MAGUSCP: Multiple sequence Alignment using Graph clUStering through Constraint Programming

Dieter Vandesande Wolf De Wulf

May 30, 2022

Abstract

This article presents a reimplementation and extension of the contribution in Smirnov and Warnow's (2021) "MAGUS: Multiple sequence Alignment using Graph clUStering". This project was made in the context of the Advanced Methods in Bioinformatics course at the Vrije Universiteit Brussel (VUB).

1 Introduction

In the field of bioinformatics, aligning related DNA, RNA, or protein sequences is a powerful method of comparison. When certain sequences can be reasonably aligned, the general hypothesis is that they are descended from a common ancestral sequence. Hence, one can use the general process of Multiple Sequence Alignment (MSA; Chen, 2005) for comparing new sequences with well-known sequences and thereby extracting their shared information and their significant differences. Through such comparisons, insight can be gained in the context of a variety of tasks such as structure prediction (S. Wang et al., 2017; Senior et al., 2020; Rao et al., 2021; Tunyasuvunakool et al., 2021; Jumper et al., 2021), biological function analysis (Pei, 2008), predictions of mutations (Kumar et al., 2009; Hicks et al., 2011), and of interactions (Tafer & Hofacker, 2008; Chitsaz et al., 2009).

Recent advancement in the field has resulted in methodologies that provide large amounts of data. Therefore, MSA methods must evolve with the size of the ever-increasing data sets. Recent studies have shown that it is possible to accurately align enormous and slowly evolving datasets (Sievers et al., 2013). However, Sievers et al. (2013) show that the accuracy of such large MSAs decreases significantly as the number of sequences increases. Sievers et al. (2013) state that this phenomenon is mostly caused by the unavoidable accumulation of alignment errors. Luckily, this phenomenon can be reduced through iterative refinement and selectively adding sequences. An example of a successful approach is the work of ? (?), who combine an existing MSA tool (SATé; Liu et al., 2009, 2012), i.e., the one that performed best on large datasets according to Sievers et al. (2013), with transitivity, resulting in a tool called PASTA, which stands for "practical alignments using SATé and TrAnsitivity". An important aspect of these tools is that they both implement a divide-and-conquer strategy. By dividing an input set of sequences into subsets, aligning each of these subsets separately, and finally merging the alignments together into an alignment on the complete set, these tools are able to scale towards enormous datasets as well as significantly improve their accuracy. While the accuracy of the final alignment is dependent on the ability to merge subalignments it can also be improved through increasing the number of refinement iterations. Both SATé and PASTA default this to three iterations. Smirnov and Warnow (2021) try to improve on this with MAGUS, which stands for "Multiple sequence Alignment using Graph clUStering". Smirnov and Warnow (2021) present a novel method, called the Graph Clustering Manager (GCM), for merging multiple alignments of disjoint subsets of sequences. Through a variety of evaluations using a collection of both simulated and biological datasets, Smirnov and Warnow (2021) find that a single iteration of MAGUS is almost always able to produce quicker and more accurate alignments. Therefore, MAGUS and its successor, recursive MAGUS (Smirnov, 2021), can be seen as the current state-of-the-art for very large MSAs.

In the context of the Advanced Methods in Bioinformatics course at the Vrije Universiteit Brussel (VUB), the contribution that is presented by Smirnov and Warnow's (2021) is reproduced here. The emphasis lies on the GCM, which Smirnov and Warnow (2021) implement through a tailored version of the A^* graph path finding algorithm (Hart et al., 1968). It is important to note that, to make such an algorithm feasible in the context of aligning enormous sets of sequences, Smirnov and Warnow (2021) use a weighted version of the A^* algorithm. Certain datasets can be so extensively tangled that keeping track of progress and gradually increasing the importance of the heuristic, i.e., weighting, is needed to keep the algorithm feasible. However, because of this, the guarantee of the optimality of the found solution is lost. The experimental evaluations that are presented in this article are divided into two sets. Firstly, the performance of a reimplementation of the A^* algorithm is compared to the original. The results show that the original A^* implementation is not as simple as Smirnov and Warnow's (2021) present it, as the reimplemented version, which is based purely on Smirnov and Warnow's

(2021) description, performs significantly worse on the larger instances. Secondly, a new approach that makes use of Constraint Programming (CP; Rossi et al., 2006) is presented. While such an approach is arguably less ad hoc than the weighted A^* approach, a set of experimental evaluations on a dataset that consists of sequences from the datasets originally used by Smirnov and Warnow's (2021) but cut such that they are of much shorter lengths, shows that it is absolutely infeasible and thus that the ad hoc techniques that are used in the weighted A^* approach are necessary to provide a useful tool.

The rest of this article is structured as follows. Section 2 gives a brief introduction to MSA. Section 3 explains the MAGUS tool as a whole. Section 4 discusses the reimplementation of the A^* approach as well as the experimental evaluations that are performed to compare it to the original approach. Section 5 does the same but for the CP approach. Finally, Section 6 concludes the article with a discussion on the conclusions that can be drawn.

2 Multiple Sequence Alignment

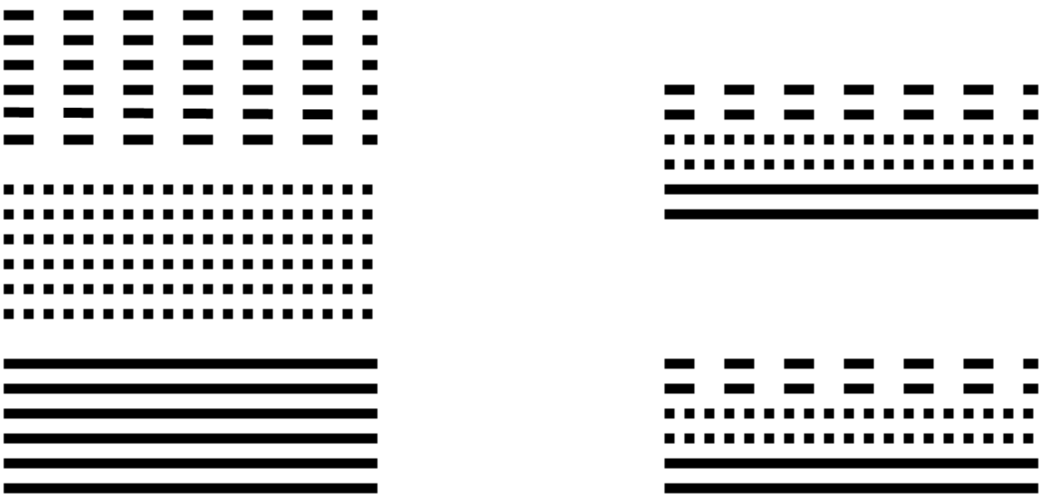
In the context of bioinformatics, sequence alignment can be described as arranging sequences of DNA, RNA, or protein, to find regions in these sequences that correspond. Such correspondence can indicate a functional, structural, or evolutionary relationship. The sequences consist of letters, which, in DNA and RNA, correspond to nucleotides, and in proteins, correspond to amino acids. A very important concept in the context of sequence alignment are mutations. Examples of three important such mutations are point mutations (one single letter changes), insertions (letters are added in the sequence) and deletions (letters are deleted in a sequence). In a MSA, point mutations can be found by looking for different letters in a single column. Insertions and deletions can be found in MSAs by looking for gaps, often denoted using the hyphen ('-') symbol. When a set of sequences has a common ancestor and they are aligned, gaps and point mutations can be seen as the changes that made them diverge from their common ancestor. Vice versa, the aligned letters can be interpreted as conserved information, which can indicate a functional or structural role. Hence, by analysing the quality of an alignment, and thus by investigating gaps and both correctly and incorrectly aligned letters, one can learn a lot regarding both the origin of a set of sequences as well as their meaning.

Aligning sequences can be done in a variety of ways. It goes without saying that aligning sequences manually can only be done with very short or very similar sequences, in which case it is often so that the alignment is not that interesting in general. Therefore, computational methods are applied. Two distinctions are to be made. Firstly, a distinction is made with regard to the method of alignment. Global alignments attempt to align every letter in every sequence. Clearly, global alignments are most useful when the input sequences are similar in content and in length. On the other hand, local alignments try to find regions with high similarity to align while not all regions of the sequences have to be aligned. Local alignments are more suitable for sequences that have dissimilar content and length. A second distinction is made with regard to the number of sequences to be aligned. Pairwise sequence alignments consist of methods that strictly align only two sequences, looking for the best possible matching. While such alignments are handy in simple cases, for example, when looking for similar sequences in a database, they are generally less interesting when compared to their counterparts, i.e., multiple sequence alignments. MSAs are an extension of pairwise alignments that, as the name suggests, try to align more than two sequences. It is important to note that MSAs are complex computational problems and that in most cases, MSAs are NP-complete combinatorial optimisation problems (L. Wang & Jiang, 1994). Nevertheless, MSAs have often resulted into evolutionary insight into, for example, genome annotation (Lander et al., 2001; Schneider et al., 2017) and the phylogenetic relations between organisms (Stackebrandt & Woese, 1984; Woese et al., 1990; Woese, 2000; Pace et al., 2012). This justifies the development of a variety of algorithms that try to solve the MSA problem. Successful examples are dynamic programming, a formally correct but computationally expensive approach and therefore almost never used for aligning more than four sequences (Needleman & Wunsch, 1970; Sankoff & Cedergren, 1983; Lipman et al., 1989; Altschul et al., 1990; Eddy, 2004), progressive methods, which start by aligning the most similar sequences and then gradually incorporate less related sequences (Feng & Doolittle, 1987; Thompson et al., 1994; Notredame et al., 2000; Katoh et al., 2002), and iterative refinement methods, which try to improve an initial alignment in an iterative manner (Barton & Sternberg, 1987; Berger & Munson, 1991; Gotoh, 1993; Edgar, 2004).

For a more detailed introduction to MSAs and the bioinformatics research field, the reader is referred to the work of Mount (2001) and Chen (2005) as well as Istvan's (2022) "The Biostar Handbook". For a review of the current state-of-the-art, the reader is referred to the work of Notredame (2002).

3 MAGUS

MAGUS implements a divide-and-conquer strategy to divide large sets of sequences into disjoint subproblems that are aligned separately. These subproblems are represented as leafs in a tree. Every non-leaf node represents the alignment that is the result of merging the alignments of its children. It is important to note that since all



(a) Constraint alignments. (b) Backbone Alignments

Figure 1: Graphical representation of the process of creating backbone alignments. Left: three constraint alignments, each consisting of 6 sequences. For each of a variable number of backbone alignments, two representative sequences are sampled. The resulting sets of sequences are then aligned. Right: The resulting backbone alignments, each consisting of six sequences. Figure after Smirnov and Warnow (2021).

leaf-nodes represent alignments of disjoint subsets, the alignments of the children of a non-leaf node are always disjoint as well. Consequently, the root node represents an alignment on the complete set of sequences.

To align the sequence subsets at the leaf level, Smirnov and Warnow (2021) make use of a state-of-the-art progressive MSA tool called MAFFT. In theory, any MSA tool could be plugged in for this task. The merging of multiple alignments into a single alignment is performed by Smirnov and Warnow's (2021) Graph Clustering Algorithm. The following paragraphs explain how the GCM works.

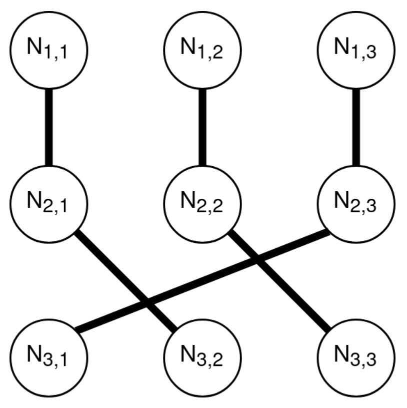
Input The GCM takes as input a disjoint set of subalignments $\{A_1, ..., A_n\}$, which Smirnov and Warnow (2021) denote constraint alignments, as they are not allowed to be changed in the final alignment A_f. In other words, if two sequences $S_i, S_j \in A_c$, the columns of S_i and S_j that are aligned in A_k, are also aligned in A_f.

Backbone Alignments Once the sequence subsets that are associated with the leaf nodes are aligned by MAFFT, information needs to be gathered regarding how to merge the resulting constraint alignments. Smirnov and Warnow (2021) do so by calculating a number of backbone alignments. Representative sequences are sampled, without replacement, from all constraint alignments. Figure 1 gives a graphical representation of this process. The resulting sequence subsets are aligned using MAFFT. Intuitively, it can be seen that the resulting backbone alignments contain information on how to best merge the constraint alignments, since they align sequences from multiple different constraint alignments.

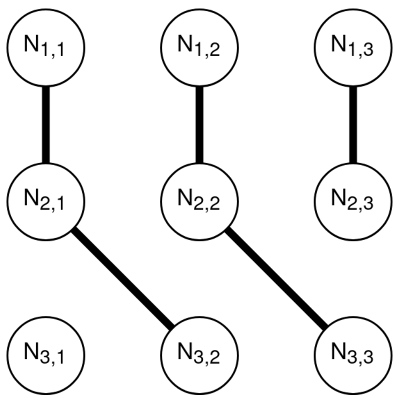
Alignment Graph Using the constraint and backbone alignments, Smirnov and Warnow (2021) establish a graph which they denote the alignment graph. This graph consists of nodes $N_{a,c}$ for each constraint alignment a and column c in the complete alignment. These nodes are connected if there exists at least one pair of letters l_{a_1,c_1} from column c_1 of constraint alignment a_1 and l_{a_2,c_2} from column c_2 of constraint alignment a_2 that occur in the same column in at least one backbone alignment. The weights of the connections correspond to the number of such pairs.

Smirnov and Warnow's (2021) alignment graph can be interpreted as follows: an edge $(N_{a_1,c_1}, N_{a_2,c_2}, w)$ with a relatively high w indicates that, according to the backbone alignments, it would be good to align column c_1 from constraint alignment a_1 with column c_2 from constraint alignment a_2, and vice versa.

Graph Clustering To be able to extract information from the alignment graph, Smirnov and Warnow (2021) cluster the graph using the Markov Clustering Algorithm (MCL; Dongen, 2000). Smirnov and Warnow (2021) claim that the purpose of clustering the alignment graph is twofold. Firstly, it makes the analysis of the alignment graph more tractable, as the connections with the lowest weights are dropped and the nodes that are connected through connections with relatively higher weights are collected. Secondly, it extends the consistency principle in the multiple sequence alignment literature to longer paths within the alignment graph, thereby, finding evidence for homology beyond what can be inferred using just a single third sequence. Due to the way



(a) Clustering with ordering violations.



(b) Valid trace.

Figure 2: A graphical representation of the process of resolving an ordering violation. Connected components depict the different clusters. Left: a set of clusters that contains ordering violations. It is not possible to align the third column of the second alignment with the first column of the third alignment because columns from the same constraint alignment would overlap. To solve these violations, the cluster $\{N_{1,3}, N_{2,3}, N_{3,1}\}$ can be broken into two separate clusters, namely, $\{N_{1,3}, N_{2,3}\}$ and the singleton cluster $\{N_{3,1}\}$. Right: the resulting trace: $\{N_{3,1}\} < \{N_{1,1}, N_{2,1}, N_{3,2}\} < \{N_{1,2}, N_{2,2}, N_{3,3}\} < \{N_{1,3}, N_{2,3}\}$.

the alignment graph is constructed, the found clusters can be seen as collections of nodes that indicate which of the constraint alignments' columns should be aligned.

Resolving horizontal and vertical violations The found clusters can contain three types of violations. Firstly, a vertical violation occurs when two nodes of the same constraint alignment are part of the same cluster, meaning that two columns of the same constraint alignment need to be aligned. Secondly, a horizontal violation occurs when a node is part of multiple different clusters, meaning that the corresponding column and constraint alignment is aligned in two places. To resolve these violations, MAGUS implements a greedy approach that repeatedly removes violating nodes from the cluster that has the smallest sum of weights of incoming edges, until all violations are resolved.

Resolving ordering violations A third possible violation occurs when clusters overlap, these are denoted ordering violations. A correct cluster ordering is one that adheres to the following constraints: let C and C' be clusters with $C < C'$, then $c_1 < c_2$ for all $N_{a,c_1} \in C$ and $N_{a,c_2} \in C'$. Deriving a valid ordering from a set of clusters might require 'breaking' clusters into parts. Smirnov and Warnow (2021) denote a set of clusters that is ordered and contains no horizontal or vertical violations a 'trace'. Figure 2 gives a graphical representation of an example of an ordering violation and how it can be resolved by breaking clusters. Finding a valid ordering while breaking as little as possible clusters is the core of this and of Smirnov and Warnow's (2021) work.

Constructing the complete alignment Given a trace, an alignment on the complete set of sequences can be constructed from the subalignments as follows: column c_1 in constraint alignment a_1 is aligned with column c_2 in constraint alignment a_2 if and only if the nodes N_{a_1,c_1} and N_{a_2,c_2} are part of the same cluster.

4 The A^* approach

Smirnov and Warnow (2021) see the search for a valid trace as a graph path finding problem. Every node in the graph represents a configuration consisting of a set of unordered clusters and a set of ordered clusters. Steps are taken by moving a cluster from the unordered set to the ordered set if such a move is legal according to the cluster ordering defined in the previous section. If no such legal moves exist, a cluster needs to be broken. To find the optimal path in this graph, Smirnov and Warnow (2021) implement a tailored version of the A^* graph path finding algorithm (Hart et al., 1968). The A^* algorithm requires a distance measure and an heuristic. In the case of MAGUS, the distance measure is defined to be the number of clusters in the ordered set, while the heuristic measure is defined to be the number of clusters in the unordered set. It is important to note that this heuristic is admissible, i.e., it never overestimates the distance needed to reach a goal node, which, in this case, is a node with an empty set of unordered clusters. When the heuristic measure used by the A^* algorithm is admissible, it is guaranteed to find an optimal path, which, in this case, is a path with the least possible cluster breaks.

Starting from a traditional A^* implementation, Smirnov and Warnow (2021) make two ad hoc extensions to make the algorithm more efficient. The following two sections discuss them separately. The subsequent third section discusses a set of empirical evaluations that is designed to compare a reimplementation of Smirnov and Warnow's (2021) A^* algorithm to the original. The source code for the reimplementation can be found on a fork repository[1] of the original MAGUS repository[2] on GitHub. The scripts that were used to run the empirical evaluations can be found on an encompassing repository[3] that contains the reimplementation repository as a submodule.

4.1 Generating neighbours

Given a set of clusters C without horizontal or vertical violations, a naive implementation of the A^* star approach would consist of generating all possible neighbours, i.e., looking for all possible legal moves and cluster breaks. However, when reasoning about the underlying problem, Smirnov and Warnow (2021) find that this can be done more efficiently. By creating a search matrix where every row represents a constraint alignment and every column represents a column in the complete alignment and then filling that matrix with unique identifiers for each cluster, one has already ordered all the clusters and the only thing that remains to be done is to break the clusters that overlap. Remember that the set of cleaned clusters C can never contain multiple nodes from the same constraint alignment and thus, in the search matrix, a single cluster identifier can occur at most once per row. By moving through the search matrix, from left to right, keeping track of a single index per row, neighbours can be generated in a much more efficient fashion. This collection of indices is called a frontier (Smirnov & Warnow, 2021). If at any point, a cluster's nodes' indices correspond exactly to those of the frontier, that cluster can be added to the set of ordered clusters, thereby, generating a single neighbour. On the other hand, when not all of a cluster's nodes' indices correspond to those of the frontier, the cluster is sure to overlap with following unordered clusters. In that case, the cluster is broken into two parts. The 'good' part consists of all the nodes whose indices do correspond with those of the frontier, the 'bad' part contains all the other nodes. For each of the constraint alignments, and thus rows in the search matrix, the currently smallest unordered cluster is maintained. The above process of generating neighbours is each time executed for each of these clusters. If at any point a cluster is moved to the set of ordered clusters, the frontier's indices that correspond to that cluster's indices are all moved one position to the right. For more details, the reader is referred to the source code.

4.2 Introducing aggression

When the search matrix is very large, which is often the case in the experiments of Smirnov and Warnow (2021), the original A^* algorithm can quickly become infeasible. To regain feasibility, Smirnov and Warnow (2021) make their A^* implementation weighted. This means that they keep track of progress and when they find that the algorithm is stuck for too long, they increase the importance of the heuristic measure by multiplying it by an aggression factor. While this can make the algorithm finish in reasonable time, it revokes the guarantee of the optimality of the found solution. Nevertheless, in their empirical evaluations, Smirnov and Warnow (2021) find that their weighted algorithm is still able to find relatively good alignments, almost always better than those PASTA finds on the same task.

4.3 Empirical evaluations

In the context of this article, Smirnov and Warnow's (2021) weighted A^* algorithm is reimplemented using the above descriptions, which roughly correspond to the descriptions Smirnov and Warnow (2021) gave in their article. The reimplemented version is compared to the original version through a set of empirical evaluations which correspond to Smirnov and Warnow's (2021) second experiment originally designed to compare MAGUS with PASTA. The used datasets are divided into two sets, each consisting of biological and simulated data. The first set is a set of relatively short instances, each consisting of 1000 sequences or less. The simulated instances are ten 1000-sequence instances from Liu et al. (2009), ranging from 1000M1 (the most difficult) to 1000M4 (relatively easy), each of these instances consist of 20 replicates over which will be averaged. An additional 20 replicates instance of 1000 sequences from the RNASim dataset from Bahr et al. (2001) is used as well. The biological instances are eight protein datasets from BAliBASE (Bahr et al., 2001), where reference alignments are available based on structural features. The second set consists of larger instances. A single 10 replicates instance of 10 K sequences from the RNASim dataset is used for the simulated part. For the biological part, four large nucleotide instances from Cannone et al. (2002) are used, these instances contain up to 24 K sequences. Mirroring Smirnov and Warnow (2021), we report the average of SPFN and SPFP as error measure, computed

[1]`https://github.com/wulfdewolf/MAGUS_CP`
[2]`https://github.com/vlasmirnov/MAGUS`
[3]`https://github.com/wulfdewolf/MAGUS_project`

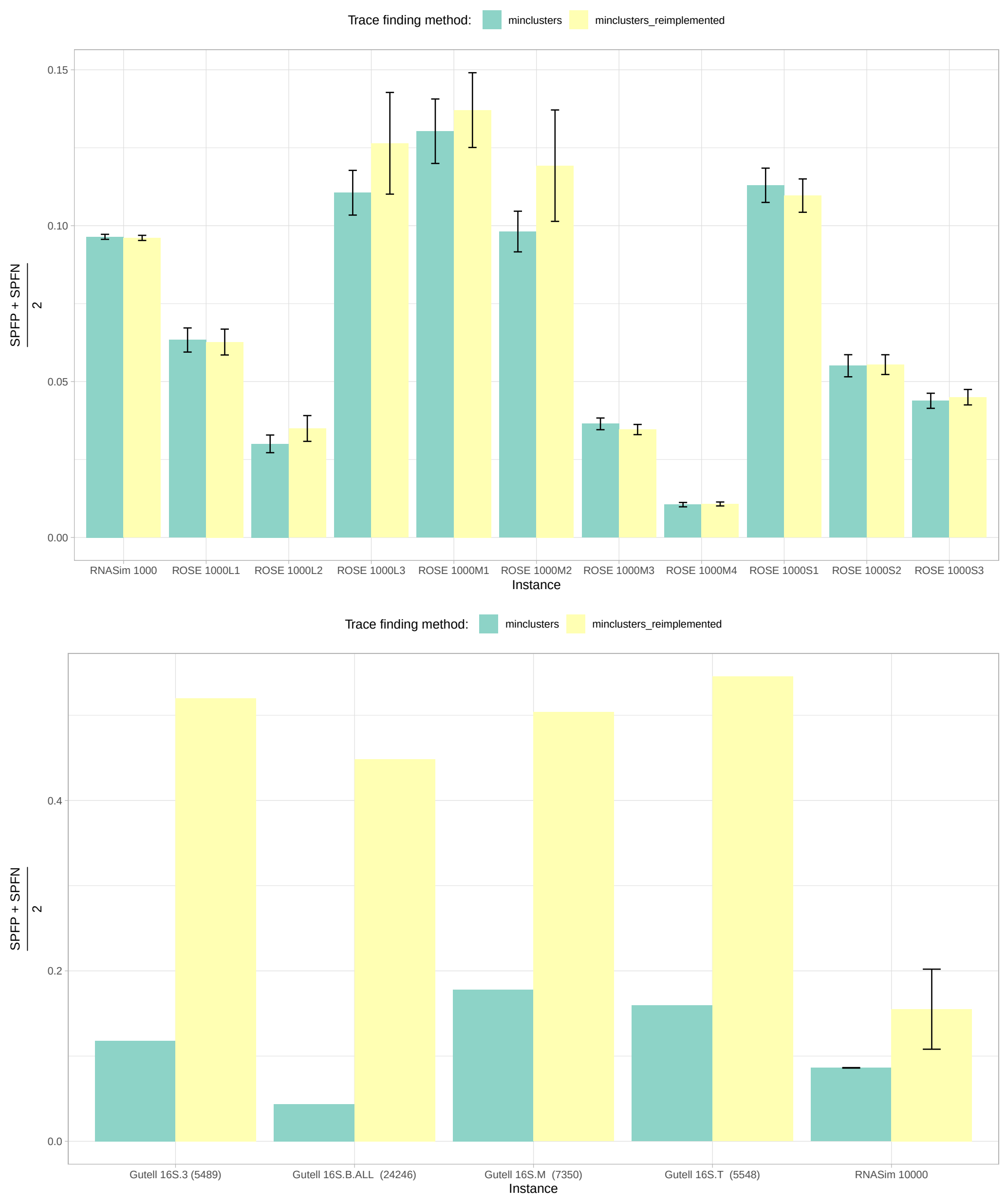


Figure 3: Error scores for the original A^* algorithm versus the reimplemented version. Top: For the 1000-taxon datasets. Results are averaged over 20 replicates. Error bars indicate the standard error. MAGUS is ran in default mode. Bottom: For the larger nucleotide datasets. The number of taxa are given on the x-axis, between parentheses. RNASim is simulated (results are averaged over 10 replicates, error bars indicate standard error), the other datasets are biological. MAGUS is ran with 50 constraint alignment subsets, except for 16S.B.ALL, where it is ran with 200 constraint alignment subsets due to time limitations.

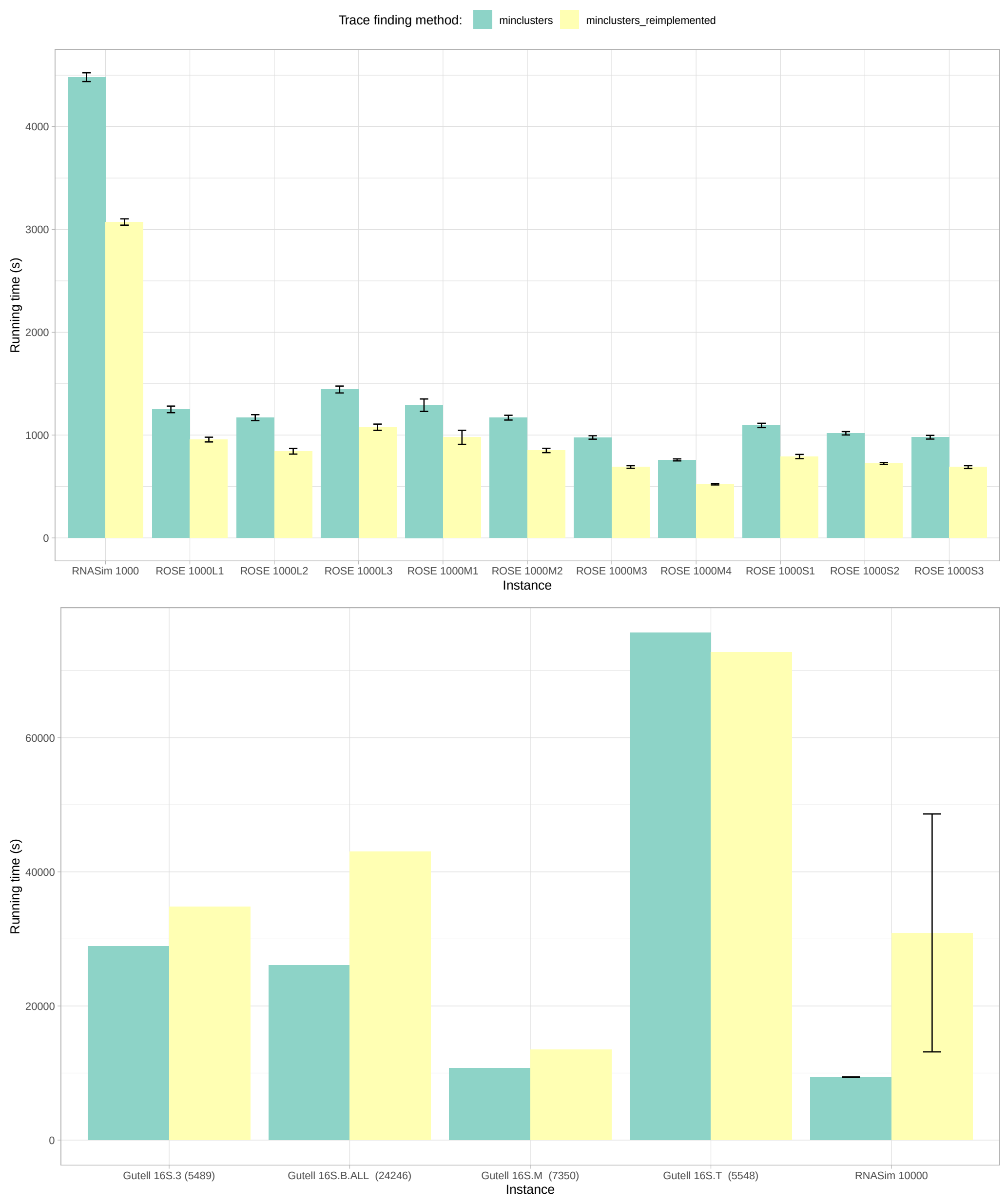


Figure 4: Running times for the original A^* algorithm versus the reimplemented version. Top: For the 1000-taxon datasets. Results are averaged over 20 replicates. Error bars indicate the standard error. MAGUS is ran in default mode. Bottom: For the larger nucleotide datasets. The number of taxa are given on the x-axis, between parentheses. RNASim is simulated (results are averaged over 10 replicates, error bars indicate standard error), the other datasets are biological. MAGUS is ran with 50 constraint alignment subsets, except for 16S.B.ALL, where it is ran with 200 constraint alignment subsets due to time limitations.

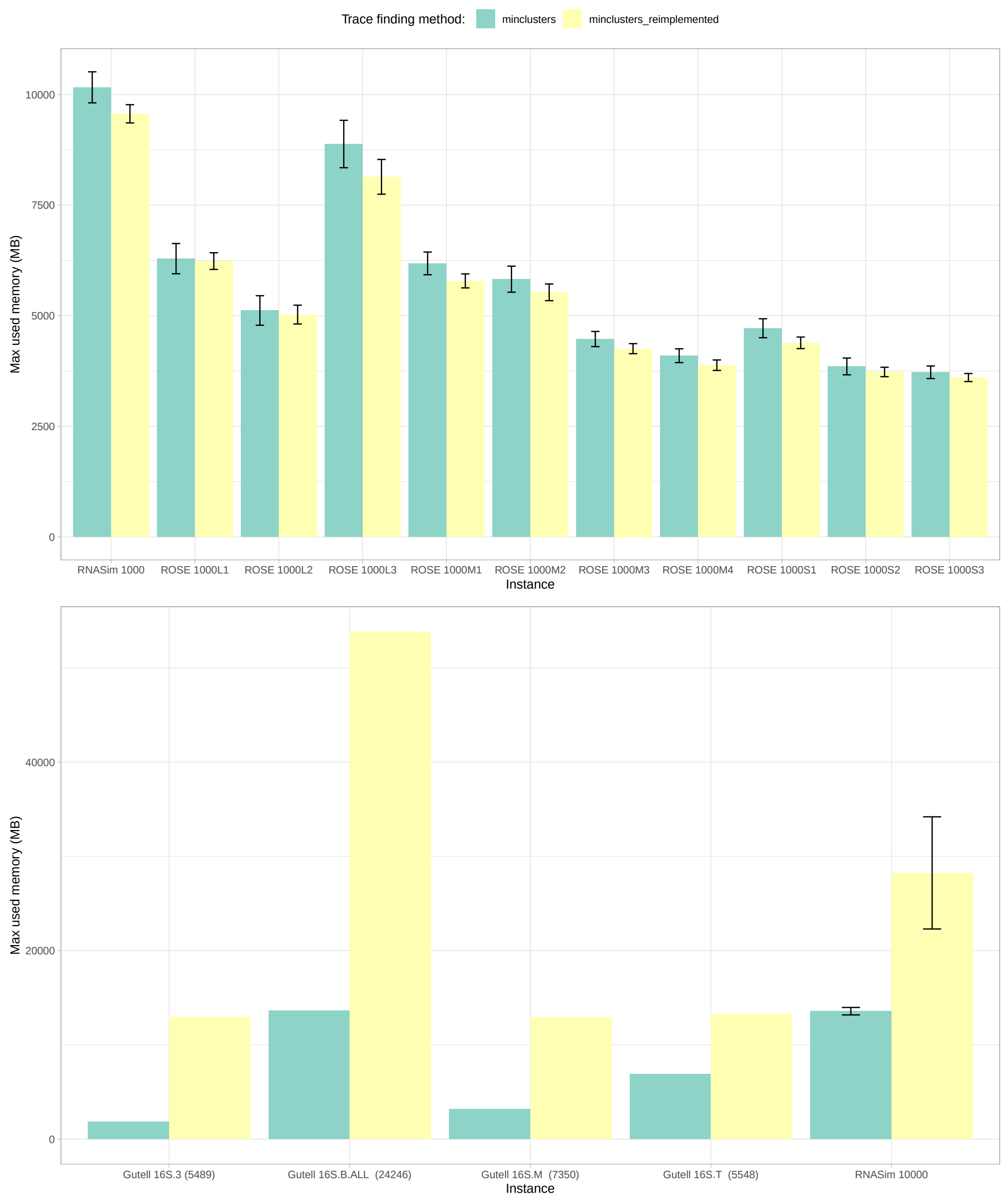


Figure 5: Maximum used memory for the original A^* algorithm versus the reimplemented version. Top: For the 1000-taxon datasets. Results are averaged over 20 replicates. Error bars indicate the standard error. MAGUS is ran in default mode. Bottom: For the larger nucleotide datasets. The number of taxa are given on the x-axis, between parentheses. RNASim is simulated (results are averaged over 10 replicates, error bars indicate standard error), the other datasets are biological. MAGUS is ran with 50 constraint alignment subsets, except for 16S.B.ALL, where it is ran with 200 constraint alignment subsets due to time limitations.

using FastSP (Mirarab & Warnow, 2011). As Smirnov and Warnow (2021) state, SPFN denotes the Sum-of-Pairs False Negative rate, which is the fraction of pairs of homologous letters in the true alignment that are not aligned in the estimated alignment. On the other hand, SPFP is the corresponding Sum-of-Pairs False Positive rate, i.e., the fraction of pairs of aligned letters in the estimated alignment that are not in the true alignment. Additionally, the running time and maximum used memory of every run is reported.

All instances were ran on the VUB Hydra HPC cluster[4] where they were each assigned 16 cores and 16GB of memory.

Results Figures 3, 4, and 5 give the error, running time, and memory, respectively, for all of the mentioned datasets and for both algorithms. For the 1000-taxon instances, always depicted by the top plots, it can be seen that there does not seem to be a significant different between the mean errors of the two algorithms. In fact, a paired Wilcoxon (1945) signed-rank test, in the cases that a preceding Lilliefors (1967) test showed that the paired differences are not likely to be normally distributed ($\alpha = 0.05$), and a paired student t-test in all other cases, finds that for all of the instances (except for ROSE 1000M3) there is no statistically significant difference ($\alpha = 0.05$). This does not hold for the memory and running time results. On the other hand, for the larger instances, always depicted by the bottom plots, there does seem to be a significant difference in error and this for all instances. The differences in running time and memory are present here as well.

Discussion While the results achieved by the original version and the reimplemented version on the shorter instances are mostly the same, this does not hold for the larger instances. A quick glance at the source code of the original implementation clarifies why that is. The original implementation makes use of additional heuristics, while the reimplemented version only uses the heuristic that Smirnov and Warnow (2021) mention in the article, i.e., the number of unordered clusters. The original implementation uses two additional heuristics for when the number of unordered clusters is equivalent. The heuristics in question come down to the following. When the sum of the distance measure and the heuristic measure of two nodes is the same, the original A^* implementation will compare the negated number of unordered clusters, and if that is equivalent, it will compare the negated number of clusters that have been found to overlap during the generation of neighbours. These additional heuristics clarify a lot. For the shorter instances, investigation of the log files of the runs shows that the reimplementation, due to not taking the additional heuristics into account, increases its aggression more quickly as its heap gets filled more quickly without progressing much. Because of the higher aggression, the reimplementation often finishes faster and does so with less used memory, when compared to the original version. As mentioned, the errors for these instances were not significantly different. On the other hand, for the larger instances, the additional heuristics seem to have more effect. The error of the reimplemented version lies significantly higher for all of these instances. The running time and used memory is higher for most of the instances as well. Clearly, the additional heuristics were designed for solving larger instances, which is, coincidentally, also the main goal of the original article. The significant (positive) differences in error when comparing the algorithm with and without the additional heuristics shows that their use is justified.

5 The Constraint Programming Approach

Constraint Programming is a programming paradigm in logic programming that requires writing an encoding of the problem to solve instead of a description of how to solve the problem (Rossi et al., 2006). A CP optimisation problem is defined to be a quadruple $P = (V, D, C, F)$ where $V = \{v_1, ..., v_n\}$ are the variables, $D = \{D_1, ..., D_n\}$ are the domains such that $x_i \in D_i$ for all $i \leq n$, $C = \{C_1, ..., C_m\}$ is the set of constraints, and F is an objective function over V. A constraint C_i is a relation over a subset of variables that refines the domains $D_i \in D$. An example of such constraints can be: for a given $i, j, k \leq n$ (1) variables $\{v_i, v_j, v_k\}$ should have a different value or (2) $v_i < v_j$. A solution to P is an assignment α that maps every variable x_i onto a value $v \in D_i$ that satisfies all constraints in C and for which F is optimal.

As mentioned, Smirnov and Warnow (2021) see the problem of finding a valid trace as a search problem that consists of minimising the number of cluster breaks needed to obtain a valid ordering. In that aspect, CP, which is designed to work with search problems, seems to be a good fit. The implementation of the CP approach is done in the Minizinc language (Nethercote et al., 2007) in combination with the Gecode solver (Gecode Team, 2006).

5.1 A CP encoding of the trace finding problem

A CP encoding of the trace finding problem is established as follows. For each node $N_{a,c}$, we define a variable $v_{a,c}$ with initial domain $D_{a,c} = [1, n]$, with n an upper bound on the number of clusters. The domains of these

[4] https://hpc.vub.be/

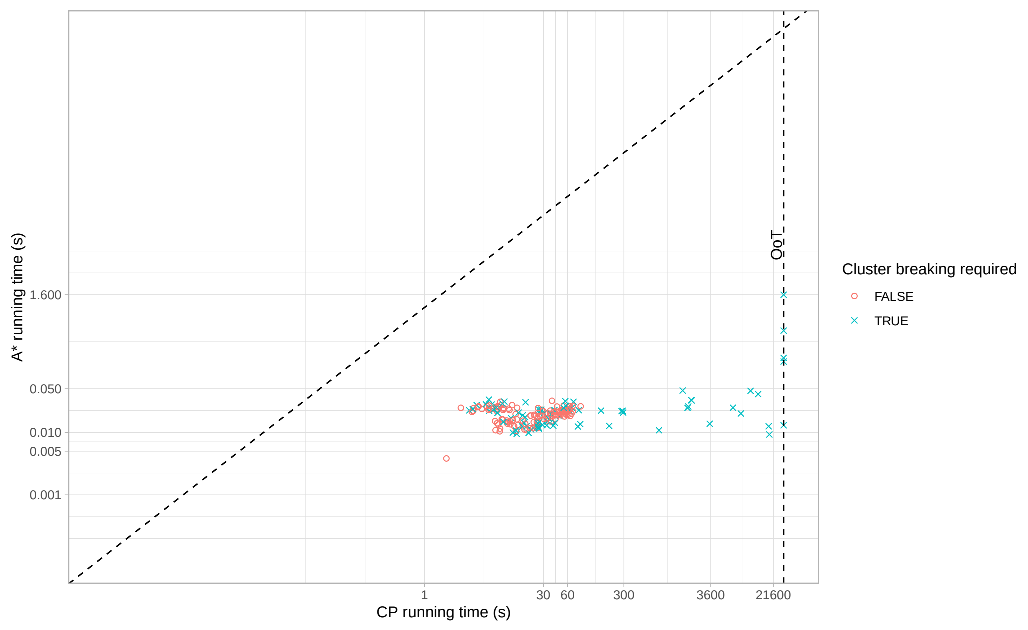


Figure 6: Running time of the original A^* approach versus the CP approach. The legend indicates for which instances MCL found a valid trace and thus for which instances no clusters breaks were necessary.

variables are constrained using two types of constraints. The first type of constraint is implied by the definition of a valid ordering, namely, for two nodes N_{a,c_2} and N_{a,c_2} of the same constraint alignment a with $c_1 < c_2$, it must be that $v_{a,c_1} < v_{a,c_2}$. To reduce the number of constraints, we only add the constraint $v_{a,c_1} < v_{a,c_2}$ if there does not exist a variable v_{a,c_3} such that $c_1 < c_3 < c_2$. Secondly, clusters can only be broken, they can never be merged. This means that for all nodes N_{a_1,c_1} and N_{a_2,c_2}, if they are not in the same input cluster, the constraint $v_{a_1,c_1} \neq v_{a_2,c_2}$ is added to C, indicating that they can also not be in the same output cluster. It is important to note that this type of constraints can make C very large. Let n be the number of nodes. In the worst case, all nodes become singleton clusters. At that point, there would be $O(n^2)$ of this type of constraints.

The used Gecode search strategy is defined as follows. When Gecode has to decide which variable to make a guess for next, it always chooses the variable with the smallest domain. Because of the constraints of the form $v_{i,j} < v_{i,k}$ (with $j < k$), the variables representing the last column of each assignment are guaranteed to have the smallest domain. Gecode then splits that domain in two and excludes the upper half of the domain of the chosen variable.

5.2 Encoding optimisations

Due to the number of constraints and the size of the variable domains, the solving of the encoding described above can quickly become infeasible. To address this, we introduce a number of improvements to the encoding.

Singleton Clusters Since singleton clusters can not be broken, they can be left out of the encoding. This is also done in the implementation of the A^* approach. For all clusters $\{N_{i,j}\}$, the variable $v_{i,j}$ is not created. This optimisations significantly decreases the number of constraints of the form $v_{i,j} \neq v_{k,l}$.

Symmetry breaking Given two variables $v_{i,j}$ and $v_{k,l}$, adding the constraints $v_{i,j} \neq v_{k,l}$ and $v_{k,l} \neq v_{i,j}$ will results in an equivalent solution. Therefore, we only add the constraint $v_{i,j} \neq v_{k,l}$ if $i < k$ or if $i = k$ and $j < l$. Doing this ensures that the constraint $v_{i,j} \neq v_{k,l}$ is only present once.

Refining the domains Preliminary tests showed that solving small instances using the encoding described up until now is infeasible. One way to mitigate this is by enforcing smaller domains on the variables. To do so, before running the CP approach, the original A^* algorithm is ran to obtain an upper bound on the domain of the variables, i.e., the maximum number of clusters allowed in the solution. Afterwards, the CP approach is ran to find an optimal assignment with respect to the number of cluster breaks needed.

5.3 Empirical evaluations

Preliminary tests showed that running the CP approach on the complete instances is infeasible. Therefore, the experimental evaluations that are presented in this section are ran using a selection of instances from the

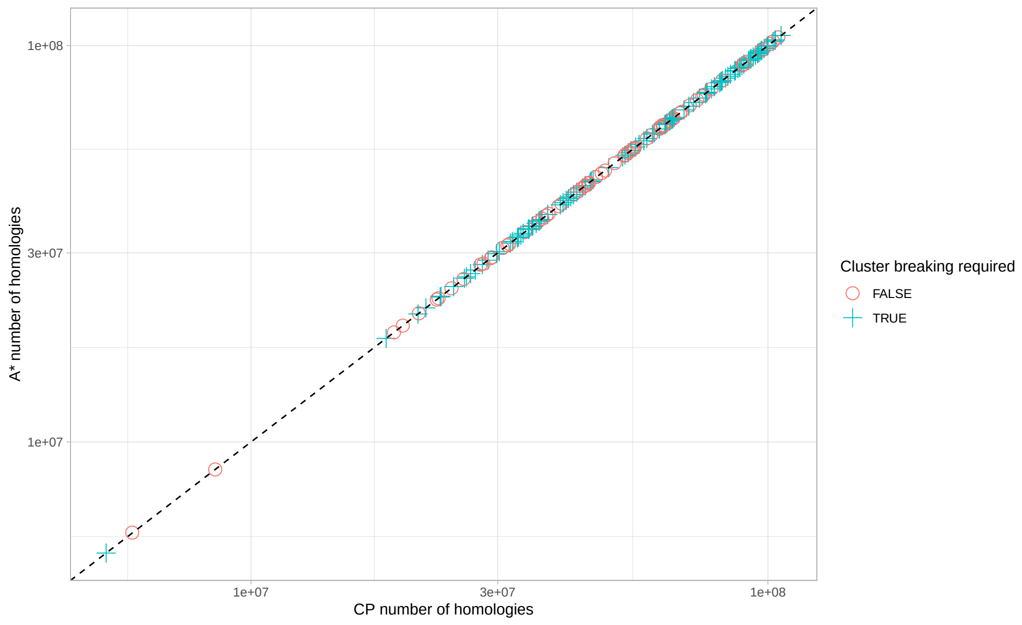


Figure 7: The number of homologies in the computed alignment of the original A^* approach versus the CP approach. The number of homologies is calculated using FastSP (Mirarab & Warnow, 2011). The legend indicates for which instances MCL found a valid trace and thus for which instances no clusters breaks were necessary.

original article that are cut such that they are of much shorter lengths. All biological instances from the BALiBASE dataset and simulated instances from Liu et al. (2009) are considered. The aligned sequences from these instances are clipped to 300 characters (which can be either letters or deletes). The unaligned sequences with less than 40 letters are removed from the instance. Instances with less than 40 remaining sequences are removed. This results in 196 instances with only 4 instances from the BALiBASE dataset. MAGUS is ran without recursively dividing subsets of alignments in smaller subsets. The sequences of the BALiBASE and ROSE instances are divided into 5 and 10 subsets, respectively.

All instances were ran on the VUB Hydra HPC cluster[5] where they were each assigned 16 cores and 16GB of memory.

Results Figure 6 shows a comparison of the running times for both the original A^* approach and the CP approach. For 5 of the 196 instances, Gecode was not able to find the optimal solution within twelve hours. There are no instances for which the number of clusters found by the algorithms differ. For 118 of the 196 instances, the Markov Clustering Algorithm already finds a clustering that is a valid trace. In these cases, Gecode can generally solve the instances more quickly.

Figure 7 shows a comparison of the number of homologies in the final alignment for both the original A^* approach and the CP approach. A homology is defined to be a pair of letters from the input sequences that occur in the same site (Mirarab & Warnow, 2011). Out of the 191 instances that finished, only 19 instances differ in the number of homologies found by both approaches.

Discussion The fact that the CP approach can never find a better solution than the original A^* approach shows that both approaches provide the optimal solution with respect to the number of cluster breaks. As mentioned, when dealing with larger instances, it will often be the case that the original A^* approach gives up the optimality of its solution. Sadly, the performance of the CP approach is insufficient to be able to handle such larger instances.

It is remarkable that Gecode can solve instances where no cluster breaks are needed more quickly compared to when breaks are needed. We conjecture that the reason behind this can be found in the used search strategy. When there is only one solution, the solver will quickly run into a contradiction. Every time a CP solver encounters a failure, it will stop the current search path and cut out a part of the search tree. The earlier such a contradiction is found, the bigger part of the search tree can be cut out. However, if there are more possible solutions, the parts of the search tree that can be cut out will be relatively small.

The difference in the number of homologies in the final alignment found by the two approaches is relatively small. Even though both approaches find different solutions, since they are optimal with respect to the number

[5]`https://hpc.vub.be/`

of clusters, the number of homologies are similar. This demonstrates that the quality of the alignment is mostly dependent on the final number of clusters.

6 Conclusions

The contribution presented by Smirnov and Warnow (2021) is reenacted. A reimplementation of Smirnov and Warnow's (2021) trace finding algorithm, based purely on the descriptions in the article, is presented. Through a set of experimental evaluations that corresponds to those which Smirnov and Warnow (2021) use to compare MAGUS to PASTA, the reimplemented A^* algorithm is compared to the original. The results indicate that the reimplementation is correct, however, for the larger instances, it is outperformed by the original version. The difference in performance is explained by a number of additional heuristics that are present in the original MAGUS source code but are not mentioned in the article. Since the main goal of the MAGUS tool is to allow for better alignments on very large sequence sets, the use of these additional heuristics is justified as they do result in significantly lower alignment errors. However, their use should have been mentioned by Smirnov and Warnow (2021) in the original article. Additionally, a novel trace finding approach that makes use of constraint programming is presented. Such an approach requires a clear and formal description of the trace finding problem. This description can then be passed to a constraint programming solver, which will look for the optimal solution. Compared to Smirnov and Warnow's (2021) arguably ad hoc solution, the constraint programming approach can be seen as more clean. A set of experimental evaluations that makes use of the datasets from the original article but cut to be much shorter is ran to verify the correctness of the constraint programming model and to analyse its capabilities. While the model is found to be correct, sadly, it is also found to be infeasible. Relatively small instances can be solved but when the instances become larger, running time explodes quite quickly.

References

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990, October). Basic local alignment search tool. *Journal of Molecular Biology*, *215*(3), 403–410. doi: 10.1016/S0022-2836(05)80360-2

Bahr, A., Thompson, J. D., Thierry, J. C., & Poch, O. (2001, January). BAliBASE (Benchmark Alignment dataBASE): Enhancements for repeats, transmembrane sequences and circular permutations. *Nucleic acids research*, *29*(1), 323–326. doi: 10.1093/nar/29.1.323

Barton, G. J., & Sternberg, M. J. (1987, November). A strategy for the rapid multiple alignment of protein sequences. Confidence levels from tertiary structure comparisons. *Journal of Molecular Biology*, *198*(2), 327–337. doi: 10.1016/0022-2836(87)90316-0

Berger, M. P., & Munson, P. J. (1991, October). A novel randomized iterative strategy for aligning multiple protein sequences. *Bioinformatics*, *7*(4), 479–484. doi: 10.1093/bioinformatics/7.4.479

Cannone, J. J., Subramanian, S., Schnare, M. N., Collett, J. R., D'Souza, L. M., Du, Y., ... Gutell, R. R. (2002, January). The Comparative RNA Web (CRW) Site: An online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. *BMC Bioinformatics*, *3*(1), 2. doi: 10.1186/1471-2105-3-2

Chen, Y.-P. P. (2005). Introduction to Bioinformatics. In Y.-P. P. Chen (Ed.), *Bioinformatics Technologies* (pp. 1–13). Berlin, Heidelberg: Springer. doi: 10.1007/3-540-26888-X_1

Chitsaz, H., Backofen, R., & Sahinalp, S. C. (2009). biRNA: Fast RNA-RNA Binding Sites Prediction. In *WABI.* doi: 10.1007/978-3-642-04241-6_3

Dongen, S. (2000). Graph clustering by flow simulation. *undefined.*

Eddy, S. R. (2004, July). What is dynamic programming? *Nature Biotechnology*, *22*(7), 909–910. doi: 10.1038/nbt0704-909

Edgar, R. C. (2004, August). MUSCLE: A multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, *5*(1), 113. doi: 10.1186/1471-2105-5-113

Feng, D. F., & Doolittle, R. F. (1987). Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, *25*(4), 351–360. doi: 10.1007/BF02603120

Gecode Team. (2006). *Gecode: Generic constraint development environment.*

Gotoh, O. (1993, June). Optimal alignment between groups of sequences and its application to multiple sequence alignment. *Bioinformatics*, *9*(3), 361–370. doi: 10.1093/bioinformatics/9.3.361

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968, July). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107. doi: 10.1109/TSSC.1968.300136

Hicks, S., Wheeler, D., Plon, S., & Kimmel, M. (2011). Prediction of missense mutation functionality depends on both the algorithm and sequence alignment employed. *Human mutation*. doi: 10.1002/humu.21490

Istvan, A. (2022). *The Biostar Handbook: 2nd Edition.*

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., … Hassabis, D. (2021, August). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583–589. doi: 10.1038/s41586-021-03819-2

Katoh, K., Misawa, K., Kuma, K.-i., & Miyata, T. (2002, July). MAFFT: A novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, *30*(14), 3059–3066. doi: 10.1093/nar/gkf436

Kumar, P., Henikoff, S., & Ng, P. (2009). Predicting the effects of coding non-synonymous variants on protein function using the SIFT algorithm. *Nature Protocols*. doi: 10.1038/nprot.2009.86

Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., … The Wellcome Trust: (2001, February). Initial sequencing and analysis of the human genome. *Nature*, *409*(6822), 860–921. doi: 10.1038/35057062

Lilliefors, H. W. (1967). On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown. *Journal of the American Statistical Association*, *62*(318), 399–402. doi: 10.2307/2283970

Lipman, D. J., Altschul, S. F., & Kececioglu, J. D. (1989, June). A tool for multiple sequence alignment. *Proceedings of the National Academy of Sciences of the United States of America*, *86*(12), 4412–4415.

Liu, K., Raghavan, S., Nelesen, S., Linder, C. R., & Warnow, T. (2009, June). Rapid and accurate large-scale coestimation of sequence alignments and phylogenetic trees. *Science (New York, N.Y.)*, *324*(5934), 1561–1564. doi: 10.1126/science.1171243

Liu, K., Warnow, T. J., Holder, M. T., Nelesen, S. M., Yu, J., Stamatakis, A. P., & Linder, C. R. (2012, January). SATe-II: Very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees. *Systematic Biology*, *61*(1), 90–106. doi: 10.1093/sysbio/syr095

Mirarab, S., & Warnow, T. (2011, December). FASTSP: Linear time calculation of alignment accuracy. *Bioinformatics*, *27*(23), 3250–3258. doi: 10.1093/bioinformatics/btr553

Mount, D. (2001). Bioinformatics: Sequence and Genome Analysis.. doi: 10.5860/choice.39-0923

Needleman, S. B., & Wunsch, C. D. (1970, March). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453. doi: 10.1016/0022-2836(70)90057-4

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). MiniZinc: Towards a Standard CP Modelling Language. In C. Bessière (Ed.), *Principles and Practice of Constraint Programming – CP 2007* (pp. 529–543). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-540-74970-7_38

Notredame, C. (2002). Recent progress in multiple sequence alignment: A survey. *Pharmacogenomics*. doi: 10.1517/14622416.3.1.131

Notredame, C., Higgins, D. G., & Heringa, J. (2000, September). T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, *302*(1), 205–217. doi: 10.1006/jmbi.2000.4042

Pace, N., Sapp, J., & Goldenfeld, N. (2012). Phylogeny and beyond: Scientific, historical, and conceptual significance of the first tree of life. *Proceedings of the National Academy of Sciences*. doi: 10.1073/pnas.1109716109

Pei, J. (2008, June). Multiple protein sequence alignment. *Current Opinion in Structural Biology*, *18*(3), 382–386. doi: 10.1016/j.sbi.2008.03.007

Rao, R., Liu, J., Verkuil, R., Meier, J., Canny, J., Abbeel, P., … Rives, A. (2021). MSA Transformer. *bioRxiv*. doi: 10.1101/2021.02.12.430858

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of constraint programming* (Vol. 2). Elsevier.

Sankoff, D., & Cedergren, R. J. (1983). Simultaneous comparison of three or more sequences related by a tree. *Time warps, string edits, and macromolecules : the theory and practice of sequence comparison / edited* *by David Sankoff and Joseph B. Krustal.*

Schneider, V. A., Graves-Lindsay, T., Howe, K., Bouk, N., Chen, H.-C., Kitts, P. A., ... Church, D. M. (2017, May). Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *Genome Research*, *27*(5), 849–864. doi: 10.1101/gr.213611.116

Senior, A., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., ... Hassabis, D. (2020). Improved protein structure prediction using potentials from deep learning. *Nature*. doi: 10.1038/s41586-019-1923-7

Sievers, F., Dineen, D., Wilm, A., & Higgins, D. G. (2013, April). Making automated multiple alignments of very large numbers of protein sequences. *Bioinformatics*, *29*(8), 989–995. doi: 10.1093/bioinformatics/btt093

Smirnov, V. (2021, October). Recursive MAGUS: Scalable and accurate multiple sequence alignment. *PLOS Computational Biology*, *17*(10), e1008950. doi: 10.1371/journal.pcbi.1008950

Smirnov, V., & Warnow, T. (2021, June). MAGUS: Multiple sequence Alignment using Graph clUStering. *Bioinformatics*, *37*(12), 1666–1672. doi: 10.1093/bioinformatics/btaa992

Stackebrandt, E., & Woese, C. (1984). The phylogeny of prokaryotes. *Microbiological sciences*. doi: 10.1126/SCIENCE.6771870

Tafer, H., & Hofacker, I. (2008). RNAplex: A fast tool for RNA-RNA interaction search. *Bioinform.*. doi: 10.1093/bioinformatics/btn193

Thompson, J. D., Higgins, D. G., & Gibson, T. J. (1994, November). 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.

Tunyasuvunakool, K., Adler, J., Wu, Z., Green, T., Zielinski, M., Zídek, A., ... Hassabis, D. (2021). Highly accurate protein structure prediction for the human proteome. *Nature*. doi: 10.1038/s41586-021-03828-1

Wang, L., & Jiang, T. (1994). On the complexity of multiple sequence alignment. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, *1*(4), 337–348. doi: 10.1089/cmb.1994.1.337

Wang, S., Sun, S., Li, Z., Zhang, R., & Xu, J. (2017). Accurate De Novo Prediction of Protein Contact Map by Ultra-Deep Learning Model. *PLoS Comput. Biol.*. doi: 10.1371/journal.pcbi.1005324

Wilcoxon, F. (1945). Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, *1*(6), 80–83. doi: 10.2307/3001968

Woese, C. (2000). Interpreting the universal phylogenetic tree. *Proceedings of the National Academy of Sciences of the United States of America*. doi: 10.1073/PNAS.97.15.8392

Woese, C., Kandler, O., & Wheelis, M. (1990). Towards a natural system of organisms: Proposal for the domains Archaea, Bacteria, and Eucarya. *Proceedings of the National Academy of Sciences of the United States of America*. doi: 10.1073/PNAS.87.12.4576