JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Christian Samwald

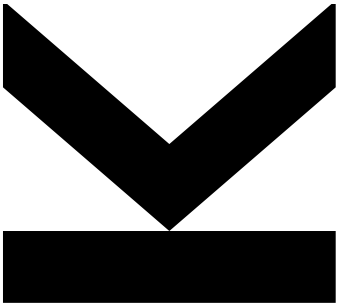
Submitted at
Institute of Machine Learning

Supervisor
Univ.-Prof. Dr. Sepp Hochreiter

Co-Supervisor
Dr. José Arjona Medina

March 2021

Effects of hyperparameters in multiple sequence alignment for Align-RUDDER using Clustal Ω



Bachelors Thesis
to obtain the academic degree of
Bachelor of Science
in the Bachelors Program
Bioinformatics

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Bibliographical Detail

Samwald, C., 2021: Effects of hyperparameters in multiple sequence alignment for Align-RUDDER using Clustal Ω. Bachelor Thesis, in English. 40 p., Institute for Machine Learning Johannes Kepler University, Linz, Austria

Annotation

Delayed rewards are detrimental to the learning of reinforcement learning agents. One approach to this problem is the usage of return decomposition and reward redistribution. It was realised in the Align-RUDDER algorithm of Patil *et al.* [14]. Their solution employed the multiple sequence alignment algorithm Clustal W. I integrated the sequence alignment Tool Clustal Ω, Clustal W's successor, into Align RUDDER to increase efficiency. During the testing process, the usage of Clustal Ω's EPA function and the effects of different sample sizes played a central role. The data set that was used came from the MineRL data set [6].

Declaration

I hereby declare that I have worked on my bachelor's thesis independently and used only the sources listed in the bibliography.

I hereby declare that, in accordance with Article 47b of Act No. 111/1998 in the valid wording, I agree with the publication of my bachelor thesis, in full to be kept in the Faculty of Science archive, in electronic form in a publicly accessible part of the IS STAG database operated by the University of South Bohemia in České Budějovice accessible through its web pages. Further, I agree to the electronic publication of the comments of my supervisor and thesis opponents and the record of the proceedings and results of the thesis defence in accordance with aforementioned Act No. 111/1998. I also agree to the comparison of the text of my thesis with the Theses.cz thesis database operated by the National Registry of University Theses and a plagiarism detection system.

Place, Date

Christian Samwald

Contents

Abstract

Sequence alignment has a wide range of applications in the field of Bioinformatics, most prominently for comparing two or more biological sequences. However, a more general approach to this method allows us to use it within Reinforcement Learning frameworks. In our paper, we examine the effects of different sample sizes and varying amounts of iterations on the alignment quality of multiple sequence alignments (MSAs). This is done by using the External profile alignment (EPA) approach of Clustal Ω. The novelty of this work is that the alignments are compared using the absolute number of matches, mismatches, gaps, and the gap length. The reason is that the test data is of non-biological origin and is used for Machine Learning (ML). It was found that adjusting the number of sequences did not change the alignment quality. However, splitting the data set into smaller subsets, depending on the sequence length, had a beneficial effect on the mismatches and gaps, while the number of matches decreased.

1 Introduction

One of the central areas in Machine Learing (ML) is Reinforcement Learning (RL). It involves an agent who learns how to achieve a certain target state by interacting with its environment. It is an independent, widely studied field with a multitude of applications, ranging from game playing over robotics to self-driving cars [21, 10].

Sparse and delayed rewards are one particular challenge RL is facing [21, 15]. To overcome it, Arjona-Medina *et al.* used return decomposition and reward redistribution processes. To properly utilise these methods, they needed to determine which actions of the agent contributed to the reward it got and to what extent they did so [1]. Key actions in a sequence of steps can be identified with sequence alignment tools used in Bioinformatics, as has been demonstrated previously [14].

However, the selection of a proper alignment tool has received little attention. In their study, Patil *et al.* used ClustalW to align the action sequences. We propose to use Clustal Ω because it is the successor of ClustalW and thus likely to be easy to integrate into Patil's Align-RUDDER algorithm. Sievers *et al.* [19] showed that Clustal Ω outperformed ClustalW by comparing many different alignment algorithms with Benchmark tests like BAliBASE or Prefab. Especially when large data sets, which contain between 1000 and 50 000 sequences or more are used, Clustal Ω performs exceptionally well [19]. This suggests that Clustal Ω could perform very well in combination with ML algorithms, as they tend to operate on very large data sets. Additionally, Clustal Ω can perform an External profile alignment (EPA), where an alignment is used as input to the algorithm to improve the resulting alignment.

This thesis investigates the suitable set of parameters for the alignment tool Clustal Ω because the choice of the proper parameters is equally important as using the appropriate alignment tool. For example, researchers found out that the EPA approach can have a detrimental effect on the alignment quality if the data set contains less than 1000 sequences [19]. However, only 2 consecutive iterations were

taken into account by Sievers. Also, no multiple sequence alignments (MSAs) with less than 100 sequences have been included in their study. In this work we test EPA under these conditions, because only a few of the many parameters offered by Clustal Ω are meaningful when non-biological sequences are involved.

2 Related work

This work uses concepts that come from two separate fields. Sequence Alignment is a technique used in the field of Bioinformatics. Return decompostition and reward redistribution are methods developed for RL purposes. In this section, we will provide background information for each of the two fields and their methods.

2.1 Bioinformatics

Bioinformatics emerged out of a combination of biology, genetics, mathematics, statistics, and computer science. It has an even wider range of applications including, but not limited to, collecting data from experiments, processing it, comparing them to the contents of databases, managing said databases, perform statistical analysis on the data, simulating experiments, and visualizing the obtained information. This is one of the reasons why an exact definition for the term bioinformatics is hard to find. Lacroix *et al.* describe it as:

"Bioinformatics is the design and develop- ment of computer-based technology that supports life science."[11]

We want to focus on explaining sequence alignment in general, MSA, and their biological basis as they represent the part of bioinformatics that is the most relevant for our experiments. Explaining all aspects of this field would be far beyond the scope of this paper.

2.1.1 Biological basics

In biology all information is stored in the DNA, which is first transcribed into RNA and then translated into proteins to express said information. Proteins, DNA, and RNA are all macromolecules and the information is encoded in the order of their building blocks. The building blocks of DNA are the 4 Nitrogen bases Adenine, Guanine, Cytosine, and Thymine (short: A, C, G, and T), which are attached to

a sugar and phosphate backbone for stability. RNA uses the same building blocks, except Thymine, which is replaced by Uracil (short: A, G, C, and U) and usage of a different sugar in its backbone. For both DNA and RNA one unit comprised of one sugar, one phosphate, and one base is called a nucleotide.

Proteins are comprised of 20 different amino acids, all with a similar chemical structure. All amino acids contain a carboxyl group ($COOH$) and an amine group (NH_2), which are both bound to the central C_α carbon atom. In addition, an organic substituent (R), also called side chain is also present. By forming peptide bonds, amino acids can be assembled into so-called polypeptide chains, more commonly known as proteins. This process is illustrated in Figure 1.

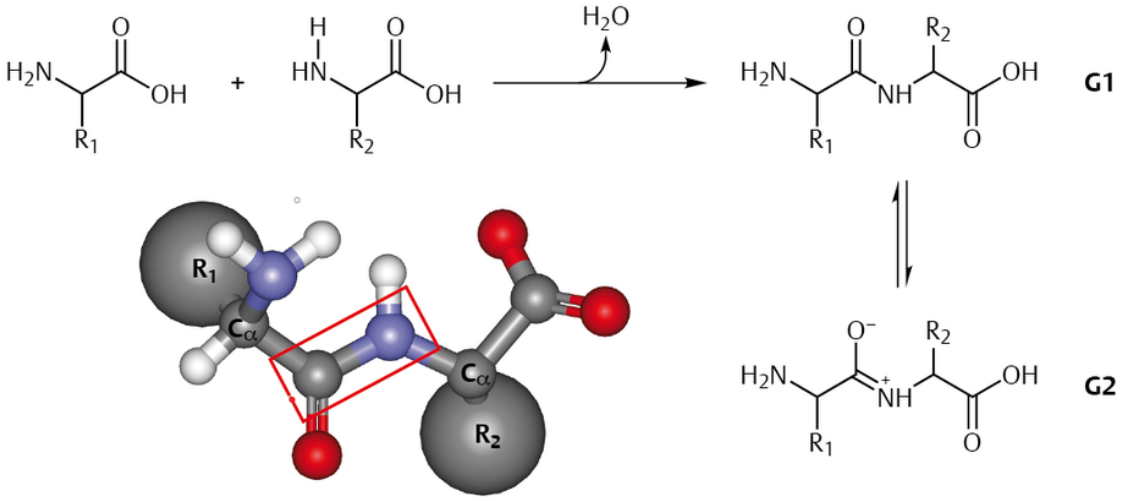


Figure 1: Formation of a peptide bond[9]

The amino acids types and their order within the protein are of particular importance. The reason for this is that the amino acids' side chains can interact with each other and thus influence the 3-Dimensional structure of the protein. Proteins are an essential part of every living organism, they can be fibrous proteins like Keratin, which are important for the structure of body tissues like hair or fingernails. They can be globular proteins like Hemoglobin that transport oxygen and carbon dioxide, or enzymes that for example catalyze digestion. The function of a protein strongly depends on its three-dimensional structure and thus also on its amino acid sequence. Which in turn depends on the sequences of the DNA and

the RNA, i.e. any change in these molecules, like insertions or deletion of bases, or mutations where a base is replaced by a different one can have effects on the amino acid composition and cause some proteins to malfunction. This is why it is imperative to obtain information on DNA, RNA and proteins. [2].

2.1.2 Sequence alignment

Sequence alignment is used to compare different biological sequences to each other. There are many fields of applications for sequence alignment they range from paternity testing to constructing phylogenetic trees. For example, the protein sequences of healthy and sick individuals can be compared to find out whether there are any patterns unique to sick individuals or patterns unique to healthy individuals. This data can help us to learn more about the disease. To build a sequence alignment each sequence is put into a separate line. Then they are arranged in a way such that the amino acids in the columns create the maximum amount of matches, as depicted in Figure 2. A match occurs when two identical amino acids are in the same position. If the amino acids are different it is called a mismatch. It is also possible to introduce gaps in one of the sequences. They are usually denoted by a hyphen. Gaps shift the rest of the sequence by one position to the right, while the other sequence stays in the same place. Gaps are commonly used when shifting one of the sequences would result in multiple matches further downstream in the sequence, they indicate that an insertion or deletion event has occurred. [23, 13, 22]

Depending on how the sequences are positioned relative to each other the number of matches, mismatches, and gaps in the alignment may vary. Thus, the alignments are rated by a score to describe their quality. As similarities are desired, matches usually receive a reward, which results in a higher score, while gaps and mismatches receive penalties, leading to lower scores. The decision when to introduce a gap and when to accept mismatches is made by so-called sequence alignment algorithms. Over the years many different alignment algorithms have been proposed [3, 4, 13, 19, 23]; we will explain the most important ones in our

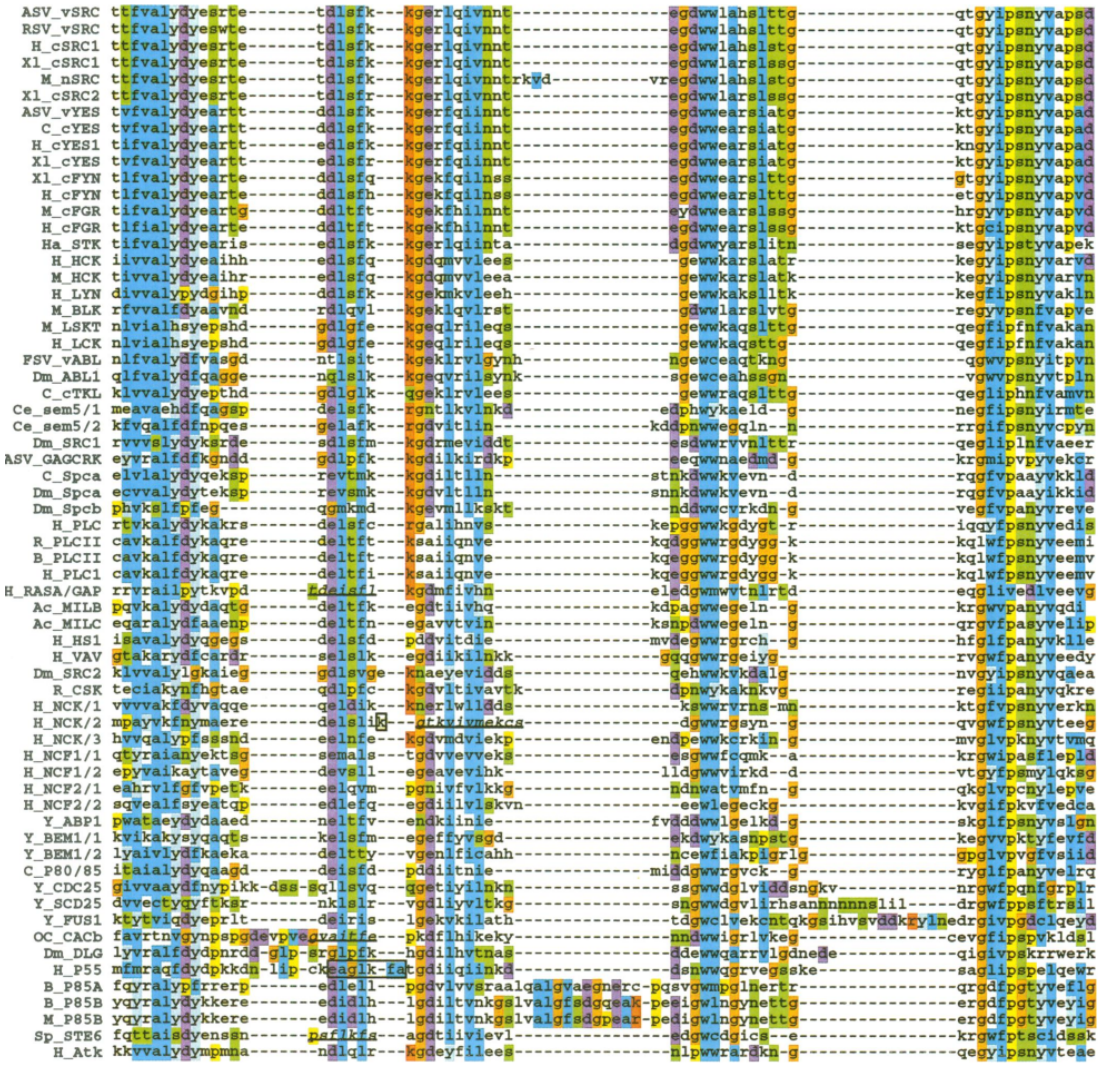


Figure 2: Multiple sequence alignment [22]

work. In the examples, we will focus on proteins. However, sequence alignment can also be done with DNA and RNA.

Dotplot

A common tool used in many sequence alignment algorithms is the dotplot. Dotplots are a means to visualize the similarity of sequences. They were developed by A.J.Gibbs and G.A.Mc Intyre to compare biological sequences more efficiently [12]. A dotplot is drawn by placing one sequence horizontally on the upmost row of a table and the other sequence vertically in the leftmost column of the table as depicted in Figure 3. Whenever two letters are the same, a dot is placed in the cell that represents the intersection of the matching letters. If many dots are diagonally adjacent this means that there are multiple consecutive matches. This in turn indicates a region of high similarity. To aid the portrayal of these similarities a line between diagonal dots is drawn in the dotplot, these lines are called diagonals [12].

The dotplot is specifically useful for illustrating the occurrence of insertions or deletions as the gap is represented by a horizontal or vertical shift of the diagonals. The length of the gaps equals the number of cells that the diagonal is moved. Due to a reduction of the size of the cells and of the dots, dotplots of very long sequences can be drawn on a single sheet of paper, as depicted in Figure 4a. To aid the depiction of similar regions, matches which are only one single amino acid long can be filtered out. This is achieved by comparing not only one but a window of adjacent amino acids at once. Adjusting the window size can be particularly useful when combined with the so-called stringency parameter. The stringency determines how many letters in the window need to match the letters of the other sequence for a dot to be plotted. For example Figure 4b shows a dotplot with window size 11 and stringency 7. This means that 7 of the 11 following amino acids have to match for a dot to be plotted [12].

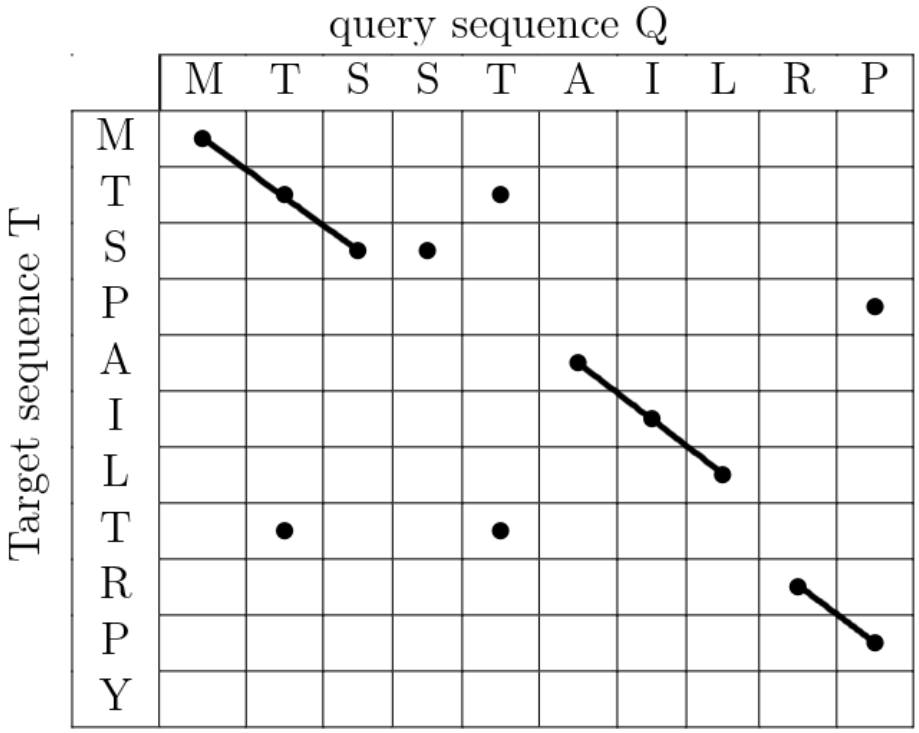
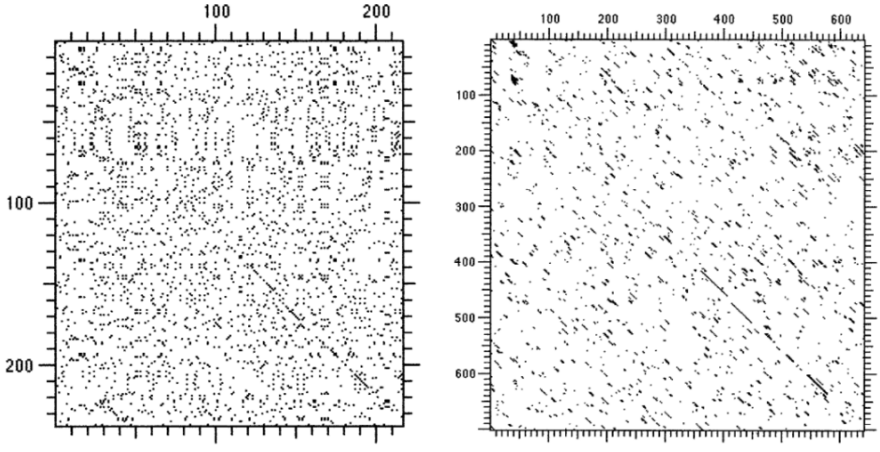


Figure 3: Dotplot



(a) window size=1 (b) window size=11, stringency=7

Figure 4: Dotplots with different window sizes [12]

Pairwise sequence alignment

Pairwise alignments contain only 2 sequences and are fairly simple to create. Due to the sophisticated modern algorithms they can be computed efficiently. The algorithms we will describe subsequently have been developed in the late 1960s and the 1980s, but their concepts are still employed as a basis for many modern algorithms like BLAST, FASTA, or Clustal Ω [4, 19].

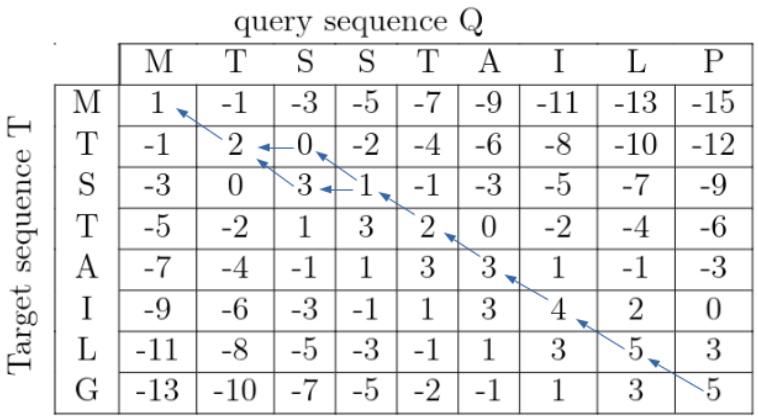
Needleman-Wunsch algorithm

The Needleman-Wunsch algorithm was published by Saul B. Needleman and Christian D. Wunsch in July 1969 [13]. Over the years there have been some adaptions. However, the key concept remains unchanged. We will explain a version of the algorithm that illustrates mentioned concepts in an understandable way [12]. The Needleman Wunsch Algorithm aligns a query sequence Q to a target sequence T. The visualization of the algorithm is very similar to the dotplot. However, the dotplot tends to be inefficient in regions of the sequence that have low similarity. This is compensated by using numbers instead of dots in the Needleman-Wunsch algorithm. It compares just two amino acids at a time. Amino acid i from Q with amino acid j from T. In case of a match, the score 1 is given. In case of a mismatch the score is 0, and when a gap is introduced the value of a predefined gap penalty g is used. The algorithm runs through every cell of the table and inserts the highest possible score $S(i, j)$. The starting point is the upper left corner. After that row by row is filled. Matches and mismatches are represented as one diagonal step to the lower right. Gaps are depicted as one step to the right or one step down, depending on the sequence in which the gap is introduced. The score $S(i, j)$ in each of the cells is calculated as follows:

$$S(i,j) = max\Big(S(i-1,j-1) + s(i,j), S(i-1,j) - g, S(i,j-1) - g\Big)$$

where $s(i, j)$ is either 1 or 0 depending on whether the amino acids i and j create a match or mismatch. The highest score that is achievable should be placed in the

lower right corner. By tracking it back to the starting point, the optimal alignment can be obtained [12, 13], as displayed in Figure 5.



		M	T	S	S	T	A	I	L	P
	M	1	-1	-3	-5	-7	-9	-11	-13	-15
	T	-1	2	0	-2	-4	-6	-8	-10	-12
	S	-3	0	3	1	-1	-3	-5	-7	-9
	T	-5	-2	1	3	2	0	-2	-4	-6
	A	-7	-4	-1	1	3	3	1	-1	-3
	I	-9	-6	-3	-1	1	3	4	2	0
	L	-11	-8	-5	-3	-1	1	3	5	3
	G	-13	-10	-7	-5	-2	-1	1	3	5

Figure 5: Alignment with Needleman-Wunsch Algorithm with gap penalty $g = 2$

In the example in Figure 5 there are 2 possible alignments, both of which lead to an optimal outcome, as both produce the same score. These two equivalent alignments are:

$$\begin{array}{cccccccccc} M & T & S & S & T & A & I & L & P \\ M & T & S & - & T & A & I & L & G \end{array}$$

$$\begin{array}{cccccccccc} M & T & S & S & T & A & I & L & P \\ M & T & - & S & T & A & I & L & G \end{array}$$

This example illustrates, that every possible alignment of the two sequences can be graphically displayed as a path through the score matrix.

k-tuple algorithm

The k-tuple algorithm was developed by W.J. Wilbur and David J. Lipman in 1983. It was later expanded and became the FASTA algorithm, which is still in use today [5]. The algorithm uses both, the Needleman-Wunsch algorithm and the

dotplot and can perform multiple pairwise alignments. It quickly matches multiple target sequences against one query sequence. This is done by reducing the search space by focusing on a predefined window space of the size w. It can also assign each of the target sequences a similarity score to find the sequence with the highest similarity to the query sequence [23].

First, the k-tuple algorithm uses the dotplot to determine regions of high similarity, the main diagonals. However, it does not compare single letters. Instead, multiple adjacent letters, so-called words, also referred to as k-tuples or k-mers are compared between the query sequence Q and the target sequence T. For proteins the typical k-tuple size is 2, for nucleotides it is 4-6 [22]. If a k-mer of Q matches one from T, a dot is plotted. After all dots are plotted, diagonal dots are connected by a line. In the k-tuple algorithm the dotplot is used to identify in which diagonals the number of matching tuples is multiple standard deviations above the mean. Those diagonals are called significant diagonals. The window space is then defined as all diagonals with a maximum distance of w cells to the next significant diagonal.

After defining the window space, a Needleman-Wunsch alignment is performed. The scoring values for matches and mismatches and the gap penalty may change depending on the application, but the concept remains the same. The alignment can be computed faster in this case, as not all possible alignments need to be considered. Only the k-tuples that occur within the window space have to be scored [23]. When the alignment algorithm is applied on multiple sequences from a database, the output is a list of raw scores. Each score corresponds to one of the sequences. These raw scores are dependent on the length of the sequences Q and T; however, normalization of the score solves this problem. Normalizing means subtracting the mean of the distribution from every raw score and then dividing by its standard deviation.

Scoring matrices

For amino acids not all mismatches are of equal significance. While some amino acids have similar physical and chemical properties and can be interchanged, others are inherently different. For this reason, many modern alignment algorithms do not give a global penalty of 0 to every mismatch and a reward of 1 to every match. Instead they use so-called scoring matrices that determine the score of aligning amino acid i to amino acid j. The scores in the scoring matrix also reflect the likelihood that one of the amino acids is replaced by the other one through evolution. Thus, depending on the ancestry of the sequences themselves and how closely they are related, different scoring matrices can be chosen. PAM (Point Accepted Mutations) and BLOSUM (BLOCKS Substitution Matrices) are two types of scoring matrices that are widely used [4].

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	B	Z
C	12																					
S	0	2																				
T	-2	1	3																			
P	-3	1	0	6																		
A	-2	1	1	1	2																	
G	-3	1	0	-1	1	5																
N	-4	1	0	-1	0	0	2															
D	-5	0	0	-1	1	2	2	4														
E	-5	0	0	-1	0	0	1	3	4													
Q	-5	-1	-1	0	0	-1	1	2	2	4												
H	-3	-1	-1	0	-1	-2	2	1	1	3	6											
R	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6										
K	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5									
M	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6								
I	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5							
L	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2	6						
V	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4					
F	-4	-3	-3	-5	-4	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9				
Y	0	-3	-3	-5	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	7	10			
W	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17		
B	-4	0	0	-1	0	0	2	3	2	1	1	-1	1	-2	-2	-3	-2	-5	-3	-5	2	
Z	-5	0	-1	0	0	-1	1	3	3	3	2	0	0	-2	-2	-3	-2	-5	-4	-6	2	3

Figure 6: The PAM250 matrix [16]

Multiple sequence alignment

MSA is an extension to pairwise sequence alignment, where more than 2 sequences are aligned. Pairwise alignment methods can be extended to include a third sequence, but building an alignment with more sequences in the same way becomes computationally infeasible. So instead of aligning all sequences at once, the problem is broken down into smaller parts. Pairwise sequence alignments are built and aligned to each other successively. This is called a progressive alignment [8].

Progressive alignment algorithms

A progressive alignment requires a so-called distance matrix. It contains the pairwise distances of each pair of sequences. A guide tree, which characterizes the similarity between and within groups of sequences, is then constructed from these distances. The information is crucial to determine in which order the sequences are to be aligned [22, 8]. There are multiple progressive alignment methods. We will describe the algorithm developed by Blackshields *et al.* as it is used by Clustal Ω [3, 19].

Sequence embedding for fast construction of guide trees (mBed)

In the last decades, both the speed of sequencers and the amount of data in various online databases have been growing exponentially. Thus, when processing today's data sets the capacities of standard algorithms are exceeded. For example, 100 000 sequences would lead to about 5 billion distance computations for the distance matrix. With standard methods like UPGMA, this would require around 12 days of computation [3]. For this reason, the mBed Algorithm uses two kinds of distances. It uses the k-tuple algorithm, described in the k-tuple algorithm paragraph, to create a similarity score for each sequence pair. From this score the distances can be calculated. The distances used in the algorithm are the Euclidean distances of embedded sequences represented by vectors. The mBed algorithm assigns each sequence to a vector in a t-dimensional embedding space. The relationships between the sequences are then represented by the way that these vectors are positioned in the embedding space.

To compute the vectors initially, a set of sequences or "seeds" is selected from the whole data set to serve as a reference of the data. The seeds are supposed to model the key features of the data, like clusters within the data set or its outliers. The seeds are generated by sampling $t = (log_2 N)^2$ sequences from the original data set X with a constant stride. We call the set of seeds R. If the distance between any two seed sequences is below a certain threshold, the shorter one is considered

redundant and discarded. After the selection of all sequences in R, the remaining sequences are assigned a t-dimensional vector, by computing their distance to each of the t seeds in R. Each distance is a value in the vector, i.e. $F(s)$ is the vector corresponding to sequence s from X and $d(s, R_1)$ is the k-tuple distance between s and the first sequence in R. Then $F(s) = [d(s, R_1), d(s, R_2), ...d(s, R_t)]$.

After the vectors are computed, the distances between the vectors resemble the distances between the real sequences. The distances between the vectors are calculated by using the Euclidean distances between the vectors. This can be determined orders of magnitude faster than the computation of k-tuple distances. After the matrix of embedded distances is created, the UPGMA clustering algorithm is used to create a guide tree with the embedded distances [3].

As the UPGMA algorithm is used, the pairwise sequence alignments are built in the same order as the guide tree. First, the two most similar sequences are aligned and their consensus sequence is then treated as one sequence by the algorithm. Thereafter, distances are recomputed as described in the Unweighted pair group method with arithmetic mean paragraph below. In each successive step either a new consensus sequence is built from the pair of most similar sequences or a sequence is aligned to one of the consensus sequences to form a new consensus sequence [8].

Unweighted pair group method with arithmetic mean

The Unweighted pair group method with arithmetic mean (UPGMA) is a clustering algorithm. It generates guide trees based on the assumption that all branches of the tree evolve at the same speed. UPGMA starts by combining the two most closely related sequences into a cluster and then treats this cluster the same as a single sequence. The most closely related sequences have the smallest distance to each other. The distance between the cluster and another sequence A is simply the arithmetic mean of the distances between each sequence in the cluster and sequence A. After each step, all distances that are affected by the new cluster are recomputed. Thus, a fast computation of distances is important. Depending

on the smallest distance, in every following step the algorithm either joins two individual sequences that are most closely related into a new cluster or joins an existing cluster with the most closely related cluster or sequence. This is repeated until there is only one cluster left that contains all sequences. Finally, a root of the generated guide tree is predicted [12].

To better illustrate the UPGMA algorithm and give the reader a visual of the guide tree, we want to explain the process with the example depicted in Figure 7. Table 1. In the figure the initial distance matrix, where the distances between the 5 biological sequences A, B, C, D, and E are stored, is displayed. The first step is to find the sequences with the least amount of distance between them, in this case, D and E. We join both sequences together in a new cluster, illustrated in Figure 7a. The distances d and e describe the evolutionary distance between the sequences D and E and their common node. The base assumption of the UPGMA algorithm is that after a common node both branches evolve at the same pace. Thus, the evolutionary distance between the common node and each of the branches is the same, half the distance between the individual sequences, i.e

$$d = e = \frac{8}{2} = 4$$

After clustering D and E together, this cluster is now treated as a new sequence DE. As a new sequence is created, the distance matrix must be recomputed. The distances between DE and the other sequences are simply the arithmetic mean of their distances to D and E i.e

$$\overline{AD} = 17, \overline{AE} = 15 \Rightarrow \overline{ADE} = \frac{15 + 17}{2} = 16$$

The distances between B and DE and C and DE are calculated in the same way. This creates matrix 2, in which 12 is the smallest number. Therefore, we will add C to the cluster DE as shown in Figure 7b. As explained at the beginning of the Unweighted pair group method with arithmetic mean paragraph, each of the

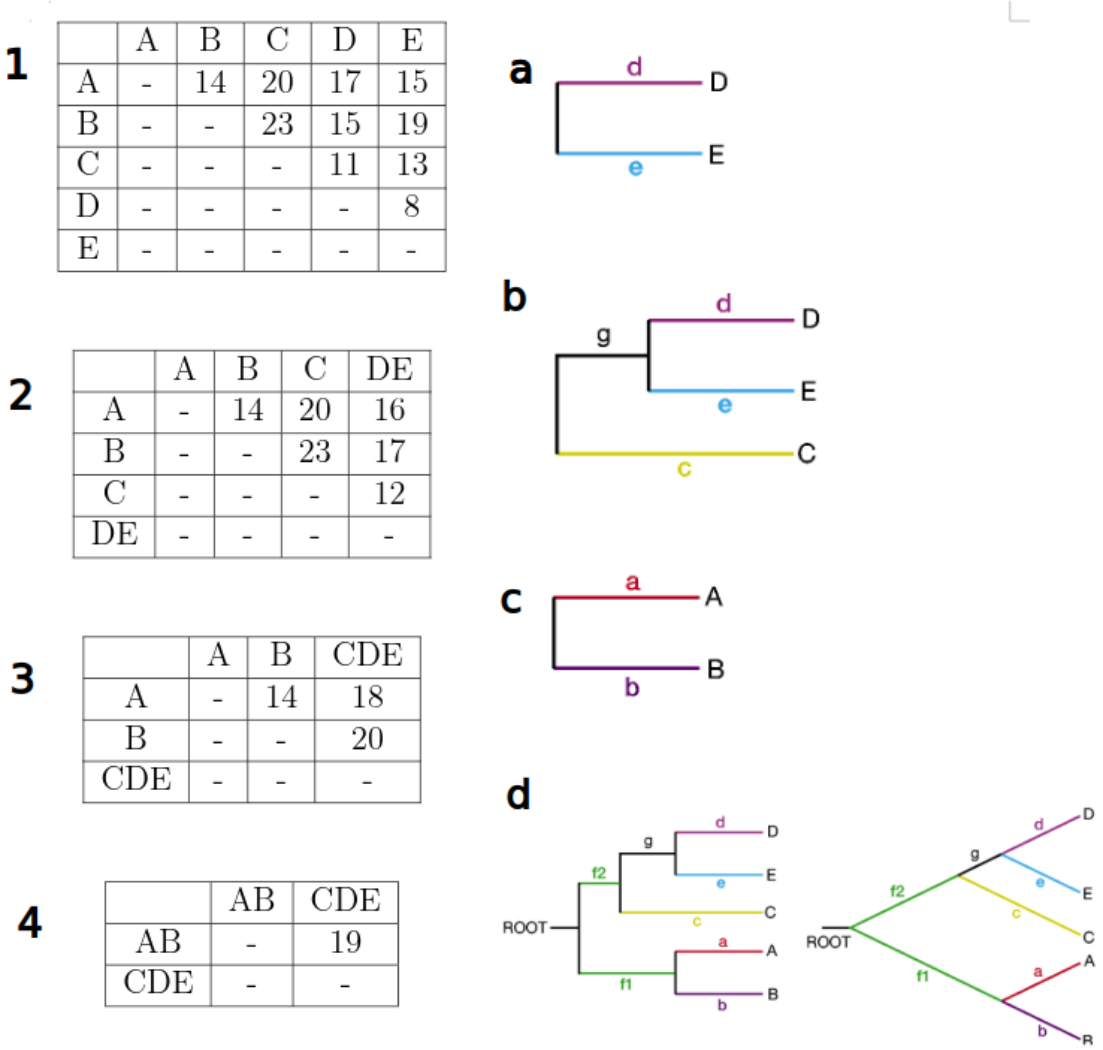
branches evolves at the same pace, thus:

$$c = g + d = g + e, c = \frac{12}{2} = 6 \Rightarrow g = c - d = 6 - 4 = 2$$

Matrix 3 is calculated in the same way as matrix 2, as the arithmetic means of the distances from A and B to C as well as the means of the distances from A and B to DE. As the distance between A and B is the smallest in the new matrix, the new cluster AB is created (Figure 7c). The distances a and b are equal to 7 and calculated as shown in the previous cycle. Finally, in Matrix 4, only 2 clusters are left, AB and CDE, which are joined together into one big cluster, depicted in Figure 7d. Finally, the distances $f1$ and $f2$ are:

$$f1 + a = f2 + g + d = \frac{19}{2} = 9.5 \Rightarrow f1 = 2.5, f2 = 3.5$$

There are two possibilities to display a guide tree, both versions of the depicted tree are equally valid. However, for consistency we decided to focus on one of them. In the portrayal of the tree that we chose, only the horizontal distances matter.



	A	B	C	D	E
A	-	14	20	17	15
B	-	-	23	15	19
C	-	-	-	11	13
D	-	-	-	-	8
E	-	-	-	-	-

	A	B	C	DE
A	-	14	20	16
B	-	-	23	17
C	-	-	-	12
DE	-	-	-	-

	A	B	CDE
A	-	14	18
B	-	-	20
CDE	-	-	-

	AB	CDE
AB	-	19
CDE	-	-

Figure 7: Construction of a guide tree with UPGMA [12]

2.2 Reinforcement learning

Reinforcement Learning (RL) is next to Supervised Learning and Unsupervised Learning one of the major paradigms in ML. However, RL algorithms are more relevant for this work. This is because Supervised and Unsupervised learning are generally more suited for Classification tasks. RL algorithms, on the other hand, aim to maximize the future expected return signal [21]. The goal of RL algorithms is to learn which sequence of steps is necessary to get to the desired goal state. This is achieved by trying to copy the learning process of nature. The learner receives feedback in form of rewards, like food, warmth, good feelings, etc. or penalties like pain, hunger, bad feelings, etc. Through this feedback the learner, who in the field of ML is called the agent, learns to take or avoid certain actions. For example, after being stung by a stinging-nettle for the first time, children learn to avoid these plants when playing outside [21].

The agent's way of learning is similar to the way children and animals learn. The agent is usually located in an environment, depending on the application it can be simulated or even be the real world. The environment consists of several states. Depending on the scenario there can be a finite or even infinite number of states. By performing certain actions (e.g. place an X to obtain the next tic-tac-toe table or move to change the room in which a robot vacuum is located), the agent must be able to change from one state to another. To notice that the states are different, it has to 'sense' the environment it is currently in. Depending on the action that was taken and the new resulting state, the agent then receives a reward signal. To maximize the obtained reward, the agent has to take actions which it has already tried and where it received a reward. However, if the agent always performs the same action that gives a reward, there might be an action with an even higher reward, which the agent never finds, because it has never tried it. One of the challenges in RL is to balance exploration and exploitation. Both, exploiting the obtained information of the environment and exploring new actions, which have never been taken before, are necessary in RL. If either exploration or exploitation are followed exclusively, the agent will usually not be able to accomplish the task and cannot reach the goal state [21].

To understand how the agent decides what action to take, we first need to look at the different components of RL. There are four main components in RL systems: its policy, the reward signal, a value function, and a model [21].

The policy maps certain states to actions that bring high rewards when in that state. If an action is discovered that leads to an even higher reward signal, the policy may be updated [21].

In every time step, the environment sends only the reward to the agent. This reward signal identifies the goal of the RL algorithm. It defines whether a state is desirable or not. The agent's task is to maximize the overall reward received over the whole run. However, the reward signal only shows the immediate, short term reward for the state [21].

The value function, on the other hand, shows the value of the state by taking future rewards into account, i.e. the value of a state resembles the total amount of rewards that can be accumulated from that state onward. For example, a state can have a low reward but a high value because following states give high rewards. By multiplying rewards from future states with a weighting factor that gets smaller the further the reward is in the future, values are computed from rewards of the successor states. Hence without rewards, there is no value. As values are more relevant, typically the action with the highest value is selected when deciding on which action to take next.

Rewards are given by the environment. Values, on the other hand need to be calculated. Thus, they are much harder to determine as they must constantly be estimated and re-estimated to incorporate new knowledge obtained by exploration. [21].

An RL algorithms model is used for planning and allows to deduce how the environment will behave. In RL we distinguish model-free methods from model-based methods. Model-free methods like try and error agents do not consider the big picture and have no access to things like reward and state transition probabili-

ties. On the other hand, there are model-based methods, which either start with a model of the environment or try to learn it from samples. The model is then used for planning, considering possible future states. However, modern algorithms can incorporate model free and model-based methods. They are not mutually exclusive, but more like a spectrum from simple try and error principle to sophisticated planning [21].

2.2.1 Return decomposition and reward redistribution

One challenge that RL is facing are sparse and delayed rewards [15, 14, 1]. Three particular issues need to be dealt with when facing this problem. First, there are cases where the agent does not even find a reward through exploration. The reason for this is that the rewards are very sparse or received later and thus mapped to another action. Hence, creating and exploiting a policy is hard. Secondly, even if a delayed reward is found, typical RL agents do not save the entire episode to map the reward to the right action as memory is limited. The third issue when facing this problem is that the algorithm often cannot figure out which of the many steps it took in that episode were beneficial and which were not [1, 14]. We will look at an approach to solve this problem, called return decomposition and reward redistribution, and two applications that utilize it.

Return decomposition for delayed rewards

One application of return decomposition and reward redistribution is Return Decomposition for Delayed Rewards (RUDDER) proposed by Arjona-Medina *et al.* [1]. It identifies which steps are essential and have to be taken to obtain a reward. As the reward is given after each episode, it is redistributed towards the relevant steps to speed up the learning process. For this, a lessons replay buffer and a Long-Short-Term Memory (LSTM) network are used.
To ensure that the agent experiences delayed rewards Arjona-Medina *et al.* use a safe exploration strategy, where state-actions with low value are avoided. Whenever an unexpected reward is encountered, that episode is saved in the lesson replay buffer. As the LSTM predicts rewards based on an input sequence, unexpected

rewards are defined as rewards that differ significantly from the reward prediction of the LSTM. By performing contribution analysis, the LSTM can identify relevant actions that led to a reward, even if they are in the distant past. It also computes the contribution of each of the individual actions to the outcome [1].

3 Methodology

To find the optimal parameters for the MSA algorithm Clustal Ω, we analyzed alignments of 117 sequences which originated from the MineRL data set [6], specifically the ObtainDaimond task, and then compared the performance of Align-RUDDER when using different strategies. As Align-RUDDER uses non-biological data [14] we first wanted to examine how the alignment quality changes for different alignment parameters. The effects of using the EPA approach [19], in combination with artificially created data are of particular interest. For this, we performed several alignments with a varying number of iterations. We also tried to assess the effect of sample size on alignment quality by dividing the data into subsets and compared the alignment quality to an alignment of the whole data set. To properly compare the alignment quality, we counted the matches, mismatches, and gaps of the alignments. The reason for this is that traditional scoring methods are well adapted for biological sequences but not to artificial ones. Using this scoring method we observed noticeable differences between the grouped and ungrouped samples as well as the samples and their respective reference values. Thus, we performed statistical analysis on these intermediate results to gain an understanding of advantages and disadvantages of the different parameter settings.

The different parameter settings were then incorporated into the Align-RUDDER [7] algorithm to test them in combination with reward redistribution, which is described in subsubsection 2.2.1. The eight rooms environment, which was also used by Patil *et al.* [14], was the evaluation. We tested our Align-RUDDER version with the adjusted MSA parameters with varying numbers of demonstrations and measured the number of episodes the agent needed to learn an 80% optimal return policy. Some runs produced errors due to incompatibilities in the python version; however, the number of erroneous runs was negligible.

3.1 Choice of Clustal Ω parameters

Our goal was to investigate the influences of iterations and small numbers of sequences on the alignment quality. We used artificial data as input and Clustal Ω as an alignment tool. Similar to other sequence alignment tools it has its own, unique set of parameters which can be set by using so-called flags when applying the algorithm to the sequences. We realized the execution of Clustal Ω by downloading the program from the Clustal Ω Webpage [18] and then calling the 'clustalo' command via Ubuntu terminal. The 'clustalo' command has different flags available, the flags that were used for the analysis here, are the '–in', '–out', '–iter', and the '–force' flag.

The '–in' flag specifies the input file that provides the sequences for the MSA. To create an alignment, Clustal Ω produces a guide tree to decide the order in which the sequences are aligned. To obtain the guide tree, however, the program needs a distance matrix, which is comprised of pairwise distances that are calculated using the k-tuple measure developed by Wilbur and Lipman [23].

Using the '–iter' flag will create an initial alignment, which is only used as input for an external profile alignment (EPA). The EPA approach will likely improve the guide tree and subsequently the alignment [17]. As opposed to the standard input of unaligned sequences, when iteration is used, full alignment distances can be calculated from the input MSA. This is essential to improve the guide tree and in turn the alignment. The '–iter' flag is followed by an integer that specifies the number of iterations that should be performed. The process of using the produced alignment as input for the next one can be repeated multiple times in a row. After the alignment process, the alignment is saved in the location specified in the '–out' flag. If the specified file is not created yet, then Clustal Ω will just create a new file. If there already was a file with that particular name in that location, this command would not work, as Clustal Ω does not overwrite existing files unless the '–force' flag is used [17].

To test whether or not a small sample size affects the quality of our alignments,

we divided our data set into subsets. Then we aligned the subsets before aligning them to each other. Finally, we compared the alignment of subsets to an alignment in which all of our sequences were aligned at once. This means that the alignments for each number of iterations the alignment was performed twice. Once the iterations were applied to our whole data set and the second time to a sample separated into groups and then realigned. We used three groups and separated depending on the length of the sequences: Sequences with a length of 100 elements or less were in the first group. The second group contained the sequences with lengths between 100 and 500 elements. The longest sequences of over 500 elements formed the third group. We iterated over the three separated groups and thus created three MSAs. These three alignments were then aligned to each other, using the '–profile1' and the '–profile2' flag to combine them into one alignment to make them comparable. These flags prevent Clustal Ω from dealigning the MSAs submitted via the flags. Otherwise Clustal Ω would dealign all of the sequences in the profiles before realigning them.

To assess the effect of iterations on the alignment, we chose to perform a different number of iterations on the data. We chose to examine the alignment quality from 1 to 10 consecutive iterations each. To gain some information regarding whether or not a higher number of iteration affects the alignment, we also iterated the alignment 20, 100, and 200 times. However, as a high number of iterations requires a lot of computing power and needs more time than a smaller number of iterations these are our only data points in that range. They were chosen to indicate the possibility of a trend for a higher number of iterations.

The iteration-process was carried out twice. The first time it was carried out on the data set that was divided into smaller subsets, the 'grouped' data set, then on the whole data set at once, the 'ungrouped' data set.

To have a control group, we needed reference alignments that do not use iterations. Thus, we created two MSAs without EPAs for each, the grouped and the ungrouped data set as reference values.

For our artificially created sequences assessing the quality of their alignments with conventional scoring algorithms would be pointless. The reason is that these algorithms are specialized on information regarding the genome, transcriptome, or proteome. Thus, they assign more weight to certain mismatches than others. There is no data stating that one mismatch is preferable compared to a different mismatch in our data set. Hence, counting the mismatches is one way to deal with this issue.

A similar problem arises when we look at gaps. We do not have metadata about the sequences, which means there is no indicator whether there should be a gap penalty. Thus, it is unknown whether many short gaps or few longer ones are preferred. It also remained unclear if gaps should be introduced to avoid a mismatch or, vice versa, accept more mismatches in the alignment to avoid gaps.

For this reason, we counted the matches, mismatches, the total length of all gaps in every sequence of the alignment, and the total number of gaps in every sequence of the alignment.

If there is a very short sequence aligned to the center of a long sequence, the parts before the beginning and after the ending of the short sequence, are displayed as gaps by Clustal Ω. However, they were not counted as such in our analysis.

To illustrate how the matches, mismatches, and gaps were counted, an example is provided below. The alignment below has 10 matches, 3 mismatches, and 2 gaps with a total gap length of 3.

A	T	N	N	V	R	E	P	A	A	S	S	S	S	K	W	W	E	E	T	T	D
	⋮	⋮			⋮	⋮	⋮		⋮			⋮	⋮		⋮	⋮					
$-$	T	N	G	$-$	R	E	P	L	A	$-$	$-$	S	S	D	W	W	$-$	$-$	$-$	$-$	$-$

3.2 Statistical analysis

We evaluated the number of matches, mismatches, gaps, and the gap length for each alignment of both the grouped and ungrouped data set. The results are displayed in Figure 10. An objective was to test whether or not our results are consistent with the study of Sievers *et al.* [19], in which it is pointed out that the EPA approach yields a worse result for data sets smaller than 1000 sequences. As our data set contains 117 sequences the results of both, the grouped and ungrouped sample, are expected to be worse than their respective reference sequences. To choose a proper statistical method to test the hypothesis that our samples are different, they first must be tested for Gaussian distribution.

To test for normal distribution, a Kolmogorov-Smirnov test with Lilliefors correction has been applied to the data. A positive result shows with a certainty of 95% that the sample is normally distributed. Thus, a confidence interval (CI) can be used to compare both our samples with their respective reference values.

We also investigated whether dividing the data into groups affected the alignment quality, so we compared both samples with each other. To test the hypothesis we performed a paired t-test. This helped us to determine whether or not there is a difference between the means of the grouped and the ungrouped sample. The t-test requires a normal distribution, for which we already checked.

The Lilliefors-test, the CI, and the paired t-test were computed for all 4 parameters of the alignment: the matches, mismatches, the length of the gaps, and the number of gaps. In the statistical analysis multiple tests were used, yet no correction for multiple tests was applied. Hence, our results cannot be called statistically significant, but rather statistically noticeable.

3.3 Align-RUDDER

One problem with reward redistribution is that for very sophisticated tasks the agent may never encounter a high reward because it never tried the correct sequence of actions. However, as high rewards are essential for learning, RL algorithms need rewards to find a solution to the problem. Animals and humans often learn by watching and copying the behavioral patterns of their role models, like parents, teachers, etc. This means that one way to help the agent find high rewards is by giving them demonstrations of episodes with high rewards as input so it can learn from them. Creating these examples is exhausting for humans, while it is very complex and time-consuming for automated methods. This is why usually only very few demonstrations are available for the algorithm. However, RUDDER's LSTM network delivers an unsatisfactory performance when there are too few examples [14].

To overcome this problem, Patil *et al.* [14] created their algorithm Align-RUDDER, where the LSTM algorithm used in RUDDER [1] is replaced by a profile model obtained by MSA. RUDDER's reward redistribution is based on the difference between two consecutive LSTM model predictions, where increased predictions lead to an immediate reward. For Align-RUDDER, where the LSTM model is replaced by MSA, the reward redistribution is performed differently. To redistribute the rewards, first, the demonstrations that are given and episodes that are produced by the agent must be converted into a sequence of events. This can be done in five steps as shown in Figure 8:

1. **Defining events:**
   State-action pairs, clusters thereof, or other factors that represent state-action pairs like changes in the inventory or the level of the player are assigned to a certain event. To paraphrase, all state actions of an episode are mapped to an event. The amount of different possible events must be kept small, e.g. around 20 symbols to increase the difference between a random alignment and an alignment of relevant state actions.

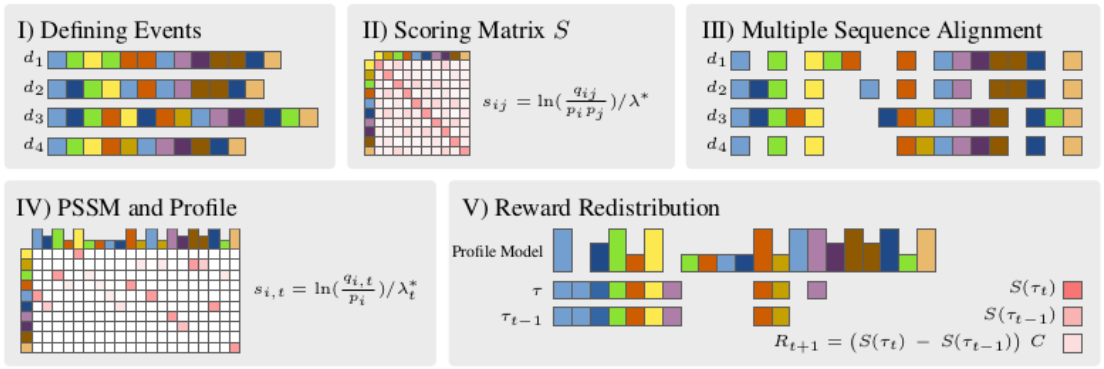


Figure 8: The five steps of reward redistribution in Align-RUDDER by Patil *et al.* [14]

2. **Alignment scoring system:**
   To estimate the alignment quality an alignment scoring system is needed. Most MSA algorithms use a scoring matrix. This matrix is a square matrix of n x n, where n is the number of possible events. It contains the scores q_{ij} for aligning event i with event j. It is unknown beforehand which events are relevant, what actions they correspond to, or in which order they should occur. However, we do know that relevant events should be aligned to themselves, thus if $i = j$ the score q_{ij} for that alignment should be high, while for $i \neq j$ the score should be low.

3. **Multiple sequence alignment:**
   A MSA is computed as described in the Multiple sequence alignment paragraph. Since demonstrations that have the same underlying strategy contain similar events, they are likely to be clustered together by the clustering algorithm. The MSA of demonstrations is required to create the profile model. Align-RUDDER uses ClustalW for this step.

4. **Position-specific scoring matrix:**
   A Position-Specific Scoring Matrix (PSSM) is used to align new sequences to the profile model. It stores the frequency $q_{i,t}$, with which an event i is aligned to a certain position t, relative to the probability p_i that i occurs in the new sequence. The frequency of each possible event is recorded in the

PSSM.

5. **Reward redistribution:**
   The reward redistribution is based on the profile model. A sequence of events is aligned to the profile and the score of the alignment is computed with the PSSM as a scoring matrix. The reward redistribution is then calculated using the alignment score [14].

3.4 Experimental Setup

To examine which parameter setting performs best in combination with reward distribution we incorporated different alignment strategies into the code of Align RUDDER. Four different Clustal Ω parameter settings were selected and we used them with 2, 5, 10 and 20 demonstrations, as Align-RUDDER's focus is on applications with a low number of demonstrations. An amount of 10 seeds was used for each of the 4 parameter settings and each of the numbers of demonstrations. We also ran the original algorithm with Clustal W for all seeds and all numbers of demonstrations to provide reference values for our data. Except for the seeds and numbers of demonstrations, the parameters of the Align-RUDDER algorithm we used were the same as the default parameters of the code version that we downloaded [18]. However, we noticed that they were not the same as in the original study by Patil *et al.* [14]. Align-RUDDER's code is likely to change when new versions are released. To ensure the reproducibility of our results we include our parameters in Table 1 in the appendix.

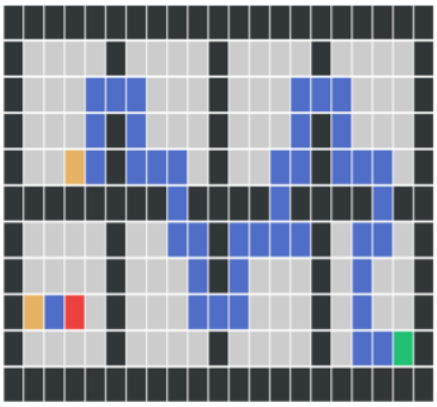
For the testing process we used the eight rooms environment by Patil *et al.* It is a variation of the Gridworld example of Sutton *et al.* [20], where the cells of the grid are individual states. In this setting, the agent's goal is reaching one particular cell that is the target state starting from an initial state. The eight rooms environment creates a 12x24 gridworld with 8 rooms, with the initial state in room #1 and the target state in room #8, shown in Figure 9a.

As the gridworld is compartmentalized into rooms, the agent needs to move through

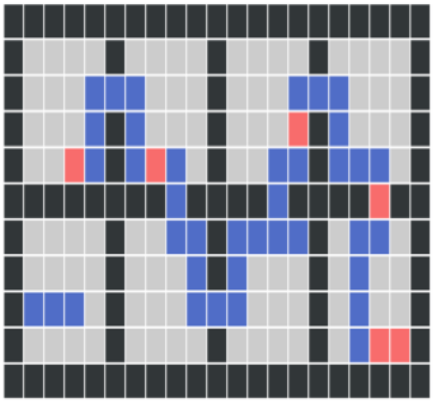
multiple rooms to get to the target states. All rooms can be entered and left through doors. Only the first room is completely encased by walls and a portal is the only way out. If the agent steps on the field that represents the portal, he is immediately teleported onto a particular cell of the second room. The entry cell for the portal in the first room is chosen randomly for each episode, while the cell in room #2, to which the agent is sent by the portal, stays the same. The portal ensures that the agent learns to look for the portal even if it is not in the same position as in the demonstrations.

As long as the agent stays on the grid, the agent has four possible moves in every state: up, down, left, and right. Each of them leads to a stochastic state transition, except for teleportation. After 200 timesteps the episode is over and the agent receives a reward of 3 if it reached the target state and 1 if it didn't. To force the agent to find the shortest way to the goal state, it receives a negative reward of -0.01 at every time step [14].

The reward redistribution is applied as explained in subsubsection 2.2.1. It redistributes the expected final reward from the end of the episode to the key actions (Figure 9b). To do this, subtasks, in this case reaching a door or the portal, must be identified [14].



(a) Setup of the eight rooms environment, the red cell is the initial state, yellow depicts the portal and the green cell represents the goal state



(b) Reward redistribution of the eight rooms environment, reward is redistributed towards the red cells

Figure 9: Eight rooms environment [14]

4 Results

The first set of analyses examined the impact of parameter settings for Clustal Ω on the sequence alignments. For this, we tested the MSA algorithm on its own before integrating it into Align-RUDDER. After that, we ran Align-RUDDER using different alignment parameters with multiple numbers of demonstrations. We then compared our results to the original Align-RUDDER algorithm which uses ClustalW.

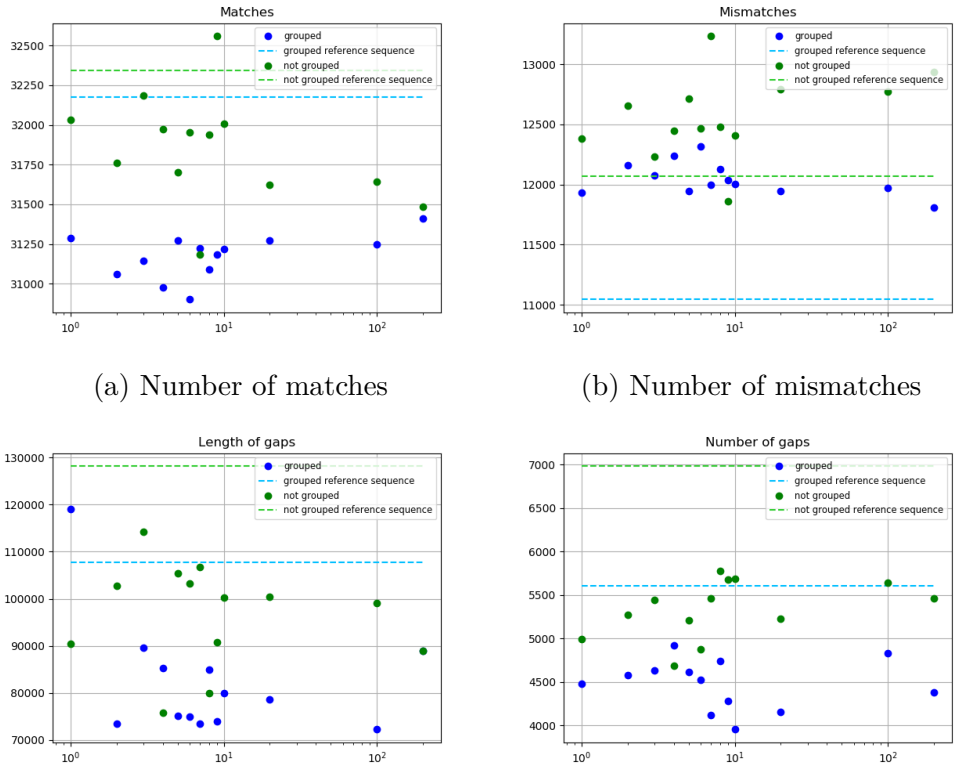
4.1 Clustal Ω parameters

As Sievers *et al.* [19] showed in their study, the EPA approach yields a worse result for data sets smaller than 1000 sequences. However, they used the TC score, the average of the total columns of the alignment, to compare the alignments. We compared the number of matches, mismatches, and gaps, as we used non-biological data.

The data was obtained from two different alignment procedures. Initially, we ran the iterative alignment process on the whole data set. Then, in the other procedure, the data set was split into three groups, depending on the sequence length. The same iterative alignment algorithm was applied on the three groups. For reference purposes, we also aligned both the grouped and ungrouped sequences once without any iterations.

Then we counted the occurrences of matches, mismatches, the length, and the number of gaps for all alignments of the data set. This is depicted in Figure 10.

As discussed in section 3, all characteristics of both samples were checked for



(a) Number of matches

(b) Number of mismatches

(c) Total length of gaps

(d) Total number of gaps

Figure 10: Alignment values

Gaussian distribution. Since all of the Lilliefors tests have a positive result, the

assumption that the data is normally distributed holds. This means that we can calculate a one-sided 95% confidence interval for all parameters and both samples. The paired t-test can also be carried out, as it requires normal distribution as well.

Looking at Figure 10a, the results suggest that both grouped and ungrouped sequences contain significantly fewer matches than their respective reference sequence. The reference value produced without iterations was above the 95% margin for both groups, this proves that the iteration leads to a noticeably smaller amount of matches. This result is consistent with the result of Sievers *et al.* [19], as we have less than 1000 sequences in our data set. Since the previous study does not involve data sets that are smaller than 100 sequences, it was uncertain whether or not splitting our data set in even smaller groups would improve the alignment quality when it is combined with iterations. Our results suggest that if you split the data into even smaller sets, the effect of iteration becomes even worse. The ungrouped sample has more matches than the grouped sample. The paired t-test confirmed a noticeable difference between the grouped and the ungrouped data. This is evidence that separating the data into groups before the alignment has a detrimental effect on the number of matches in the alignment.

Figure 10b shows that there are noticeably more mismatches when iterations are used, as the reference values are lower than the means of the sample data for both groups. Regrading the mismatches it is important to consider that fewer mismatches make a better result. As opposed to the matches, where a high number of matches is desirable. Taking this into account, the result is still consistent with data from the previous study by Sievers [19]. The effect of dividing the data set into groups was analyzed for the mismatches as well. The statistical analysis confirmed a noticeable difference, the mean value of the grouped data set was lower than the one of the ungrouped data. In other words, regarding mismatches, separating the data has a positive effect on the alignment as opposed to the matches.

When we look at the graphs Figure 10c and Figure 10d, both the length and the number of gaps show similar results. Both visualizations showcase that the ref-

erence values are higher than the sample data for both, grouped and ungrouped samples. In sequence alignment shorter and fewer gaps are usually desirable. This means that when the gaps are compared, the iteration has a positive effect on the alignment. This is contrary to the results of Sievers [19]. By comparing the grouped and the ungrouped sample, we found that dividing the sequences into groups noticeably reduces the amount and the length of gaps present in the MSA.

Comparing the four graphs in Figure 10, no clear relation between either of the 4 characteristics or the amount of iterations is visible. The number of matches, mismatches, and gaps as well as the gap lengths are practically constant for all iterations.

4.2 Clustal Ω in Align-RUDDER

Summarising the results of the comparison in subsection 4.1, the ungrouped alignment that does not use iterations has the most matches, the grouped alignment without iterations has the least matches, while the grouped alignments with iterations have the least gaps. This means that there is no 'best' alignment strategy, they all have advantages and disadvantages, which is why we tested all of the strategies in combination with reward redistribution. However, as it is imperative for relevant events to be aligned to themselves [14], we expect the strategy that leads to the most matches to be the most promising approach. As we do not know which events are relevant beforehand, the other approaches could lead to the best result as well. After integrating the alignment algorithm into Align-RUDDER, we used different seeds and various numbers of demonstrations to test the strategies, as described in subsection 3.4.

As can be seen in the results displayed in Figure 11, none of the alignment strategies are significantly better than the original ClustalW alignment algorithm. The points shown in the graph represent the median of the 10 seeds. All seeds were used for each number of demonstration, for each alignment strategy. The best of our strategies is, as expected, the ungrouped alignment without iterations. It produced the most matches in the trial alignments. Except for the runs with

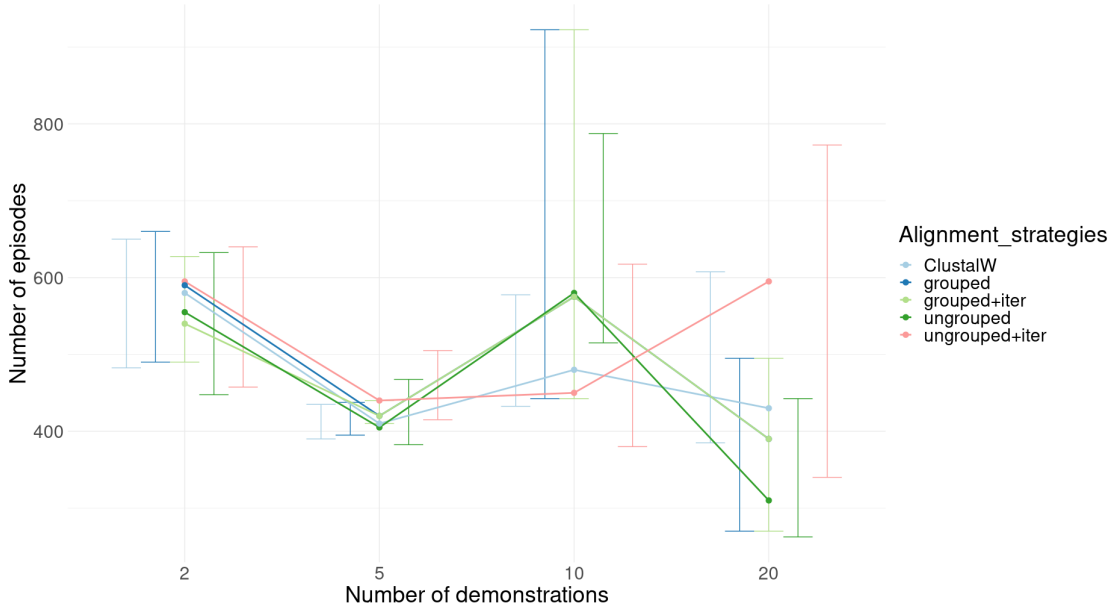


Figure 11: Comparison of Align-RUDDER algorithms with different alignment parameters, the ends of the error bars represent the 1st and 3rd quartiles.

10 demonstrations, it performs slightly better than the ClustalW alignment algorithm, which was used by Patil *et al.*

Since the error bars of the ungrouped alignment without iterations in Figure 11 almost always include the median of the ClustalW strategy, this result is not statistically significant. Most of the error bars contain the reference values of the ClustalW algorithm. No statistical analysis was conducted, since it was clear that there was not enough difference between each of the groups to reject the H_0 hypothesis. Especially for small numbers of demonstrations, the key application area of Align-RUDDER, all strategies produce almost identical results. It diverges when more demonstrations are included. However, none of our strategies show a significant improvement compared to the already existing algorithm. Particularly noticeable is the fact that the effects of iteration and the grouping of the sequences seem to interact with each other when the demonstrations are increased. The results of the algorithms that separate the sequences into groups produce the same medians and quartiles regardless of the iteration process. In contrast, the results of the two ungrouped strategies show the largest differences out of all samples.

5 Conclusion

Prior works established that using an alignment as a template for EPA has a beneficial effect on alignments of samples containing more than 1000 sequences, while it has a negative influence on smaller data sets. However, data sets smaller than 100 sequences were not included in the study and only the effects of 2 consecutive iterations were taken into account [19].

In this study, we experimented with both different amounts of sequences and varying amounts of consecutive iterations. While multiple iterations do not seem to affect the alignment quality for small sample sizes, separating the data into even smaller subsets caused a noticeable reduction of mismatches and gaps. This confirms that smaller data sets have some advantages when it comes to alignment quality. However, our data also indicates that when Clustal Ω is used, bigger samples seem to produce a larger number of matches, which is important in RL as common patterns need to be recognized. Opposed to previous works, this paper does not rely on the TC score, or total column average, but uses the amounts of matches, mismatches, gaps, and the gap length instead to evaluate its alignments. This is a more versatile and flexible way to assess alignment quality, which can be used for different kinds of sequential data. Our results suggest that for small samples of 100 sequences or less, iteration has a detrimental effect on the amounts of matches and mismatches in the alignment, which is consistent with Sievers *et al.* [19]. This means that for ML cases with only few demonstrations to learn from, it is better to abstain from using iterations. We further discovered that the choice of alignment parameters has little to no effect on the results when the number of demonstrations is below or equal to 5. While there seems to be some effect when the number of demonstrations is increased, the precise nature of said effect remains unclear. Future work should focus on experiments that examine a higher amount of demonstrations to investigate the effects of sample size and the EPA approach when using MSAs.

References

[1] Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards, 2019.

[2] Jeremy M Berg. *Biochemistry*. Freeman, New York, NY [u.a.], 6. ed., 2. print.. edition, 2007.

[3] Gordon Blackshields, Fabian Sievers, Weifeng Shi, Andreas Wilm, and Desmond G Higgins. Sequence embedding for fast construction of guide trees for multiple sequence alignment. *Algorithms for Molecular Biology*, 5(1):21, 2010.

[4] Supratim Choudhuri. *Bioinformatics for beginners : genes, genomes, molecular evolution, databases and analytical tools*. Academic Press, 2014.

[5] EMBL-EBI. FASTA, Protein Similarity Search. "`https://www.ebi.ac.uk/Tools/sss/fasta/`. Accessed: 2020-29-11.

[6] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. MineRL: A large-scale dataset of Minecraft demonstrations. *Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019.

[7] Markus Hofmarcher. Align-RUDDER. "`https://github.com/ml-jku/align-rudder`". Accessed: 2020-19-11.

[8] P. Hogeweg and B. Hesper. The alignment of sets of sequences and the construction of phyletic trees: An integrated method. *Journal of Molecular Evolution*, 20(2):175–186, June 1984.

[9] Prof. Dr. Wolfgang Höhne. Proteine: Peptidbindung und Proteinstruktur. "`https://viamedici.thieme.de/lernmodul/548810/subject/biochemie/aminos%C3%A4uren+peptide+proteine/proteine/proteine+peptidbindung+und+proteinstruktur`". Accessed: 2021-23-2.

[10] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[11] Zoé Lacroix and Terence Critchlow. *Bioinformatics : managing scientific data*. The Morgan Kaufmann series in multimedia information and systems. Morgan Kaufmann Publishers, 2003.

[12] David W Mount. *Bioinformatics : sequence and genome analysis*. Cold Spring Harbor Laboratory Press, 2. ed. edition, 2004.

[13] Saul B. Needleman and Christian D. Wunsch. 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, March 1970.

[14] Vihang P. Patil, Markus Hofmarcher, Marius-Constantin Dinu, Matthias Dorfer, Patrick M. Blies, Johannes Brandstetter, Jose A. Arjona-Medina, and Sepp Hochreiter. Align-rudder: Learning from few demonstrations by reward redistribution, 2020.

[15] Hazhir Rahmandad, Nelson Repenning, and John Sterman. Effects of feedback delay on learning. *System Dynamics Review*, 25(4):309–338, October 2009.

[16] Pedro F. Rodriguez, Luis F. Niño, and Oscar M. Alonso. Multiple sequence alignment using swarm intelligence. *International Journal of Computational Intelligence Research*, 3(2), 2007.

[17] Fabian Sievers and Desmond G. Higgins. Clustal omega, accurate alignment of very large numbers of sequences. In *Methods in Molecular Biology*, pages 105–116. Humana Press, August 2013.

[18] Fabian Sievers, Desmond G. Higgins, Andreas Wilm, and David Dineen. Clustal Omega. "`http://www.clustal.org/omega/`". Accessed: 2020-10-12.

[19] Fabian Sievers, Andreas Wilm, David Dineen, Toby J Gibson, Kevin Karplus, Weizhong Li, Rodrigo Lopez, Hamish McWilliam, Michael Remmert, Johannes Söding, Julie D Thompson, and Desmond G Higgins. Fast, scalable

generation of high-quality protein multiple sequence alignments using clustal omega. *Molecular Systems Biology*, 7(1):539, January 2011.

[20] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181 – 211, 1999.

[21] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):2–9, September 1998.

[22] Julie D. Thompson, Desmond G. Higgins, and Toby J. Gibson. CLUSTAL w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[23] W. J. Wilbur and D. J. Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences*, 80(3):726–730, February 1983.

Appendices

Hyperparameters of Align-RUDDER	
Parameter	value
total amount of timesteps	10 000 000
maximum number of episodes	20 000
learning rate	0.1
gamma (discount factor for q learning)	1.0
seeds	0-9
epsilon (exploration constant)	0.1
stopping criterion	80opt (until 80% optimal return is reached)
annealing rate for exploration constant	1.0
experience buffer size	30 000
maximum reward	1

Table 1: Align-RUDDER parameters