be constructed by a sequence of decisions and solved via NRPA and BeamNRPA. We study two possible encodings.

We assume that each letter v in Σ' has a fixed location $index(v)$, so that for a string $V = \{v_1, v_2, \dots, v_n\}$ in Σ'^* we obtain $index(V) = \sum_{i=1}^n index(v_i) \cdot |\Sigma'|^{n-i}$, where n is the length of V and $|\Sigma'|$ the size of the alphabet.

Construction of All Alignment Columns

An MSA consists of columns. Every column is a string in Σ^n . In the search tree we generate, the root represents an empty node and all other nodes a column in the alignment. Thus, an MSA corresponds to a path from the root to the leaf (Fig. 3, optimal MSAs of Fig. 2 have bold edges).

Algorithm 2: alignment_col(*alignment*, *policy*)

```

1:  $char\_idx \leftarrow \{1, \dots, 1\}$ 
2:  $align\_idx \leftarrow 1$ 
3:  $col \leftarrow alignment.start$ 
4: repeat
5:    $col.num \leftarrow enumeration(col.alternatives, char\_idx, 0)$ 
6:    $sum = 0.0$ 
7:   for  $i \leftarrow 1$  to  $col.num$  do
8:      $value[i] \leftarrow \exp(policy[align\_idx][col.alternatives[i]])$ 
9:      $sum \leftarrow sum + value[i]$ 
10:     $r \leftarrow \text{rand}([0, \dots, sum])$ 
11:     $i \leftarrow 1$ 
12:     $sum \leftarrow value[1]$ 
13:    while  $sum < r$  do
14:       $i \leftarrow i + 1$ 
15:       $sum \leftarrow sum + value[i]$ 
16:     $col.index \leftarrow col.alternatives[i]$ 
17:    transform the index  $col.alternatives[i]$  to the corresponding string
18:    and save in  $col.string$ 
19:    for  $i \leftarrow 1$  to  $n$  do
20:      if  $col.string[i]$  is not a gap then
21:         $char\_idx[i] \leftarrow char\_idx[i] + 1$ 
22:       $align\_idx \leftarrow align\_idx + 1$ 
23:       $col \leftarrow col.next$ 
24: until all sequences are read through
25: return  $alignment$ 

```

During the construction the first step is to recursively enumerate all possible strings that may appear in this column (see Alg. 3). The depth of the tree is n as all strings have to have the same length. In each level for every letter of an alternative string s_i we have a) if all letters have been inserted then the following columns are labeled by a gap (line 6). b) if there are remaining letters that have a fit, then they are inserted to the MSA and the position i in this column either is the corresponding letter in s_i (lines 11–12) or a singleton-letter gap (line 8). Additionally, the number of all alternative strings is returned. Temporary variables *char.idx*[i] store, how many letters have already been inserted to s_i .

In this model we learn, which string should appear in which column. The maximal length of an MSA is the sum of all input strings. A policy in this model is a mapping $(\sum_{i=1}^n |s_i|) \times |\Sigma|^n$ where $|s_i|$ is the length of s_i .

A random MSA is constructed in Alg. 2. Exploiting the policy, a string is randomly chosen (lines 6–18). The variable *align_idx* represents which column is currently constructed. With the variable and the index of an alternative string, we can access the policy value and determine the probability of choosing it. The last step is to update the variables to prepare for the next column (lines 19–23). The steps are repeated until all letters have been inserted, so that all columns are constructed and stored in a list. At the end, the MSA is evaluated and returned (line 25).

The enumeration process is recursive, starting with $seq_idx = 0$ and ending with $seq_idx = n$. As the transformation reads a string of length n , the worst case of Alg. 3 takes $T_{enum}(n) = 2 \cdot T_{enum}(n - 1)$ steps with $T_{enum}(0) = O(n)$. This induces $T_{enum} = O(n \cdot 2^n)$. We see that the time for constructing a column is equal to

Algorithm 3: **enumerate**($A, char_idx, seq_idx$)

```

1: if  $seq\_idx = 1$  then
2:   static  $num \leftarrow 0$ 
3:   static  $str \leftarrow \{0, 0, \dots, 0\}$ 
4: if  $seq\_idx \leq n$  then
5:   if  $char\_idx[seq\_idx] > |s_{seq\_idx}|$  then
6:      $str[seq\_idx] \leftarrow$  the index of the gap character
7:      $enumerate(A, char\_idx, seq\_idx + 1)$ 
8:   else
9:      $str[seq\_idx] \leftarrow$  the index of the gap character
10:     $enumerate(A, char\_idx, seq\_idx + 1)$ 
11:     $str[seq\_idx] \leftarrow$  the index of the  $char\_idx[seq\_idx]$ -th
12:    character in sequence  $s_{seq\_idx}$ 
13:     $enumerate(A, char\_idx, seq\_idx + 1)$ 
14: else
15:    $num \leftarrow num + 1$ 
16:   transform the string  $str$  to the corresponding index
17:   and save in  $a[num]$ 
18: return  $num$ 

```

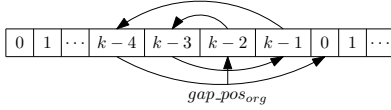


Figure 4: Resolving gap-only columns.

$T_{col} = T_{enum} + 2 \cdot O(2^n) + 2 \cdot O(n) + O(1) = O(n \cdot 2^n)$. Moreover, as we use the sum-of-pairs evaluation we get $T_{eval} = C_n^2 \cdot k = O(k \cdot n^2)$, where k is the length of the sequence alignment. Together we have $T_{colalign} = k \cdot T_{col} + T_{eval} = O(k \cdot n \cdot 2^n + k \cdot n^2) = O(q \cdot n^2 \cdot 2^n)$, with $k = n \cdot q$ being the worst case, and q being the maximal length of all sequences.

Construction of All Alignment Gaps

Def. 1 implies that a sequence alignment is fully determined by the position of gaps. Based on this state representation idea for each sequence s_i the policy is stored as a matrix of size $gap(a_i) \cdot k$, where $gap(a_i)$ is the number of gap letters in the aligned sequence a_i and k the length of the alignment. Again, Monte-Carlo tree search is used to learn, where a gap letter is present in which column of the alignment.

If the length of the alignment is known the number of gap letters can be determined upfront (line 2). Then the positions of all gaps letters can be chosen one after the other (lines 5–17). The temporary variable is_gap helps to determine all legal gap positions (lines 3, 6–11 and 19). The algorithm is executed for all sequences until the entire MSA can be evaluated (line 21). After all gaps in one sequence are done, we can sort them (line 20) which has pros and cons.

We avoid gap-only columns by moving the gap in the longest sequence to $gap_pos_{new} = (gap_pos_{org} + (-1)^i \cdot \lfloor (i+1)/2 \rfloor) \bmod k$, $i = 1, 2, 3, \dots$ (see Fig. 4). We check that there are no gap-only columns left. If no satisfying position can be found, the original one is maintained. Alg. 4 does, however, not cover this special case. Alternatively, we may allow gap columns, as they do not change the score.

The running time of this model is easy to analyze. A

Algorithm 4: **alignment_gap**($alignment, policy$)

```

1: for  $seq\_idx \leftarrow 1$  to  $n$  do
2:    $alignment.gaps\_num[seq\_idx] \leftarrow$ 
      $alignment.length - |s_{seq\_idx}|$ 
3:    $alignment.is\_gap[seq\_idx] \leftarrow \{FALSE, \dots, FALSE\}$ 
4:   for  $gap\_idx \leftarrow 1$  to  $alignment.gaps\_num[seq\_idx]$  do
5:      $sum \leftarrow 0.0$ 
6:     for  $pos \leftarrow 1$  to  $alignment.length$  do
7:       if  $\neg alignment.is\_gap[seq\_idx][pos]$  then
8:          $value[pos] \leftarrow \exp(policy[seq\_idx][gap\_idx][pos])$ 
9:          $sum \leftarrow sum + value[pos]$ 
10:      else
11:         $value[pos] \leftarrow 0.0$ 
12:     $r \leftarrow \text{rand}([0, \dots, sum])$ 
13:     $pos \leftarrow 1$ 
14:     $sum \leftarrow value[1]$ 
15:    while  $sum < r$  do
16:       $pos \leftarrow pos + 1$ 
17:       $sum \leftarrow sum + value[pos]$ 
18:     $alignment.gaps\_pos[seq\_idx][gap\_idx] \leftarrow pos$ 
19:     $alignment.is\_gap[seq\_idx][pos] \leftarrow TRUE$ 
20:    /* sort  $alignment.gaps\_pos[seq\_idx]$  or not */
21: return  $alignment$ 

```

A-CGG		
A--TG	ACGG	A--TG
ATCGG	A-TG	ATCGG

Figure 5: Sample MSA projections.

random alignment is constructed one by one. Sequence s_i contains $k - |s_i|$ gap letters. We obtain $T_{gapalign} = O(\sum_{i=1}^n \sum_{j=1}^{k-|s_i|} (2k)) + T_{eval} = O(q^2 \cdot n^3)$, with $k = n \cdot q$ in the worst case and q being the maximal sequence length.

Construction of an Initial Alignment

In the second model, prior knowledge is requested in the form of the length of the optimized alignment. This information can be supplied by the user or via an initial alignment. This section provides an algorithm to construct an initial alignment automatically (Kurtz 2007).

Definition 6 (Projection) Let $S = \{s_1, \dots, s_n\}$ be a set of sequences and S' a subset of S . Assume $A_S = \{a_1, \dots, a_n\}$ to be an MSA of S . The projection of A_S wrt. S' is the MSA $proj(A_S, S')$, constructed as follows

- all rows in A_S that do not correspond to sequences in S' are removed
- all columns that only contain gap letters are removed.

If $A_{S'} = proj(A_S, S')$, where $A_{S'}$ is an MSA of S' , we say that A_S is compatible with $A_{S'}$.

An example for $S = \{\text{"ACGG"}, \text{"ATG"}, \text{"ATCGG"}\}$, $S' = \{\text{"ACGG"}, \text{"ATG"}\}$ and $S'' = \{\text{"ATG"}, \text{"ACTCGG"}\}$ is shown in Fig. 5. We see an MSA A_S of S , a projection $proj(A_S, S')$, and another projection $proj(A_S, S'')$.

Definition 7 (Alignment Tree) An alignment tree for a set of sequences S is a labeled tree. In this tree the node set is S and every edge (i, j) is labeled by the optimal pairwise alignment of two sequences s_i and s_j .

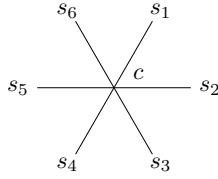


Figure 6: A star alignment tree of sequences $\{c, s_1, \dots, s_6\}$.

Algorithm 5: initial_alignment()

```

1: for  $i \leftarrow 1$  to  $n$  do
2:   for  $j \leftarrow i + 1$  to  $n$  do
3:     compute the optimal alignment of  $s_i$  and  $s_j$  with distance  $d_*(s_i, s_j)$ .
4:   for  $i \leftarrow 1$  to  $n$  do
5:      $total[i] \leftarrow 0$ 
6:   for  $j \leftarrow 1$  to  $n$  do
7:      $total[i] \leftarrow total[i] + d_*(s_i, s_j)$ 
8:    $c \leftarrow \arg \min_i total[i]$ 
9:   choose an arbitrary sequence  $s \in S \setminus \{s_c\}$ 
10:  let  $A$  be the optimal pairwise alignment of  $s_c$  and  $s$ 
11:   $S' \leftarrow \{s_c, s\}$ 
12:  while  $S' \neq S$  do
13:    choose an arbitrary sequence  $s \in S \setminus S'$ 
14:    combine  $A$  with the optimal pairwise alignment of  $s_c$  and  $s$ 
15:     $S' \leftarrow S' \cup \{s\}$ 
16:  return  $A$ 

```

In an alignment tree the relation between all sequence pairs are represented. There are different options for constructing such tree. We consider the special case of the tree being star-shaped (Fig. 6).

The algorithm for constructing an initial MSA has two stages. The basis is a set of precomputed pairwise alignments (see Alg. 5). For each pair of sequences (s_i, s_j) the distance to the optimal alignment is computed (lines 1–5). For each sequence s_i all distances of the optimal alignment corresponding to s_i are added (lines 6–11). The sequence with the minimal total distance is chosen as the center (line 12), all other sequences are leaves.

The second stage is to construct an MSA based on the pairwise alignment stored at the edges. Whenever an MSA of the sequences $\{c, s_1, \dots, s_i\}$ is constructed, the optimal pairwise alignment of c and s_{i+1} is inserted. This insertion preserves the rule *once a gap always a gap*. Therefore, the constructed MSA is compatible with all pairwise alignments in the alignment tree. For example, $c = \text{"ATGCATT"}$, $s_1 = \text{"AGTCAAT"}$ and $s_2 = \text{"ACTGTAATT"}$. The alignments of c and s_1 or c and s_2 are

$$\begin{aligned}
a &= \text{ATGC}-\text{CATT} & a' &= \text{A}-\text{TGC}-\text{ATT} \\
a_1 &= \text{A}-\text{GTCAAT} & \text{and} & & a'_2 &= \text{ACTGTAATT}
\end{aligned}$$

In the second alignment we find a gap prior to letter ‘T’ in d sequence a' . According to the golden rule the gap in a'' is preserved. Through the combination from a and a' we can generate $a'' = \text{"A-TG-C-ATT"}$, so that the final MSA is

The MSA is not optimal as we could substitute a'_2 by “ACTGT-AATT”. It is, however, a good approximation.

Definition 8 (Proper Cost Function) A *similarity*

$$\begin{aligned}
a'' &= \text{A-TG-C-ATT} \\
a'_1 &= \text{A--GTC-AAT} \\
a'_2 &= \text{ACTG-TAATT}
\end{aligned}$$

cost function f is proper if 1) for all $x \in \Sigma'$, we have $f(x, x) = 0$; 2) for all $x, y, z \in \Sigma'$, we have $f(x, z) \leq f(x, y) + f(y, z)$.

Lemma 1 Assume a proper similarity cost function f , and d being the column sum of f , a set of sequences $S = \{c, s_1, \dots, s_n\}$ and a star alignment tree T with center c . If $A = \{a, a_1, \dots, a_n\}$ is an MSA of S with length k that is compatible with all optimal alignments in T , then for all $1 \leq i, j \leq n$ we have $F(a_i, a_j) \leq F(a_i, a) + F(a, a_j) = F_*(s_i, c) + F_*(c, s_j)$.

Proof: We consider column r in MSA A . According to the second property of a proper cost function for an arbitrary letter $b \in \Sigma'$ we have $f(a_i[r], a_j[r]) \leq f(a_i[r], b) + f(b, a_j[r])$. If $b = a[r]$, we have $f(a_i[r], a_j[r]) \leq f(a_i[r], a[r]) + f(a[r], a_j[r])$. The distance of a pairwise alignment is the sum of distances of all columns. Thus,

$$\begin{aligned}
F(a_i, a_j) &= \sum_{r=1}^k f(a_i[r], a_j[r]) \\
&\leq \sum_{r=1}^k (f(a_i[r], a[r]) + f(a[r], a_j[r])) \\
&= \sum_{r=1}^k f(a_i[r], a[r]) + \sum_{r=1}^k f(a[r], a_j[r]) \\
&= F(a_i, a) + F(a, a_j).
\end{aligned}$$

Following the assumption we have that the MSA A is compatible with all optimal alignments in T . Therefore, the projections of A wrt. $\{s_i, c\}$ are optimal alignments of s_i and c . Folling the first property of a proper cost fuction, we have $f(-, -) = 0$, so that the distance of a pairwise sequence alignment does not change if an only-gap column is removed. Hence, $F(a_i, a) = F_*(s_i, c)$, and $F(a, a_j) = F_*(c, s_j)$. \square

Theorem 1 Let $S = \{s_1, \dots, s_n\}$ be a set of sequences, f be a proper similarity cost function, F be the column sum of f , and $A = \{a_1, \dots, a_n\}$ be an MSA of S , constructed via Alg. 5. Then, $F(a_1, \dots, a_n) \leq (2 - \frac{2}{n}) \cdot F_*(s_1, \dots, s_n)$.

Proof: We assume that MSA $A^* = \{a_1^*, \dots, a_n^*\}$ is optimal for S , i.e., $F(a_1^*, \dots, a_n^*) = F_*(s_1, \dots, s_n)$, and $c = s_n$ is the center. We compute the distance between A and A^* .

$$\begin{aligned}
F(a_1, \dots, a_n) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n F(a_i, a_j) = \frac{1}{2} \sum_{i=1}^{n-1} \sum_{j=1}^n F(a_i, a_j) \\
&\leq \frac{1}{2} \sum_{i=1}^{n-1} \sum_{j=1}^n (F_*(s_i, c) + F_*(c, s_j)) \\
&= \frac{1}{2} \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} F_*(s_i, c) + \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} F_*(s_j, c) \right) \\
&= \frac{1}{2} \left(\sum_{j=1}^{n-1} \sum_{i=1}^{n-1} F_*(s_i, c) + \sum_{j=1}^{n-1} \sum_{i=1}^{n-1} F_*(s_i, c) \right) \\
&= (n-1) \cdot \sum_{i=1}^{n-1} F_*(s_i, c)
\end{aligned}$$

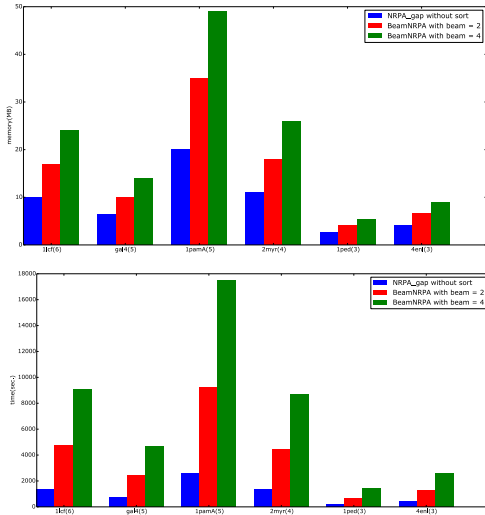


Figure 7: Space (top) and time needed by (Beam)NRPA.

and

$$\begin{aligned}
F(a_1^*, \dots, a_n^*) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n F(a_i^*, a_j^*) \\
&= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n F(a_i^*, a_j^*) \\
&\geq \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n F_*(s_i, s_j) \\
&= \frac{1}{2} \sum_{i=1}^n \left(\sum_{j=1}^n F_*(s_i, s_j) \right) \\
&\geq \frac{1}{2} \sum_{i=1}^n \left(\sum_{j=1}^n F_*(c, s_j) \right) \\
&= \frac{1}{2} n \cdot \sum_{j=1}^n F_*(c, s_j) = \frac{1}{2} n \cdot \sum_{i=1}^n F_*(s_i, c)
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\frac{F(a_1, \dots, a_n)}{F_*(s_1, \dots, s_n)} &= \frac{F(a_1, \dots, a_n)}{F(a_1^*, \dots, a_n^*)} \\
&= \frac{(n-1) \cdot \sum_{i=1}^{n-1} F_*(s_i, c)}{\frac{1}{2} n \cdot \sum_{i=1}^n F_*(s_i, c)} = 2 - \frac{2}{n}. \square
\end{aligned}$$

The MSA that is constructed via the star-shaped alignment tree is, therefore, an upper bound for the distance of the optimal MSA (Kurtz 2007).

Experimental Results

Experiments were ran on a Debian v7.8 32 GB RAM PC (using 1 of the AMD FX(tm)-8350's 4.0/4.2GHz 8-cores), taking GNU's g++ (v4.7.2, -O3). For scoring, PAM250 and affine gap cost wrt. $-10x - 1$ for gap length x were used.

We took the BALiBASE benchmark (<ftp://bess.u-strasbg.fr/pub/BALiBASE2>), which has been designed to

compare the efficiency of different MSA algorithms¹. BALiBASE is a library of biologically alignments that optimize an informal biological *meaning*. Having a formal sum of pairwise scores on BALiBASE entries cannot replace a comparison with bioinformatics competitors such as Clustal-Omega (Clu 2011), MUSCLE (Edgar 2004a; 2004b) or MAFFT (Katoh 2013). However, our interest was showing the potential of MCTS for the MSA problem in terms of saving space and posthoc optimization during the search. Originally, we wanted to compare our algorithm with Genetic Algorithms (e.g., the program SAGA). But we did not do it, due to the non-optimal results for the search without the initial alignment.

Reference 1 consists of 82 sequence groups, partitioned into 9 classes according to the length (short, medium, long) and similarity (large, medium, small). Among those we chose *test* 3, consisting of 28 sequence groups with three to six sequences of different similarity. From the set of MSAs we chose 1ped and 4enl (3 sequences) and 1lcf (6 sequences), together with the groups 2myr (4), ga14 (5), and 1pamA (4), which are supposedly the hardest (Hatem and Ruml 2013; Schrödl 2005). The implementation supports FASTA and MSF formats. The web presentation comes with manual close-to-optimal solutions.

For these sequence groups at most 20MB RAM was allocated, which is by far lower than the one in IDDP and variants. On the other hand, BeamNRPA was better than NRPA: the wider the beam, the better the solution. The number of rollouts for BeamNRPA its $beam \cdot iteration^{level}$ (we allow a beam width other than 1 only in level 1), and chose $beam = 1, 2, 4$, $iteration = 50$ and $level = 3$. BeamNRPA with $beam = 1$ is NRPA. The initial alignment is defined by the star algorithm and improved by the optimizer.

In NRPA_col a policy is a matrix of size $(\sum_{i=1}^n |s_i|) \times |\Sigma|^n$, so that the memory requirements are exponential in n . This leads NRPA_col to fail for 5-6 sequences and to bad results in many others.

For NRPA_gap a policy is a matrix of size $(k - |s_i|) \times k$ for every s_i , so that memory requirements are polynomial in $|s_i|$ and k . Only 4 of 28 groups needed more than 10MB space, and 20MB was the overall maximum. For DP and its variants the space complexity is $O(|s_1| \cdot \dots \cdot |s_n|)$. A biological sequence (DNA/protein) may have over one thousand bases/amino acids. Hence, the memory requirements are huge. Our algorithm saves only the positions of all gaps in an alignment. Obviously, the number of gaps is much less than the length of an aligned sequence. Therefore, the required memory in our program is very small.

Sample learning curves for 1ped and 1pamA are shown in Fig. 8 and Fig. 9), respectively. NRPA_gap without sorting often resulted in a better quality than with sorting, where 1pamA, 2myr and 1lcf are the only exceptions (see Table 1). Thus, we used no sorting in BeamNRPA. Memory and time performances of NRPA and BeamNRPA are cross-compared

¹BALiBASE3

(<http://www.ncbi.nlm.nih.gov/pubmed/16044462>) is considered by specialists as a bad benchmarking resource even for identifying good scoring functions. Moreover, BALiBASE version 2 is used in all precursing AI publications.

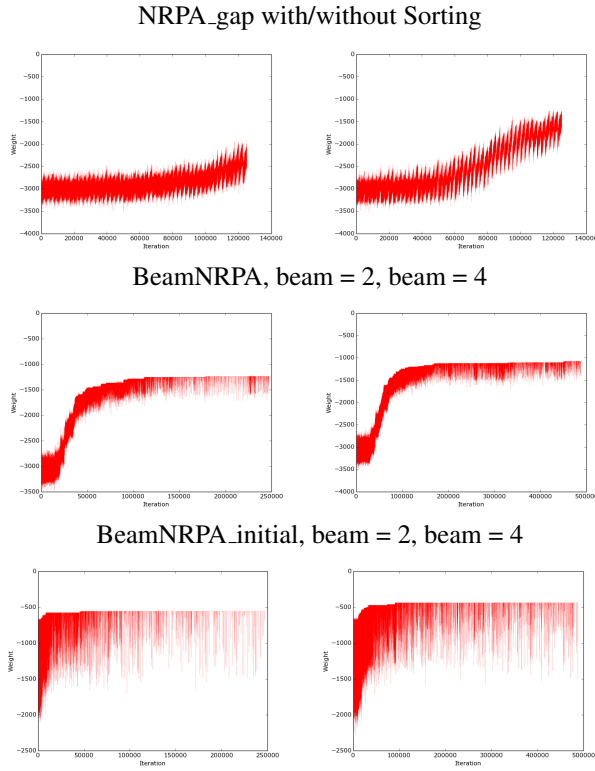


Figure 8: Learning curves of *lped*.

in Fig. 7 and listed in Table 1 and 2. The wider the beam, the higher the computational cost. On the other hand, as shown in Fig. 8, the larger the search tree, the better the solution found by BeamNRPA.

Next, we tested whether an initial alignment could be improved (see Table 3). After determining the alignment, we called the adaptation function 10 time (α -value of 1) to come up with an initial policy. For the sequence group *lped* an alignment better than the initial one was found quickly (see Fig. 8). The initial alignment of *lpamA* had a score of -8291. Unfortunately, for this hardest group BeamNRPA did not improve much within the given parameter range (see Fig. 9).

Finally, we optimize best-known solutions from the BALiBASE benchmark with BeamNRPA_gap. Table 4 shows an improvement (wrt. our cost function) in 20 groups, equal results in 6 groups (1ac5, 1bgl, 1dlc, 1fieA, 1gpb, 1gtr) and worse result in 2 groups (1pamA and 1taq).

Altogether there are 28 sequence groups. For the groups *lpamA* and *ltaq* our program cannot return a better solution than BALiBASE (from beam = 2 and 4). For these 6 groups (1ac5, 1bgl, 1dlc, 1fieA, 1gpb and 1gtr) our program returns the same good solutions as BALiBASE (from beam = 2 and 4). For the other 20 groups the better solutions are found from beam = 2 or 4 (beam = 2 sometimes can return a better solution than beam = 4).

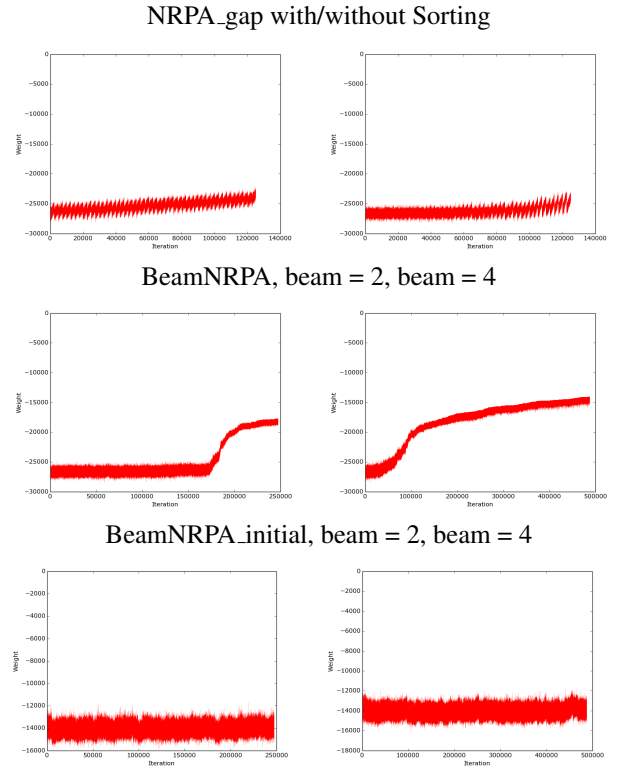


Figure 9: Learning curves of *lpamA*.

Conclusion and Outlook

In this paper we pioneered Monte-Carlo tree search for the multiple sequence problem. The results for learning gaps with BeamNRPA are very promising. The approach has a very low memory overhead, can be used from scratch and for post-hoc optimization. Wrt. our cost function we found improvements to many published BALiBASE alignments.

It is possible to improve the policy representation by learning inter-dependencies of gap positions within the set of sequences. A further yet unexplored option is the parallelization of BeamNRPA. In (Rosin 2011) it has been said that parallelizing NRPA is involved, since the policy has to be shared among the threads. The advantage of BeamNRPA is that it is easier to parallelize as all policies in the beam can be read and updated concurrently. It has the additional feature that it can be parallized in every level of the search. As the number of iterations is usually larger than the number of threads, the searches in each thread are iterative. Another option to deal with concurrency issues in the parallelization is to use low-level compare-and-swap.

Table 1: NRPA_gap

	with sorting				without sorting			
	len	score	time	mem	len	score	time	mem
lajsA	433	-6456	573	5524K	434	-4871	579	5516K
lcpt	455	-5711	471	4926K	458	-4509	477	4656K
llvl	506	-7335	761	6778K	510	-6709	767	6770K
lpamA	656	-22053	2546	20M	677	-22877	2568	20M
lped	385	-1909	223	3022K	386	-1239	225	2748K
2myr	543	-9800	1308	11M	546	-9890	1324	11M
4enl	433	-2701	407	4256K	426	-2031	412	4250K
gal4	431	-10423	720	6604K	433	-8866	736	6600K
lac5	517	-7690	708	6390K	519	-6932	708	6386K
ladj	421	2931	71	1609K	421	2954	69	1612K
lbg1	1002	-7085	746	6402K	1002	-6403	750	6394K
ldlc	636	-6008	585	5556K	637	-5683	588	5544K
left	420	-3371	316	3432K	419	-2658	318	3422K
lfieA	689	-641	221	2808K	689	-268	222	2800K
lgowA	542	-7471	692	6378K	541	-6706	700	6370K
lpkm	466	-2231	213	2812K	468	-1534	214	2800K
lsesA	463	-6949	373	3848K	465	-5766	376	3838K
2ack	534	-11462	752	6594K	534	-10214	757	6586K
arp	449	-8972	507	4912K	449	-7536	511	4904K
glg	513	-8423	508	4922K	514	-7127	513	4916K
lad3	459	-2086	172	2350K	459	-277	173	2342K
lgpb	854	-9015	847	7012K	854	-8726	867	7002K
lgr	451	-3715	230	2800K	451	-1842	236	2792K
llcf	747	-20636	1361	10M	747	-20645	1374	10M
lrthA	556	-1284	269	3004K	556	-318	270	2998K
ltaq	948	-17728	1656	13M	950	-16778	1667	13M
3pmg	588	-2868	329	3656K	589	-2105	330	3648K
actin	415	-4411	272	3238K	415	-3619	273	2964K

Table 3: BeamNRPA_gap with initial alignment

	<i>beam</i> = 2				<i>beam</i> = 4			
	len	score	time	mem	len	score	time	mem
lajsA	457	-2663	2126	11M	459	-2680	4169	15M
lcpt	468	-937	1669	9792K	467	-828	3313	12M
llvl	501	-2027	1915	11M	502	-1961	4117	13M
lpamA	730	-11736	12350	53M	728	-11896	23831	73M
lped	402	-556	722	5128K	402	-430	1447	6664K
2myr	598	-4788	6150	26M	595	-4501	11504	37M
4enl	425	-892	997	6124K	427	-903	1959	8228K
gal4	492	-4643	4813	22M	488	-4342	8832	30M
lac5	551	641	3084	13M	545	802	6090	19M
ladj	432	3210	479	9552K	429	3392	964	9552K
lbg1	1072	1958	7248	47M	1071	3890	13190	47M
ldlc	655	2555	2550	18M	654	2615	5029	18M
left	419	1355	957	8572K	417	1440	1888	8576K
lfieA	702	5565	1147	22M	703	5567	2250	22M
lgowA	542	1138	1975	12M	542	1225	3925	15M
lpkm	474	1809	834	10M	473	2081	1652	10M
lsesA	494	2917	2238	16M	488	3379	4390	16M
2ack	561	-509	3557	20M	556	-215	7039	22M
arp	490	435	3209	15M	488	622	6341	20M
glg	553	2568	3222	19M	551	2620	6376	19M
lad3	464	5133	611	10M	463	5121	1210	10M
lgpb	877	17561	4097	53M	878	17578	7891	53M
lgr	466	7671	1162	16M	465	7658	2289	16M
llcf	799	2330	8778	58M	799	3392	17135	58M
lrthA	565	8897	1120	23M	563	9022	2202	23M
ltaq	978	1889	7947	62M	977	1879	15483	62M
3pmg	619	6744	2006	16M	620	6731	3936	16M
actin	416	7883	824	13M	416	7916	1622	13M

Table 2: BeamNRPA_gap

	<i>beam</i> = 2				<i>beam</i> = 4			
	len	score	time	mem	len	score	time	mem
lajsA	437	-4262	1810	9080K	432	-3684	3546	12M
lcpt	457	-3766	1483	7656K	452	-2857	3024	10M
llvl	502	-4693	2517	10M	497	-3833	4935	15M
lpamA	665	-17679	9235	35M	665	-14016	17458	49M
lped	388	-1209	677	4224K	383	-1075	1399	5556K
2myr	532	-6520	4469	18M	536	-5930	8662	26M
4enl	433	-1796	1302	6760K	424	-1381	2571	9128K
gal4	431	-7133	2459	10M	429	-6751	4686	14M
lac5	519	-4733	2351	10M	513	-4304	4598	13M
ladj	421	1594	205	2188K	421	2804	409	2548K
lbg1	1002	-2510	2602	10M	1002	-892	5128	13M
ldlc	636	-2295	1935	8752K	636	-1986	3775	11M
left	420	-1618	993	5092K	420	-1361	1943	6780K
lfieA	689	3033	705	4012K	689	3652	1345	5216K
lgowA	537	-4415	2413	10M	537	-3049	4628	14M
lpkm	466	207	655	4240K	467	-132	1280	5164K
lsesA	465	-3060	1192	6052K	464	-1713	2329	8112K
2ack	533	-6652	2537	10M	534	-5442	4960	14M
arp	447	-5633	1632	7864K	447	-5169	3277	11M
glg	513	-4263	1632	6888K	514	-2804	3202	10M
lad3	459	906	520	3368K	458	690	1042	4548K
lgpb	854	257	2905	11M	854	1708	5555	15M
lgr	451	759	715	4036K	451	2076	1385	5368K
llcf	747	-8938	4725	17M	747	-6393	9068	24M
lrthA	556	3992	825	4592K	555	4744	1624	5976K
ltaq	950	-9353	5771	22M	950	-7945	11080	32M
3pmg	589	606	1055	5580K	589	1632	2055	7444K
actin	414	-255	854	4776K	414	991	1667	6248K

Table 4: BeamNRPA_gap for BALiBASE optima (BBO)

	BBO	<i>beam</i> = 2				<i>beam</i> = 4			
	score	len	score	time	mem	len	score	time	mem
lajsA	-1292	449	-1264	1698	9852K	449	-1258	3378	13M
lcpt	520	461	558	1397	8440K	461	602	2750	10M
llvl	-750	516	-750	2284	12M	516	-720	4522	16M
lpamA	-2366	677	-5252	8715	39M	678	-3290	17215	54M
lped	-42	398	-15	647	4548K	396	40	1274	5956K
2myr	-1490	554	-1561	4018	21M	554	-1452	8048	28M
4enl	-336	441	-293	1164	7428K	438	-265	2298	9804K
gal4	-876	439	-811	2168	11M	438	-779	4283	15M
lac5	2375	524	2375	2141	11M	524	2375	4247	15M
ladj	4037	421	4064	200	2192K	421	4087	395	2556K
lbg1	7394	1002	7394	2263	11M	1002	7394	4505	15M
ldlc	4906	638	4906	1733	9724K	638	4906	3419	11M
left	2211	412	2257	921	6100K	412	2257	1831	7232K
lfieA	6815	689	6815	640	4300K	689	6815	1279	5628K
lgowA	2710	546	2712	2112	11M	545	2730	4135	15M
lpkm	2981	468	2981	617	4252K	468	2984	1231	5428K
lsesA	5896	465	5896	1086	6488K	465	5907	2167	8520K
2ack	3470	536	3473	2321	11M	536	3473	4542	15M
arp	3875	450	3889	1492	8556K	450	3891	2974	11M
glg	4959	514	5007	1502	9268K	513	5109	2937	10M
lad3	5409	459	5415	491	4100K	459	5426	982	4752K
lgpb	20141	854	20141	2605	12M	854	20141	5145	17M
lgr	8807	451	8807	665	4320K	451	8807	1321	5660K
llcf	25001	747	25007	4168	19M	747	25015	8268	26M
lrthA	10400	556	10475	788	4940K	556	10472	1575	6336K
ltaq	13545	949	13048	5222	25M	949	13300	10482	34M
3pmg	7867	589	7869	956	6080K	589	7868	1899	7912K
actin	8489	415	8556	793	5108K	415	8530	1575	6620K

References

- Altschul, S. F.; Gish, W.; Miller, W.; Myers, E. W.; and Lipman, D. J. 1990. Basic local alignment search tool. *Journal of Molecular Biology* 215(3):403–410.
- Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.
- Boeckenhauer, H.-J., and Bongartz, D. 2010. In *Algorithmic Aspects of Bioinformatics*. Springer. 94–96.
- Cazenave, T., and Teytaud, F. 2012. Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In *Learning and Intelligent Optimization*. Springer. 42–54.
- Cazenave, T. 2009. Nested monte-carlo search. In *IJCAI*, 456–461.
- Cazenave, T. 2012. Monte carlo beam search. *Computational Intelligence and AI in Games, IEEE Transactions on* 4(1):68–72.
2011. Fast, scalable generation of high quality protein multiple sequence alignments using clustal omega. *Mol. Syst. Biol.* 7(539).
- Dayhoff, M. O.; Schwartz, R. M.; and Orcutt, B. C. 1978. A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*, volume 5. National Biomedical Research Foundation. chapter 22, 345–352.
- Edelkamp, S., and Gath, M. 2014. Solving single vehicle pickup and delivery problems with time windows and capacity constraints using nested monte-carlo search. In *ICAART*, 22–33.
- Edelkamp, S., and Kissmann, P. 2007. Externalizing the multiple sequence alignment problem with affine gap costs. In *KI*, 444–447.
- Edgar, R. C. 2004a. MUSCLE: a multiple sequence alignment method with high accuracy and throughput. *Nucleic acids research* 32(5):1792–7.
- Edgar, R. C. 2004b. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC bioinformatics* 5(113).
- Gusfield, D. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.
- Hatem, M., and Ruml, W. 2013. External memory best-first search for multiple sequence alignment. In *AAAI*.
- Henikoff, S., and Henikoff, J. G. 1992. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences* 89(22):10915–10919.
- Hirschberg, D. S. 1975. A linear space algorithm for computing common subsequences. *Communications of the ACM* 18(6):341–343.
- Hodgman, T. C.; French, A.; and Westhead, D. R. 2010. *BIOS Instant Notes in Bioinformatics*. Garland Science, 2nd edition.
- Jones, D. T.; Taylor, W. R.; and Thornton, J. M. 1992. The rapid generation of mutation data matrices from protein sequences. *Comput. Appl. Biosci.* 8(3):275–282.
- Katoh, S. 2013. MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Molecular Biology and Evolution* 30:772–780.
- Korf, R. E.; Zhang, W.; Thayer, I.; and Hohwald, H. 2005. Frontier search. *Journal of the ACM* 52(5):715–748.
- Kurtz, S. 2007. Lecture notes on basics of sequence analysis.
- Levenshtein, V. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady* 10(8):707–710.
- Pearson, W. R. 1994. Using the fasta program to search protein and dna sequence databases. *Methods in Molecular Biology* 24:307–331.
- Rimmel, A.; Teytaud, F.; and Cazenave, T. 2011. Optimization of the nested monte-carlo algorithm on the traveling salesman problem with time windows. *Applications of Evolutionary Computation* 501–510.
- Rosin, C. D. 2011. Nested rollout policy adaptation for monte carlo tree search. In *IJCAI*, 649–654.
- Schrödl, S. 2005. An improved search algorithm for optimal multiple sequence alignment. *Journal of Artificial Intelligence Research* 23:587–623.
- Wang, L., and Jiang, T. 1994. On the complexity of multiple sequence alignment. *Journal of Computational Biology* 1(4):337–348.
- Zhou, R., and Hansen, E. A. 2002. Multiple sequence alignment using A*. In *AAAI*. Student abstract.
- Zhou, R., and Hansen, E. 2003. Sweep a*: space-efficient heuristic search in partially ordered graphs. In *15th IEEE International Conference on Tools with Artificial Intelligence*, 427–434.