Influence of Clustal W hyperparameters in multiple sequences alignment for AlignRUDDER

Submitted by

Marlene Ganz

Submitted at

Institute for

Machine Learning

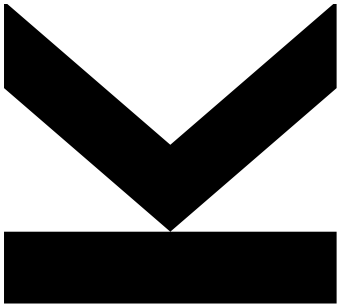
Supervisor

Univ.-Prof. Dr. Sepp Hochreiter

Co-Supervisors

José A. Arjona-Medina, PhD

January 2021



Bachelor Thesis

to obtain the academic degree of

Bachelor of Science

in the Bachelors's Program

Bioinformatics

JOHANNES KEPLER

UNIVERSITY LINZ

Altenbergerstraße 69

4040 Linz, Österreich

www.jku.at

DVR 0093696

Bibliographical Detail

Ganz, Marlene, 2021: Influence of Clustal W hyperparameters in multiple sequences alignment for AlignRUDDER. Bachelor Thesis, in English. 48 p., Institute for Machine Learning Johannes Kepler University, Linz, Austria

Annotation

Reinforcement learning algorithms suffer from the delayed reward problem and usually only perform well when being trained with vast amounts of data. AlignRUDDER overcomes this problem by using a multiple sequence alignment for the initialization performed with ClustalW. In this thesis we work with different data-sets and AlignRUDDER to search for optimal alignment hyperparameters for reinforcement learning problems.

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

Marlene Ganz

Contents

1 Abstract **iv**

2 Introduction **1**

2.1 Reinforcement Learning 2

3 Related Work **4**

3.1 Sequence Alignment 4

3.2 Multiple Sequence Alignment 5

3.3 Multi Sequence Alignment Algorithms 6

3.3.1 ClustalW 6

3.3.2 ClustalΩ 8

3.4 What is a good alignment ? 9

3.5 Reinforcement Learning 11

3.6 RUDDER 11

4 Methodology **13**

4.1 MineRL Dataset 13

4.2 Optimal parameter search for the alignment of Minecraft sequences 14

4.2.1 Creation of the scoring matrix 14

4.2.2 Multiple Sequence Alignment 16

4.2.3 Consensus Strand 17

4.2.4 Grid Search for the optimal parameters for the alignment 17

4.3 Optimal parameter search for RL agent in AlignRUDDER 19

4.3.1 AlignRUDDER 19

5 Results and Discussion **20**

5.1 Results for the optimal parameter search for the Minecraft sequences 20

5.1.1 Parameter selection for the Minecraft sequences 21

5.1.2 The grid search 22

5.2 Results for the optimal parameters search for AlignRUDDER 29

6 Conclusion **37**

6.1 Parameter Search for the Minecraft Task 37

6.2 Parameter Search for the AlignRUDDER Task 38

List of figures **39**

List of tables **41**

References **42**

7 Appendix **45**

1 Abstract

Reinforcement learning algorithms suffer from the delayed reward problem and usually only perform well when being trained with vast amounts of data. One way to overcome these problem is the use of demonstrations for training, from human or current policies which are attempted to be improved. But demonstrations are often limited, which makes learning a policy difficult. AlignRUDDER performs well even with few demonstrations as it combines effective sequence alignment and RUDDER. In this paper, we search for optimal alignment hyperparameters to be used for reinforcement learning problems. For our search we tested various parameters and analysed their effect on the alignment and on the behaviour of a reinforcement algorithm. The results showed that the alignment is strongly influenced by the chosen parameters and that the parameter depends on the sequence set. Additionally, we observed that for the reinforcement agent the quality of the alignment is less important when increasing the number of demonstrations. In summary, we were able to give recommendations for optimizing the hyperparameters, but the concrete values depend on the concrete problem.

2 Introduction

Multiple sequence alignment is a widely used method in Bioinformatics to investigate similarities in between amino acid sequences or nucleotide sequences. Biological conclusions can then be derived from the resulting alignments. The alignment depends on many parameters that need to be adjusted for the particular input sequences. The alignment algorithm can be optimised using alignment parameters.

The DNA of each organism contains the hereditary information packed in a four letter code (A,T,G,C). These are enough to contain all necessary information for the cells to build any cellular compound on Earth. This code is transcribed into messenger RNA (mRNA) which is then translated by the ribosomial units into an amino acid sequence. Proteins, which are amino acid sequence chains, are the building blocks of organisms. Most living things form similar proteins with only minimal differences hidden in the amino acid sequence. Researchers have been using these differences for decades to trace evolutionary processes. Sequence Alignment is the process of finding the similarities within two sequences.

The optimal alignment is where the maximum number of similar residues are matched. Mismatches (two different letters aligned) can occur and are being caused by mutations which is when individual amino acids or nucleotides changed. Gaps can appear between the matches, which can be a sign of evolutionary information loss. Causes can be the evolutionary insertion of additional residues (insertion), the deletion of one to many residues (deletion), the relocation of chromosome pieces into other chromosomes (translocation) or genetic inversions [4].

Sequence alignment is not limited to amino acid sequences but can also be applied in the same form to DNA and RNA, or to any problem that seeks to align sequential structures according to their similarity. This idea is used in the AlignRUDDER [23]. Here, the initial knowledge of a reinforcement learning agents is based on a multiple alignment of previous successful runs. This paper presents a way to find parameters to align these non-biological sequences reasonably. We used ClustalW [5] to align the sequences with different parameters. ClustalW is a very powerful multiple sequence alignment tool.

2.1 Reinforcement Learning

In reinforcement learning, the learner is not taught which steps lead to which goal. Similar to human children, they have to learn by trying different actions and receiving punishment or reward. Certain actions can, in turn, influence other actions and also future rewards. Trial and error and delayed reward are the two fundamental principles of reinforcement learning and are based on the ideas of human learning [29]. The important factor in this learning model is that the most important aspects for solving a problem and a goal are provided. The learner must be able to perform actions in his environments and to grasp and change his respective state.
What distinguishes reinforcement learning from other machine learning algorithms such as unsupervised or supervised is that the learning agent both selects the actions for which it has received a reward in the past and must try new actions to gain such knowledge. This allows the agent to understand the whole problem and not just learn bluntly based on labeled data without a predefined goal.
The four most significant components of a reinforcement system are: reward signal, policy, value function and model. The agent's main goal is to maximize its future expected sum of rewards. The reward, a simple number, is sent to the agent directly after each step. So, there can be steps in the environment that lead to more or less reward. This knowledge about such relationships is included in the policy. The policy reflects the behavior of the agent. It can be a simple tabular structure, but it can also be a complex function. Rewards influence the policy. An action with more rewards will be performed more often than one with less rewards. Compared to the immediate reward, the value function is relevant for the agent's long-term decision making. The value tells how high the worthy the action is for the future and is based on what the agent has learned. Values are much more complex and can only be assigned in retrospect, but they have a major role in reinforcement learning [29]. The model helps to make predictions for the future. There are two approaches model-based and model-free reinforcement learning. In the model-based approach, the policy is based on a machine learning model such as random forest. In the model-free approach, on the other hand, there is no machine learning model for the policy. In such cases, the policy tries to find a balance between load and effort [25].

The thesis is structured as follows. First we review the related work. Then we describe our

methods in the methodology section. In the result section we show our results and discuss them. Finally, there is a conclusion that summarizes the thesis.

3 Related Work

In this thesis we combine two different fields of science. We combine bioinformatics topics like sequence alignment with machine learning methods, especially reinforcement learning. In the following section, we address related works to provide a basis for the respective field.

3.1 Sequence Alignment

Sequence Alignment is the process of finding the similarities within two sequences. The optimal alignment is where the maximum number of similar residues are matched. While there are matches, mismatches can also occur or gaps can be introduced. Both mismatches and gaps may have appeared from evolutionary or spontaneous mutations. Therefore, sequence alignment is necessary to investigate similarities in between amino acid sequences or nucleotide sequences. Biological conclusions can then be derived from the resulting alignments. Differences and similarities in the sequence of organisms can be used to trace evolutionary processes. Sequence alignment can be divided into two methods, the pairwise alignment (PA) and the multiple sequence alignment (MSA). PA aligns only two sequences. For two sequences it is possible to reach the optimal alignment. When dynamic programming [13] is used for PA, the resulting alignment will always be the optimal one [4]. The informative power about homologies and evolutionary relationships is low in contrast to MSA. While two very short sequences can easily be optimally aligned by hand, it is very difficult for longer ones. The number of possible alignments increases with the length of the sequences.

The Needleman-Wunsch algorithm [20] describes a method to find the optimal alignment of two sequences. The method, published in 1970, calculates the optimal alignment with the help of a similarity score. For this purpose, a match, a mismatch score and a gap penalty are determined. The Similarity Score, the sum of the match and mismatch scores and the gap penalty, is calculated for the different alignments and the alignment with the highest Similarity Score corresponds to the best alignment. This algorithm can be performed with dynamic programming.[20].

While the Needleman-Wunsch algorithm yields optimal global alignment, the algorithm was modified by Waterman and Smith in 1981 to be able to optimise local alignment [28]. A global

alignment tries to align the complete sequences optimally. For this reason, sequences of the same length are particularly suitable for this process. The local alignment focuses on parts of the whole sequence. In this way, a local alignment tool finds the local regions in the sequences with the highest similarity. This difference allows this method to find local similarities in sequences of any length. But finding a local alignment does not mean that the sequences are globally optimal aligned.

3.2 Multiple Sequence Alignment

Multiple Sequence Alignment (MSA) is an essential tool in bioinformatics. In this algorithm more than two sequences are compared with each other. Homology and evolutionary processes can be determined from the resulting alignment and an example of an alignment can be seen in figure 1.

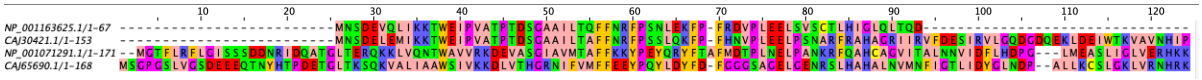


Figure 1: **Example of an MSA performed with ClustalW.** Alignment of the globin 1 protein of Drosophila melanogaster, Drosophila virilis, honey bee and mosquito (Anopheles gambiae)

There are many different schemes to determine the level of identity between sequences. For nucleotide sequences, a positive value is given for a match. For proteins, in addition to the exact identity of two amino acids, similarity in terms of physicochemical properties is also considered. To put these dependent values together, a scoring matrix is used. Typically either block substitution matrix (BLOSUM) [11] or point accepted mutation (PAM) [4] are used.
Unlike PA, MSA is not easy to compute and requires a lot of computing power. MSA is a high dimensional problem, as the dimensions correspond to the number of sequences. As a consequence, computing the exact MSA is np-hard and thus almost impossible. Known MSA algorithms use heuristics or approximations for the calculation. Thus, a quite good result can be achieved quickly [4].
Alignment algorithms use different parameters (gap penalties and scoring matrices, among others) to improve the result. It is important to select these parameters cautiously because inappropriately chosen parameters ”will arrive at a false global minimum” [30], and that results in a wrong alignment. Moreover, the result needs to be biologically reasonable. Mutations are

biological processes and follow chemical rules. The probability of one amino acid changing to another is known because of its chemical similarities and already existing alignments. It can be differentiated between three kinds of alignment strategies exact, progressive and iterative [4].

One of the biggest problems with multiple sequence alignment is the complexity caused by the number of sequences (N) and their lengths (L) . A lot of computational power is needed to receive a reasonable alignment. Having N sequences of length L the computational complexity is O(L^N). Therefore this can only be calculated for a small amount of sequences [27]. For this reason, most methods [27] apply a heuristic progressive alignment approach. For this progressive approach, the large problem of the MSA is broken into many small PAs directed by a guide tree. Through this approach the complexity can be reduced to O(N^2). Nevertheless, it is only possible to align a few thousand sequences to a moderate length [27]. Another problem is that progressive algorithm have a high chance to find only a local minimum and not a global one. This problem arises due to the greedy behavior of the algorithm. The algorithm gradually assembles the sequences based on a previously created guide tree. Errors that occurred during the creation of the tree as well as errors in initial alignments cannot be corrected [27] [30]. There is no guarantee to reach an optimal result with these greedy alignment approaches [30]. Newer alignment methods like T-Coffee [21] solved the problem partly by introducing a consistency principle and could improve the accuracy by at best 10% [27].

3.3 Multi Sequence Alignment Algorithms

3.3.1 ClustalW

The most commonly used multiple alignment tools come from the Clustal series [3]. Clustal products have been on the market since 1988. In this paper we used ClustalW [30], the third generation of the Clustal series. The algorithm was developed to align nucleotide and amino acid sequences optimally. ClustalW uses heuristics to reduce the complexity of the problem [31]. The ”W” in ClustalW stands for weight. ClustalW differs from its predecessors in position- and residue specific gap penalties and a weighting pattern that discounts over represented sequence groups [1]. Residue specificity is based on the fact that gaps in biological sequences do not occur randomly. At certain positions within an amino acid sequence, it is the

A	1.13	M	1.29
C	1.13	N	0.63
D	0.96	P	0.74
E	1.31	Q	1.07
F	1.20	R	0.72
G	0.61	S	0.76
H	1.00	T	0.89
I	1.32	V	1.25
K	0.96	Y	1.00
L	1.21	W	1.23

The values are normalised around a mean value of 1.0 for H. The lower the value, the greater the chance of having an adjacent gap. These are derived from the original table of relative frequencies of gaps adjacent to each residue (12) by subtraction from 2.0.

Figure 2: **Multiplicand table for amino acids.** The image shows a table with the values by which the Gap Open Penalty is multiplied. After certain amino acids, if not hydrophilic stretches, gaps are more likely to happen than after others. The table is from the paper [30] and the calculations are based on the paper [22].

likelihood that a gap will occur differs. These specificities are taken into account. Thus, the gap open penalties are multiplied by one of the numbers in figure 2, since the probability of a gap following an Asparagin (N) is higher than for an Methionine (M) [30].

Since ClustalW appearance, it has been modified further to improve and sensitize the program[17]. The sequences are aligned in the ClustalW program by a modification of the progressive multiple alignment method [8]. The multiple alignment is made by sequence of pairwise alignments. The alignment with ClustalW is done in four steps (figure 3). First, the sequences are pair-wisely aligned. This means that only two sequences are compared with each other at once. Second, an unrooted guide tree is generated by calculating the similarity scores of the pairwise alignment. Thirdly, this tree is rooted and guides the following progressive programming in the fourth step. Dynamic programming is used at each step to merge the already aligned sequences from the previous step with the weight matrix and gap open/extension penalty [1].

ClustalW allows to use a variety of weighting matrices, both very well-known ones like PAM or BLOSUM but also custom weighting systems can be used. The user can select the parameters for the PA as well as for the multiple alignment [1]. The previous Clustal programs offered only UPGMA as a guide tree, now Neighbour Joining (NJ) is used as default, ”which is more

robust against the effects of unequal evolutionary rates in different lineages" [30]. The quality of an alignment depends on the similarity of the sequences as well as the respective parameters. The more divergent sequences are the harder it is to find an optimal alignment. ClustalW also serves as a good starting point for different sequences, which may need some refinement. But with good parameter selection, a ClustalW alignment gives a good direction [30]. "In cases where a lot the sequences in a dataset are very similar (e.g. no pair less than 35 % identical), CLUSTALW will find an alignment which is difficult to improve by eye" [30].

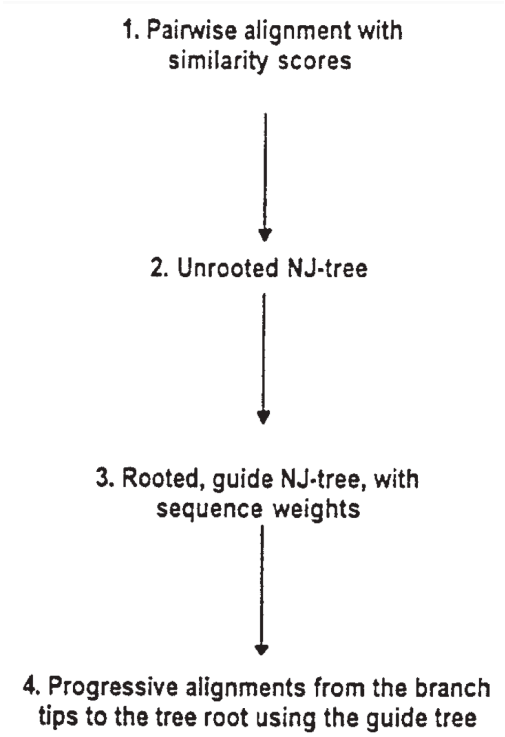


Figure 3: **Four Steps of ClustalW Alignment.** The figure illustrates the four steps of multiple alignment with ClustalW. ClustalW performs the Multiple Sequence Alignment by a modification of the progressive multiple alignment method [8]. The figure originates form the paper [1].

3.3.2 ClustalΩ

As described before, many alignment algorithms are based on the progressive heuristic approach. They have the problem of reaching a local minimum due to its greedy behavior. Nevertheless, ClustalW, MAFFT/PartTree [16] and others create good alignments. The alignments are better if the sequences are more similar[30] but the length and number of sequences used as input are limited. The latest Clustal method ClustalΩ [7], on the other hand, is an approach

to align a large number of sequences as accurate as possible [27]. ClutalΩ "is accurate but also allows alignments of almost any size to be produced" [27]. Unlike MAFFT, ClustalΩ manages to reduce computational complexity to O(Nlog(N))by using an Embedding method. A space with n dimensions ('n proportional to log N' [27].) is created. Into this space the sequence, which have been replaced by an n-element vector, are inserted. One element corresponds to the distance to one of n reference sequences is. This vectors representation simplifies the clustering. Clustering is performed with UPGMA or k-means. The alignment is then calculated by the exact HHalign package [7], the algorithm is described by [26]. Furthermore, ClustalΩ now provides the possibility to add new sequences to already aligned sequences, which saves the precomputation effort. Hidden Markov Models (HMM) are exploited in the alignment process. HMMs that are homologous to the input can be specified by the user in ClustalΩ. There are already many HMMs at disposal that can be used. ClustalΩ can be utilized for datasets with more than 10 000 sequences. The MAFFT algorithm is faster but ClustalΩ is more accurate [27].

3.4 What is a good alignment ?

Alignment algorithms, such as ClustalW, ClustalΩ, and others, try to find the optimal alignment by using of heuristics for the inserted sequences and the given parameter (Gap penalties and scoring schemes). However, parameters can change this result and then the question arises which solution is the better one. In genetic alignment tasks, the knowledge of amino acids and nucleotides accumulated over many years can be used to determine a good alignment. Mutations occur randomly, but the fact that the mutation is not detected and corrected by the organism's own control system is not random. In addition, the mutation must prevail and be accepted as a predominant form. It is more likely that a mutated amino acid is accepted if it is similar to the amino acid it replaces. Similarity refers to its physical and chemical properties [6]. To gain this knowledge, many sequences and their mutations have to be observed. This was used to create one of the standard scoring systems the Point Accepted Mutation Matrix (PAM) [6]. The figure 4 illustrates that the very similar Amino Acids Aspartic acid (Asp) und Asparagine (Asn) Have one of the highest values, that shows that it is likely that one mutates in the other one [6]. An abundance of previously aligned sequences was available,

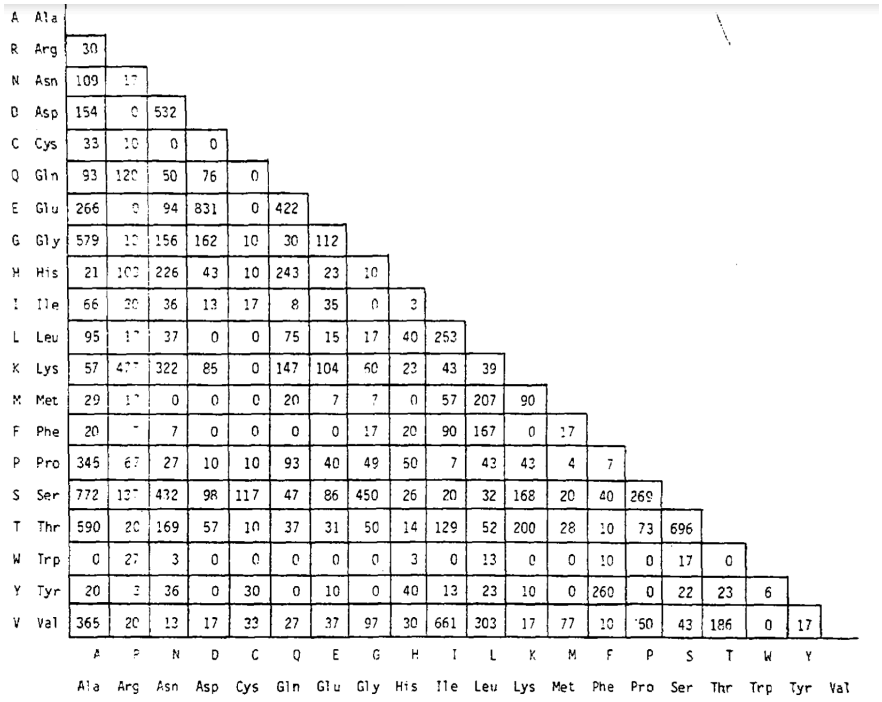


Figure 4: **PAM Numbers**The figure shows the accumulated number of point accepted mutations mulitplied by 10. The observations were done on basis of closely related sequences[6]. The figure originates form the paper [6].

so these assumptions of the likelihood of a change of amino acid A to amino acid B could be done. Moreover, information over the evolutionary distance is available because of phenotypic characteristics or existing phylogenetic trees. Consequently, suitable scoring matrices can be chosen to achieve an 'optimal' alignment. This prior knowledge lacks in this task. The most difficult challenge in this task was the missing information about the sequences. We got sequences of tasks and we did not know their associations. Certainly, amino acid properties (physical or chemical) were not significant because the letters in the sequence did not represent amino acids. We decided to use the alignment length, the consensus strand, and the score as indicators of a good alignment. The alignment length has helped to observe whether the sequences are aligned or just placed after one another. If no gap penalty was used, arranging the sequences successively was the optimal alignment for the alignment algorithm. The consensus strand has shown how stable the alignment is. The consensus is the conjoint sequences of all sequences. No consensus strand being present could be an indicator of an unfavorable alignment. Additionally, the length has indicated the accuracy of the alignment. If the consensus

strand had a length of ten letters, but all sequences had a length of thousand that indicates that the alignment did not succeed or that the sequences are distinct. Lastly, the alignment score was difficult to interpret. The score was depended on the previously chosen parameters as it is just the sum of all the matches, mismatches, and gap penalties. If a scoring matrix with higher mismatch penalties was applied, the overall score was smaller. We used the score to compare the different parameters, but not to relate the scores of different matrices. In summary, we used the alignment length, the consensus strand, and the score as metrics for a good alignment.

3.5 Reinforcement Learning

Reinforcement learning attempts to replicate human learning. Through trial and error, an RL agent investigates the world and learns in the process. In model-based RL, the algorithm has a model that helps to make predictions for the future and thus selects its next steps. Defining sufficiently detailed models is often very difficult. In addition, these algorithms are difficult to use for real life projects, because there is no simulation for the real world [12].

Imitation learning is based on the idea of a supervised machine learning method to learn rules of the world by utilizing demonstrations. Demonstrations are examples of how others (for example humans) have solved a problem. This learning is also found in humans. While machine learning algorithms always start from scratch when learn, humans have the comparison with other humans role models [12] [14]. RUDDER [2] and [12] are two examples that combine RL with imitation learning successfully.

3.6 RUDDER

Return Decomposition for Delayed Rewards (RUDDER) [2] is a new reinforcement algorithm applicable for problems where delayed rewards occur. The basic idea of RUDDER is based on a Markov Decision Process (MDP)[18]. The special characteristic of the MDP used in RUDDER is that the expected future rewards equal to zero [2].

In many real-world problems, there are delayed rewards. For example, the RL's task is to find the exit from a maze. Whether the steps the agent chooses are correct is only revealed at the end, when the exit has been found. Then the reward must be distributed back to the steps taken. Instant rewards are given to the agent immediately after performing an action. These rewards

make it easier for the agent to learn than if it has to wait to see what rewards will be given in the future. This is the idea of RUDDER. If the expected future rewards are zero, it is easier to estimate the Q-Values.In MDPs Q-Values are the sum of the expected immediate reward and the future reward. Future Reward are problematic because of problems with the bias (in TD learning) and with high variance problems (in MC learning). Both of these problems manifest themselves even worse when rewards are delayed. The Q-Values estimates the quality the possible actions to take. The fact that delayed rewards appear complicates the calculation of this estimate. RUDDER intents at making the expected future rewards zero if this goal is achieved, this simplifies the Q-value estimation [2].

To get the future value zero two new concepts are required: reward redistribution and return decomposition. Reward Distribution must redistribute the reward that the new concept, the MDP with zero expected future reward is equivalent to the delayed reward MDP. Generally, with MDP it is not possible to reach zero expected rewards. Due to this Sequence Markov Decision Process (SDP) are introduced. This new implemented function is able to have zero expected future rewards and the Q values and the subsequent optimal policies can be calculated as usual [2]. The mathematical detail is described in this publication [2].

Return decomposition is important for return redistribution to work. The idea behind this concept is that the actual RL task is transformed into a regression task, more precise a pattern recogniztion task. The regression task is to predict the sequence-wide return from the entire state action sequence. The regression task identifies which state-action pairs to predict return distributed the respective return. This transformation from RL task to regression task is done via contribution analysis. At the regression task deep learning algorithms can be used to recognize patterns. Learning is based only on complete episodes, thus reducing problems with unknown state-action pairs.

The most important task of the algorithm is to recognize the return of the entire sequence and then to divide it among the respective state action pairs. The division of the rewards to the state action pairs is done by contribution analysis. The Q value difference (expected return at sequence begin - the expected return at sequence end) is the pattern the Deep Learning approaches use [2]. RUDDER uses Long Short Term Memories (LSTM) to perform the Return decomposition.

Tests [2] showed that RUDDER is better than all other common RL algorithms when there are delayed rewards. If this is not the case, the RUDDER algorithm is not the most effective, because it takes longer due to the LSTM and has problems with particularly long sequences [2]. The RUDDER was adapted to be able to learn even from less demonstration. The reward redistribution by the LSTM in the original RUDDER was replaced by a multiple sequence alignment. Deep learning methods need many demonstrations for learning. Therefore the LSTM was replaced by multiple sequence alignment of successful demonstrations [23].The initialization of the RL algorithm is based on a multiple alignment [23].

4 Methodology

In the following, we performed two different experiments. First, we observed the alignment behavior of alignment algorithm ClustalW with respect to various parameters when applied it to sequences of non-biological origin. Secondly, we analyzed the behavior of a reinforcement learning (RL) agent whose knowledge is based on qualitatively different alignments.

4.1 MineRL Dataset

”Minecraft is a 3D, first- person, open-world game centered around the gathering of resources and creation of structures and items”[10]. The game can be played both alone as a single player version or together in multiplayer mode. A game is played for several hours and sessions per player. The game environment in Minecraft consists of square blocks, the players can intervene in this environment and change it, so for example to extract wood from trees. The game is characterized by several sub-goals and interrelationships between different game processes. For example, in order to build a certain tool that is needed for another task, players must search for and collect certain materials[10].

Since Minecraft is an open-world sandbox game [19], there are no clear game objectives. Players can choose their own subgoals or make the game whatever they want, as long as they survive (when playing in Survival Mode) [15]. Survival mode implies that the agent can feel hunger, so he has to get food and protect himself from dangers, such as monsters. For the data collection of the MineRL dataset the survival mode was mostly used, as this is the most popular game variant and one of the most complex, as it contains many subtasks like navigation, item

collection and treechoping. Furthermore, time-limited tasks, such as ObtainDiamond, were included in the dataset. The dataset is described in detail by [10]. The "dataset consists of over 60 million automatically annotated state-action pairs across a variety of related tasks in Minecraft, a dynamic, 3D, open- world environment" [10]. Reinforcement learning usually requires a high amount of good quality data. Often such data is not available. That was the reason for creating the MineRL dataset, as it has the size, structure and quality to be utilized for reinforcement learning [10].

4.2 Optimal parameter search for the alignment of Minecraft sequences

For the first experiment, we worked with Minecraft game demonstrations to observe the effects of different parameters on the alignment of non-biological sequences. The Minecraft data originated from the MinRL Dataset[10]. We got amino-acid-like sequences that represent actions in a Minecraft game. Every sequence is one play from the start to the achievement of different targets. The Minecraft game consists of many subgoals that a player can choose depending on his strategy. We did not have prior information about which sequences correspond to what task or sub goal. The goal of the multiple sequence alignment was to detect similarities in separate games and find an optimal way of achieving the tasks.
In a game like Minecraft [15], there are many different steps/actions that a player can choose. In achieving the same goal or sub-goal, the game-play of different players will be similar. The game-play is not identical, as there are multiple paths leading to the goal. But the idea of the sequence alignment is based on finding exactly these similarities, as they might be the key points to reach the goal. Past experiments [23] applied to the Minecraft dataset [10] already showed great performance. We studied the influence of different parameters on the resulting alignment.

4.2.1 Creation of the scoring matrix

The alignment tool ClustalW used in our experiments performs the alignment in three steps. Firstly, the sequences are pairwise aligned. Afterward, the algorithm creates a phylogenetic tree that serves as a guide tree in the last step. In the end, the tree is used to perform the multiple sequence alignment [30]. For the optimal alignment strategy, the program requires

parameters such as scoring matrix and gap penalties. In the following paragraph we describe the construction of our scoring matrix.

A scoring matrix contains the probabilities of one amino acid changing to another. The matrix serves as a lookup table for the alignment algorithm. The diagonal of the matrix holds the match score that means that the amino acid remains the same. All non-diagonal entries represent mismatches consequently the costs for a mutation. Multiple scoring matrices are used. The most prominent scoring matrices are Point Accepted Mutation (PAM) [6] and Block Substitution Matrix (BLOSUM) [11]. We used the idea of PAM to create a scoring matrix that is suitable for our sequences of actions. For the calculation of the PAM, a mutability matrix ***M(i,j)*** is calculated. The mutability matrix contains the likelihood of amino acid ***i*** changing to ***j***. The probabilities are calculated with the assistance of previously aligned sequences [9]. The equation to calculate the PAM matrix:

$$PAM(i,j) = log(\frac{f(i)M(i,j)}{f(i)f(j)}) = log(\frac{M(i,j)}{f(i)})$$

[9] The variables ***i*** and ***j*** represent the amino acids. The frequency of ***i*** is calculated and proximate normalized by the total number of amino acids ***f(i)***. The mutability matrix ***M(i,j)*** holds the probability that ***i*** mutates to ***j*** given by entry of the mutability matrix.

$$f(i) = \frac{\text{number of occurrences of } \mathbf{i}}{\text{total number of amino acids}} f(j) = \frac{\text{number of occurrences of } \mathbf{j}}{\text{total number of amino acids}}$$

We calculated ***f(i)*** and by counting the occurrence of each amino and dividing it by the total number of amino acids that were present in the FASTA file. We could not calculate ***M(i,j)*** because we did not have prior knowledge about the sequences, but we could approximate it. We used the optimal ***M(i,j)***. In an optimal alignment, there are only matches and no mismatches. Hence, the optimal ***M(i,j)*** contains only two values:

- **i=j**:***M(i,j)*** is ***1*** - The matrix is optimal when i stays i
- **i** $\neq$ **j**::***M(i,j)*** is ***0*** - Zero means that no mutation from i to j occurred.

The theoretical approximation of taking zero could not be used because the logarithm of zero is not defined. But as we substituted it with 0, it could be seen that all non-diagonal entries are the same: the logarithm of zero. Besides the fact that it is undefined, it helped that we could

use the same value for all non-diagonal entries. Nevertheless, the diagonal entries could be determined using the following procedure: We counted all the letters in the given sequences and calculated the probability for each letter.

$$f(i) : \frac{\text{occurrence of one letter}}{\text{total number of letter}}$$

For all probabilities the logarithm $log(\frac{1}{f(i)})$ was calculated. The logarithms are the scores for matches and are placed in the diagonal of the scoring matrix. The values were rounded because scoring matrices have integers as records. Furthermore, the Minecraft sequences did not include the characters 'B', 'D', 'T', 'X', and 'Z'. Primal PAM matrices include all possible amino acids, so the alignment algorithm did not work without them. Hence we added one as diagonal value. Additionally, the original PAM matrix contains '*' column and row, which comprise the lowest record in the scoring matrix. Consequently, we added the star row and column as well. The non-diagonal entities could not be calculated easily, but as mentioned before, we knew that all have the same value. Hence, we used some approximation. We created different matrices to have some comparison. An example of a scoring matrix is given in figure 5.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	0	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
R	−1	8	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
N	−1	−1	6	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
D	−1	−1	−1	1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
C	−1	−1	−1	−1	8	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Q	−1	−1	−1	−1	−1	3	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
E	−1	−1	−1	−1	−1	−1	6	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
G	−1	−1	−1	−1	−1	−1	−1	3	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
H	−1	−1	−1	−1	−1	−1	−1	−1	4	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
I	−1	−1	−1	−1	−1	−1	−1	−1	−1	2	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
L	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	6	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
K	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	5	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
M	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	4	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
F	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	6	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
P	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	4	−1	−1	−1	−1	−1	−1	−1	−1	−1
S	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	3	−1	−1	−1	−1	−1	−1	−1	−1
T	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	1	−1	−1	−1	−1	−1	−1	−1
W	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	10	−1	−1	−1	−1	−1	−1
Y	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	5	−1	−1	−1	−1	−1
V	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	4	−1	−1	−1	−1
B	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	1	−1	−1	−1
Z	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	1	−1	−1
X	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	1	−1
*	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1

Figure 5: **Scoring Matrix with -1 non-diagonal value.** Image shows the scoring scheme with -1 as non-diagonal entry. The diagonal entries are the same in all our matrices. The matrices only differ in the value of mismatch penalty (non-diagonal entries). In this example the mismatch penalty is -1.

4.2.2 Multiple Sequence Alignment

The goal of this work is to understand how different parameters affect the alignment. To perform the multiple sequence alignment, ClustalW was chosen as multiple sequence alignment tool. We used the alignment program without any further changes. We only adjusted the available parameters.

4.2.3 Consensus Strand

The consensus strand is an important metric in our analysis. We needed it for the later evaluation of our results. The consensus is the conjoint sequences of all sequences. The consensus was computed from the alignment result by returning the most frequent letter in a column. If no character occurred, just dashes, the function returned nothing. Moreover, if the function could not select one of two letters because both appeared in the same amount in the alignment, it returned both letters with a slash in-between. Additionally, we added a threshold of 30 percent that means that just the letter emerged in the consensus strand that was in 30 percent of the column present. Through the addition of the threshold the chance of a slash occurring in the consensus is very low because each letter needs to be present in 30 percent of the column. An Example can be seen in figure 6.

Seq1	A	-	C	D	E
Seq2	-	-	C	D	E
Seq3	-	-	I	-	A
Seq4	-	-	H	-	A
Seq5	-	-	C	-	-
Seq6	-	-	C	-	-
Consensus			C	D	E/A

Figure 6: **Calculation of the Consensus Strand.** This Image shows how the consensus was calculated in our experiments. (1) In the first column from the left, the letter need to be present at least in 30% of the sequences, the A is just present in 1/6 of the sequences. 1/6 <= 30% hence it does not occur in the consensus. (2) In the second column if just dashes are present, nothing occurs in the consensus. (3) In the third and fourth column the letter occured in the consensus strand because it appeared in at least 30% of the sequences. (4) The last column depicts a very rare case. If two letter occur in the same amount and each one in at least 30% of the sequences, both letters appear in the consensus strand with a dash between them.

4.2.4 Grid Search for the optimal parameters for the alignment

To compare the different parameters and matrices, we aligned sequences (of a particular length range) with all parameter-matrix combinations. The input parameters of the function were the sequence length range, so start length and end length, a list of gap open penalties, a list

of gap extension penalties, a list of matrices, and a list containing both tree options. Multiple sequence alignments works best if the sequences have the same or at least similar length, therefore we split the large FASTA file in multiple small ones. First, the grid search function created a text file, holding all sequences of actions within a particular range. Afterwards, the sequences were aligned with all matrix-gap penalty-tree combinations, which were inserted into the function. The results were saved in a DataFrame. Every outcome of a combination was saved as

Offdiag	GO	GE	Tree	Score	Total length	Minimum Length	Nr. Sequences	Alignment Length	Consensus Length
-1	1	0	NJ	23496	7730	1217	7	4233	2336
-1	0.1	0	NJ	27777	7730	1217	7	4324	2142
-10	0.1	0	NJ	29416	7730	1217	7	4425	1939
-15	0.1	0	NJ	29416	7730	1217	7	4425	1929
-100	0.1	0	NJ	29416	7730	1217	7	4425	1939
-5	0.1	0	NJ	30212	7730	1217	7	4285	1930
-5	1	0	NJ	21240	7730	1217	7	4998	1815
-15	1	0	NJ	15395	7730	1217	7	5862	1269
-100	1	0	NJ	15321	7730	1217	7	5889	1240
-10	1	0	NJ	14069	7730	1217	7	6058	1120

Table 1: **Alignment results of sequences with 1000 to 1400 residues.** All the sequences with length of 1000 to 14000 residues are aligned with all parameter-matrix combinations and just with NJ. The figure shows the top ten results sorted by the length of the Consensus (descending), the Alignment Length(ascending), and the Score(descending).

one row in the table. The columns represented the different metrics that we chose to compare the sequences: the score, given by the ClustalW output, the consensus strand, and Length, and the alignment length. These four values are the most important indicators for proper alignment. The columns 'total length' and 'minimum length' are the boundaries for the alignment length. For better evaluation, we sorted the Dataframe by the length of the Consensus, the Alignment Length, and the Score. The Score and the Consensus were ordered descendingly and the Alignment Length ascendingly. The best parameters concerning our chosen metrics are at the top of the table. An example output is indicated in figure 1. With help of this function we could easily compare all parameter.

4.3 Optimal parameter search for RL agent in AlignRUDDER

In the second experiment, we searched for optimal alignment parameters for an RL algorithm whose prior knowledge is based on a multiple alignment, and we observed how the agent reacted to qualitatively different alignments. As with Minecraft sequences, the aligned sequences are not sequences of biological origin. Here, various step chains that the agent performs have been aligned. The reinforcement agent does not play a complex game like Minecraft but in a simple gridworld environment [23]. Thus, possible effect of the parameters could be better observed. Our experiments are based on the AlignRUDDER [23] program.

4.3.1 AlignRUDDER

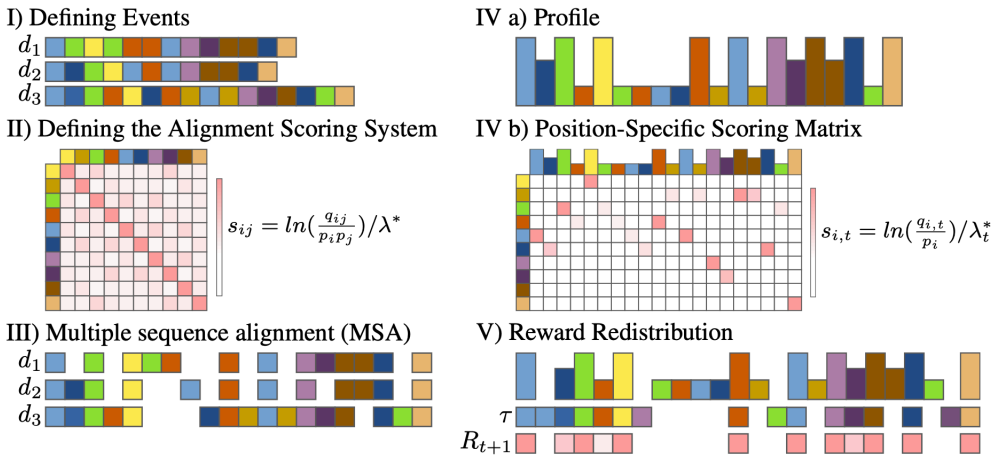


Figure 7: **MSA implementation in AlignRUDDER.** The figure shows the five steps of the MSA implementation in AlignRUDDER. The figure originates form the paper [23]

AlginRUDDER [23] is a reinforcement algorithm that is able to learn from only a few demonstrations. RL is based on an imitation of human learning. Humans learn by trial and error but also have role models, such as parents and teachers, from whom they learn. Normal RL usually needs many runs to come to a result, because everything has to be learned first. Learning happens by random action selection and the rewards come rarely and by chance. This is the basis of the idea to provide the RL with knowledge, in form of already completed solution attempts. This procedure was used in the RUDDER. But the LSTM in RUDDER, which is used for the reward redistribution, needs many demonstrations. In many cases demonstrations are rare and costly. AlignRUDDER solves this problem by replacing the demonstration-intensive

LSTM with a multiple sequence alignment. It was assumed that successful game sequences with the same goal/intermediate goal would have certain similarities. These similarities are then the key points to achieve the goal. Multiple sequence alignment (MSA) is used to find these matches. MSA algorithms are specialized for this task. This serves to find important events in the sequences [23].

The rewards redistributions works with five steps 7 (1) The first step is to transform the demonstrations into a sequential form so that the MSA can process them. To do this, events are specified, which consist of state-action pairs. Events can be, for example, receiving a reward, picking up an object or ect. Different state-action pairs can lead to the same event. The events must then be ordered in the sequence so that the strategy remains recognisable. (2) The second step is to establish an appropriate scoring system. This scoring matrix is needed for the MSA, as standardised tools like BLOSUM or PAM are not optimal for the non-biological sequences. We used a very similar variant as in this Paper (see [23]). (3) The third step is the MSA. The sequences are aligned optimally to the scoring system. (4) The fourth step is to create the consensus strand and calculate a Position specific scoring matrix PSSM (details [23]). (5) In the fifth step the score is redistributed[23].

In order to calculate the scoring matrix for the sequence alignment, we used the same algorithm as for the Minecraft experiment. This algorithm was only very slightly different from the one already implemented in AlignRUDDER, so we only had to make little changes. Subsequently, we used TuneSearch [24] to iterate over the different parameters.

5 Results and Discussion

In the following we present and discuss the results of the optimal parameter search for the Minecraft sequences and for the AlignRUDDER program.

5.1 Results for the optimal parameter search for the Minecraft sequences

In this subsection we describe our results of our search for finding the optimal alignment of the Minecraft sequences. As indicators for a good alignment we focused on the alignment length, the consensus strand, and the score.

5.1.1 Parameter selection for the Minecraft sequences

For the alignment the ClustalW was used. The alignment tool offers many additional features, that were not needed to align the Minecraft sequences. The algorithm was developed to align nucleotide and amino acid sequences optimally. Therefore, there are multiple features to improve the particular alignment. For instance, one can choose to utilize the secondary structure, hydrophilic gaps, or specific trees for the alignment[5]. As mentioned previously, the sequences in our experiments have not had any biological meaning. Consequently, we did not use amino acid-specific options wherever possible. Otherwise, those options would interpret the sequences wrongly. The table 2 shows which features we used and which we switched off. To investigate which parameters are best suited to align Minecraft sequences, we compared different parameters with our grid search algorithm. We used different matrices with off diagonal values of -1, -2 to -15 and -100. For the gap open penalty we utilized the values 1 and 0.1 and for the gap extension penalty 0 and 1. Additionally, we tried two different guide trees neighbor joining (*NJ*)and Unweighted Pair Group Method with Arithmetic mean (*UPGMA*). There are infinite Matrix and Gap penalty options. We decided to use these combinations because rough tests in advance showed that values in this range are suitable for obtaining a proper alignment and by means of normally utilized parameters for biological sequence alignment. Furthermore, it should be said that the number of possible combinations increased with every additional option.

The reason for picking these matrices is that in our previous assumptions, the most suitable scoring matrix will be among 5 and 15 as a mismatch penalty. We added matrix -1 and -100 as 'control' matrices. Both matrices are extreme instances. Matrix -1 resulted in many mismatches because of the low mismatch penalty. Matrix -100 had fewer mismatches but many gaps.

Gap open penalties describe the cost of opening a gap within the alignment. The choice of Gap open penalty relied on a rule of thumb: The Gap Open Penalty should be the maximal value of the scoring matrix, divided by ten or 100. Between ten and 100 is a huge difference, therefore we tried both numbers to observe the behavior of the alignment. For all matrices the highest match value is ten, therefore we got the gap open penalties 0.1 and 1. If gaps are extend the cost (usually smaller than the gap open penalty) given by the gap extension penalty

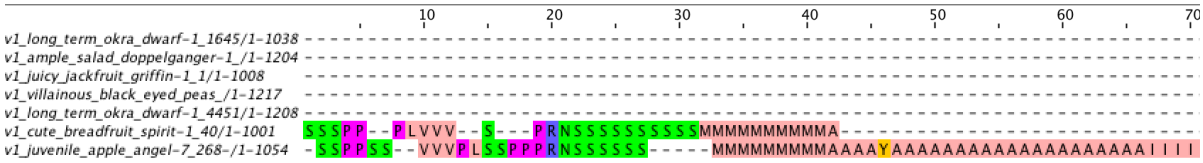


Figure 8: **Alignment of sequences with GO: 0.1, GE: 0.5** The image illustrates the alignment of sequences of length 1000 to 1400 with Gap Open Penalty of 0.1, Gap Extension Penalty of 0.5 and NJ. The figure shows the first 70 residues. JalView was used to visualize the alignment

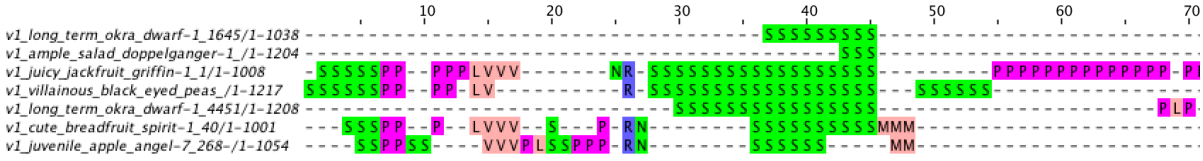


Figure 9: **Alignment of sequences with GO: 0.1, GE: 0.** The image illustrates the alignment of sequences of length 1000 to 1400 with Gap Open Penalty of 0.1, Gap Extension Penalty of 0 and NJ. The figure shows the first 70 residues. JalView was used to visualize the alignment

is used. For the gap extension penalty, we tried randomly different values and observed that the penalty of one and zero resulted in the best outcome. Moreover, we checked the values in the range of zero to one. The alignment with a gap extension penalty of 0.1 to 0.9 worked, the outcomes were good. But as figure 8 and figure 9 depict, the alignment with gap extension of zero contains more matches and therefore seems to be proper. We observed similar results with other gap extension penalties within zero and one. Hence, we did not use them for closer analysis.

5.1.2 The grid search

For the comparision of the different alignments we used our grid search algorithm. The grid search algorithm outputs a dataframe. It contains one column for each of our metrics. An example of such a result can be seen in table 3. For the optimal alignment of the Minecraft sequences we focused on the alignment length, the consensus strand, and the score as indicators of a good alignment. The 'Total length' column holds the sum of all sequences lengths within the used file. This value is the longest possible alignment that results from all sequences being placed after each other in the alignment. Figure 10 depicts a cutout of an alignment where alignment length and total length were equal. The 'minimum length' is the length of the longest

gapopen	0.1 or 1
gapext	0 or 1
matrix	One of the five matrices
tree	NJ or UPGMA
gapdist	0 - that is the gap separation pen range
pwgapopen	the same as the gap open penatly
pwgapext	the same as the gap extension penatly
endgaps	True - No end gap separation pen is used
noweights	True - The sequences are not weighted
align	True - results in full multiple alignment
nohgap	True - No use of Hydrophilic gaps
type	”PROTEIN” - specify that amino acids are aligned
negative	True - otherwise negative values in the scoring matrix are not allowed
seqnos	”ON”
nosecstr1	True - No secondary structure-gap penalty mask for profile 1 is used.
nosecstr2	True - No secondary structure-gap penalty mask for profile 2 is used.
maxdiv	0 - Percent identity for delay

Table 2: **ClustalW features.** The table of features ClustalW offers that we used for the alignment or switched off. The detailed meaning of all the features can be seen [5].

sequence in the file and therefore represents the shortest possible alignment. The two values could not be used to judge the properness of the alignment, but they served to understand whether the alignment is rather long or short.

With the grid search function, we aligned all sequences with the five matrices (with non-diagonal values of -1,-5,-10,-15,-100) and the different parameters. We noticed by looking at the resulting table (figure 3) that two equal rows occurred always after each other. At these rows, all metrics had the same values. The identical rows were aligned with the same matrix, the same parameters, just the guiding trees were distinct. Hence, the choice of NJ or UPGMA did not alternate the result. We observed the equivalent results with all sequences, for all length varieties we used. For confirmation, we used Python to compare the consensus

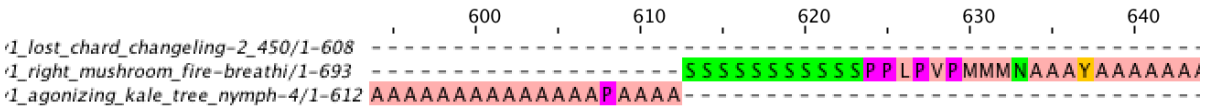


Figure 10: **Alignment of sequences with 600 to 700 residues with GO: 1 and GE: 1** All the sequences with length of 600 to 700 residues are aligned with Gap Open Penalty(GO) of 1, Gap Extension Penalty(GE) of 1, Matrix -5 and NJ as tree method. The Image is just a small part of the complete alignment around the residue 600. Nevertheless, it is visible that the sequences are not aligned, but placed after each other.

Offdiag	GO	GE	Tree	Score	Total length	Minimum Length	Nr. Sequences	Alignment Length	Consensus Length
-1	0.1	0	NJ	47024	3827	196	25	1071	55
-1	0.1	0	UPGMA	47024	3827	196	25	1071	55
-100	0.1	0	NJ	46370	3827	196	25	1606	53
-100	0.1	0	UPGMA	46370	3827	196	25	1606	53
-15	0.1	0	NJ	47099	3827	196	25	1565	52
-15	0.1	0	NJ	47099	3827	196	25	1565	52
-5	0.1	0	NJ	44999	3827	196	25	1400	46
-5	0.1	0	NJ	44999	3827	196	25	1400	46

Table 3: **Top ten of alignments results of sequences in the range of 0 to 300.** The table shows the top ten results of the alignment sorted by the length of the Consensus (descending), the Alignment Length(ascending), and the Score(descending). All sequences were aligned with all matrix-parameter combinations and the two different trees. The table shows that the choice of NJ or UPGMA did not change the result. GO - Gap open penalty, GE - gap extension penalty.

strands and detected that the consensus strands were equal. Furthermore, we used JalView to compare randomly chosen sequences sets, but no difference appeared. Therefore we decided to use only NJ for the next alignments, because the Neigbour Joining method is more stable than UPGMA [30].

In the next step we searched for the best gap penalties at first and afterwards adjusted the matrices. The results of the alignment with only neighbour joining as guiding tree are shown by figures 4 and 5. It can be seen that the top five alignments are always aligned with the gap open penalty of 0.1 and a gap extension penalty of 0.0. The same appeared with all sequences regardless of the sequence length and the size of the set of sequences. The order of the matrices changes, but it can be seen that the choice of gap penalty 0.1 and 0.0 provided the best result.

Offdiag	GO	GE	Tree	Score	Total length	Minimum Length	Nr. Sequences	Alignment Length	Consensus Length
-4	0.1	0	NJ	373152	14529	376	78	2410	37
-9	0.1	0	NJ	363177	14529	376	78	3782	36
-5	0.1	0	NJ	360124	14529	376	78	2889	33
-11	0.1	0	NJ	317235	14529	376	78	3480	33
-8	0.1	0	NJ	349125	14529	376	78	3635	32
-6	0.1	0	NJ	330573	14529	376	78	2886	31
-2	0.1	0	NJ	361015	14529	376	78	1859	30
-3	0.1	0	NJ	350470	14529	376	78	2919	30
-7	0.1	0	NJ	324068	14529	376	78	3181	27
-13	0.1	0	NJ	313374	14529	376	78	3737	26

Table 4: **Top ten of alignments of sequences in the range of 0 to 400.** The table shows the first ten results of the alignments sorted by the length of the Consensus (descending), the Alignment Length(ascending), and the Score(descending). All the Alignments were performed with NJ tree option.

Offdiag	GO	GE	Tree	Score	Total length	Minimum Length	Nr. Sequences	Alignment Length	Consensus Length
-5	0.1	0	NJ	53201	10742	1577	9	5355	1437
-1	0.1	0	NJ	50499	10742	1577	9	5329	1355
-10	0.1	0	NJ	49118	10742	1577	9	5593	1126
-15	0.1	0	NJ	49057	10742	1577	9	5582	1101
-100	0.1	0	NJ	49057	10742	1577	9	5582	1101
-1	1	0	NJ	36847	10742	1577	9	5649	960
-5	1	0	NJ	32919	10742	1577	9	6470	775
-10	1	0	NJ	25923	10742	1577	9	7544	480
-100	1	0	NJ	23335	10742	1577	9	7943	349
-15	1	0	NJ	23597	10742	1577	9	7784	319

Table 5: **Result of sequences in range 1000 to 1600 aligned with NJ.** The table shows the result of the first ten results of the alignment of the sequences of length 1000 to 1600. All alignments were performed with the NJ tree method. The table is sorted by the length of the Consensus (descending), the Alignment Length(ascending), and the Score(descending)

Offdiag	GO	GE	Tree	Score	Total length	Minimum Length	Nr. Sequences	Alignment Length	Consensus Length
-4	0.1	0	NJ	1168859	44437	1577	117	6326	40
-10	0.1	0	NJ	1034068	44437	1577	117	7599	33
-2	0.1	0	NJ	1215203	44437	1577	117	5334	30
-9	0.1	0	NJ	1008308	44437	1577	117	7459	30
-3	0.1	0	NJ	1090512	44437	1577	117	5745	25
-11	0.1	0	NJ	918044	44437	1577	117	6797	25
-5	0.1	0	NJ	1071684	44437	1577	117	6457	24
-15	0.1	0	NJ	985259	44437	1577	117	7892	23
-8	0.1	0	NJ	967089	44437	1577	117	6791	22
-14	0.1	0	NJ	915262	44437	1577	117	6877	22

Table 6: **Alignment of all sequences with GO: 0.1, GE:0.** The table shows the top ten results of the alignment sorted by the length of the Consensus (descending), the Alignment Length(ascending), and the Score(descending). All sequences were aligned with gap open penalty of 0.1 and gap extension of 0, NJ and all matrices from 2 to 15.

The outcomes had the longest consensus strand, the shortest alignment and the highest score. Therefore the preferred penalties for the Minecraft sequences are the gap open penalty of 0.1 and the gap extension of 0.0.

Subsequently, we searched for the best scoring scheme. Therefore, we used all matrices for the alignment and with the gap open penalty of 0.2 and the gap extension of 0.0. The scoring matrices did not alter the alignment as strong as the gap penalties. The outcomes of the matrices were similar (table 4, table 5). We expected that the different scoring matrices would vary more than they did.

Even outcomes of matrix -100 and matrix -1 differed little from the other results. The difference between a mismatch penalty of -1 and -100 is large, nevertheless, the difference is small within the columns. We chose matrix -100 and matrix -1 to observe the performance of ClustalW with an extreme mismatch penalty. But the extreme matrices did not result in extreme results. In some cases, the alignments of matrix 100, as well as matrix -1, were the best resulting alignment. However, we have never intended to use these two extremes. Matrix -100 allows ClustalW almost no mismatches. Opening a gap is cheaper for the algorithm than tolerating mismatches. Notwithstanding, in some cases, it is better to allow a mismatch than to open a gap. Controversy, matrix -1 leads to many mismatches but fewer gaps.

Offdiag	GO	GE	Tree	Score	Total length	Minimum Length	Nr. Sequences	Alignment Length	Consensus Length
-4	0.1	0	NJ	373152	14529	376	78	2410	37
-9	0.1	0	NJ	363177	14529	376	78	3782	36
-5	0.1	0	NJ	360124	14529	376	78	2889	33
-11	0.1	0	NJ	317235	14529	376	78	3480	33
-8	0.1	0	NJ	349125	14529	376	78	3635	32
-6	0.1	0	NJ	330573	14529	376	78	2886	31
-2	0.1	0	NJ	361015	14529	376	78	1859	30
-3	0.1	0	NJ	350470	14529	376	78	2919	30
-7	0.1	0	NJ	324068	14529	376	78	3181	27
-13	0.1	0	NJ	313374	14529	376	78	3737	26

Table 7: **Alignment of Sequences smaller than 400 residues with GO: 0.1, GE:0.** The table shows the top ten results of the alignments sorted by the length of the Consensus (descending), the Alignment Length(ascending), and the Score(descending). All Sequences smaller than 400, were aligned with gap open penalty of 0.1 and gap extension of 0, NJ and all matrices from 2 to 15.

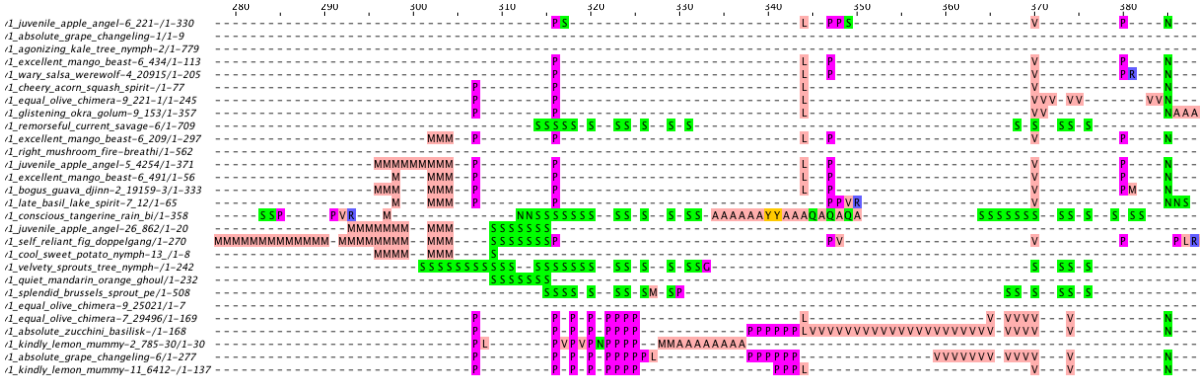
Offdiag	GO	GE	Tree	Score	Total length	Minimum Length	Nr. Sequences	Alignment Length	Consensus Length
-8	0.1	0	NJ	18782	4517	593	8	2640	455
-9	0.1	0	NJ	18782	4517	593	8	2640	455
-10	0.1	0	NJ	18782	4517	593	8	2640	455
-15	0.1	0	NJ	18782	4517	593	8	2640	455
-11	0.1	0	NJ	18782	4517	593	8	2640	455
-12	0.1	0	NJ	18782	4517	593	8	2640	455
-13	0.1	0	NJ	18782	4517	593	8	2640	455
-14	0.1	0	NJ	18782	4517	593	8	2640	455
-5	0.1	0	NJ	17301	4517	593	8	2672	429
-6	0.1	0	NJ	17315	4517	593	8	2685	427

Table 8: **Alignment of Sequences between 500 and 600 residues with GO: 0.1, GE:0.** The table shows the top ten results of the alignments sorted by the length of the Consensus (descending), the Alignment Length(ascending), and the Score(descending). The sequences with length in range 500 to 600 residues, were aligned with gap open penalty of 0.1 and gap extension of 0, NJ and all matrices from 2 to 15.

We aligned the sequences sets another time with all matrices from two to 15, gap open penalty of 0.1, gap extension of 0 and NJ as tree. The tables 6 and 7 demonstrate the effects of these schemes. The order of the matrices changed again for every set. The problem with the scoring schemes is that for similar sequences, the scoring matrix has little influence. The sequences already have very conserved regions that are recognized by every model. ”When identities dominate an alignment, almost any weight matrix will find approximately the correct solution” [30]. That was the case as we aligned the sequences of similar length like the sequences in range 500 to 600 (figure 8). The consensus strands are similar.

For rather distinct sequences it was difficult to find an optimal scoring model. Different matrices are optimal for disparate sequences. It is impossible to find a general scoring matrix that works with all the Minecraft sequences. If one wants to align each sequence sets optimally, different scoring schemes are needed. The optimal matrix always depends on the sequences themselves, this can not be known beforehand. Hence, we would recommend using an intermediate matrix e.g. matrix 10. The problem with the smaller matrices is that these allow many mismatches, an example is shown in figure 11.

Figure 11: **Sequences less than 400 residues GO: 0.1, GE: 0, matrix -4.** All Sequences smaller than 400, aligned with a gap open penalty of 0.1 and gap extension of 0, NJ and matrix 4. JalView was used to produce this visual alignment.



5.2 Results for the optimal parameters search for AlignRUDDER

In the following subsection, we present the results for our parameter search for Align-RUDDER, a RL algorithm which is initialized based on a multiple alignment. We searched for optimal alignment parameters by observing how the agent reacted to qualitatively different alignments. This was done by adjusting the following parameters: the gap open penalty, the gap extension penalty and the scoring matrix.

Each run was started at a random initial starting point within the eight rooms. Due to that, we used ten different seeds. Moreover, we used four different numbers of demonstrations to train the agent: 2, 10, 50 and 100. For this subsection, the number of episodes needed to perform the task was our main metric to analyse the runs. Due to limited computational resources, we concentrated on the eight room rudder.

Table 9 contains all parameters which we used for the grid search. The majority of the values

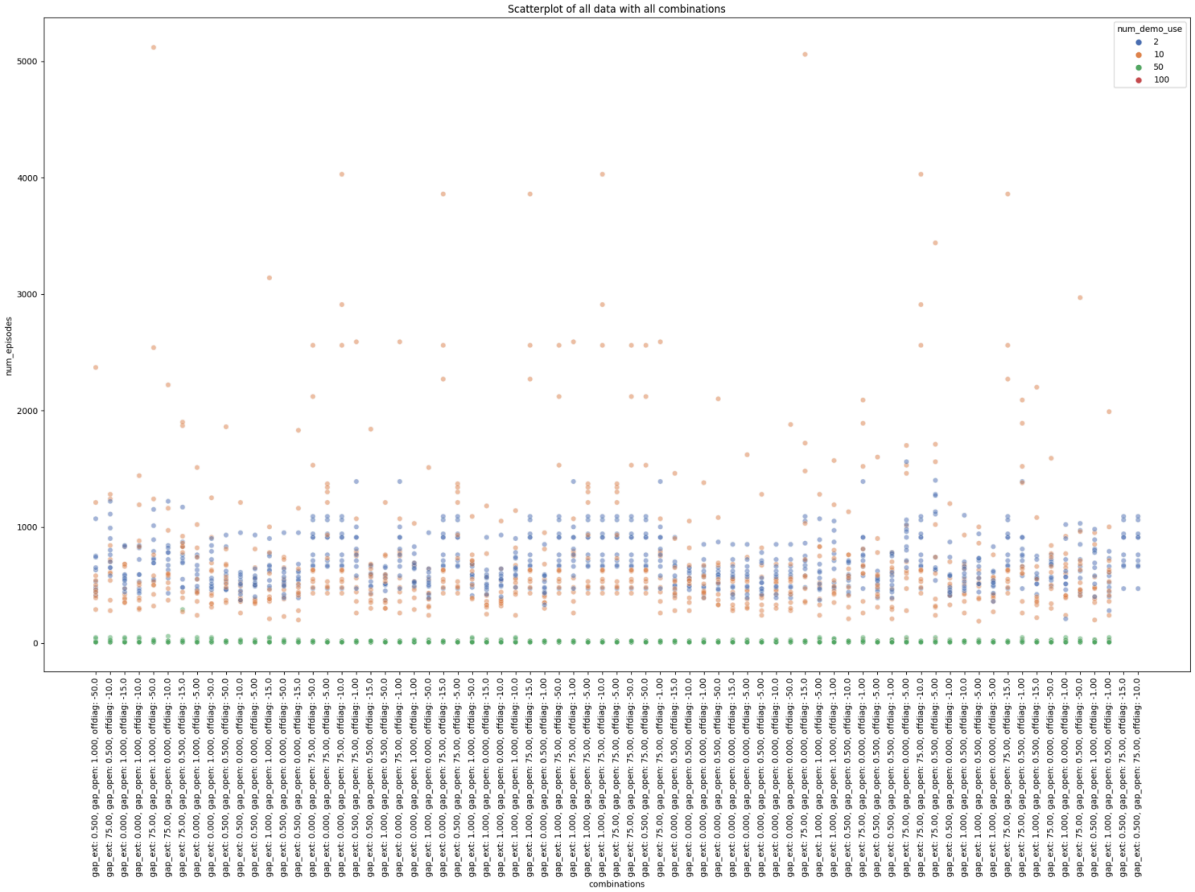
Parameters	Values
seed	2, 10, 50 and 100.
number of demos	2,10,50,100
gap open penalty	0, 0.5, 1, 75
gap extension penalty	0, 0.5, 1, 75
matrix off diagonal values	-1, -5, -10, -15, -50

Table 9: **RUDDER grid search parameter.** Parameters we used for the grid search for RUDDER

we used seemed reasonable to us and we expected good results. Additionally, we used some more extreme parameters like a gap open penalty of 75. For these extreme parameter we knew from previous experiments that for rather distinct sequences it will lead to a weaker alignment. Therefore, we wanted to check how the agents behavior changes with worse alignments.

Figure 12 shows the results of all combinations of the parameters (table 9) against the number of episodes the agent needed to solve the task. We have chosen to include figure 12 as it shows all available data and gives a good introduction to our results. It is already clearly visible that the number of demonstrations play a major role in the number of episodes, as more than 50

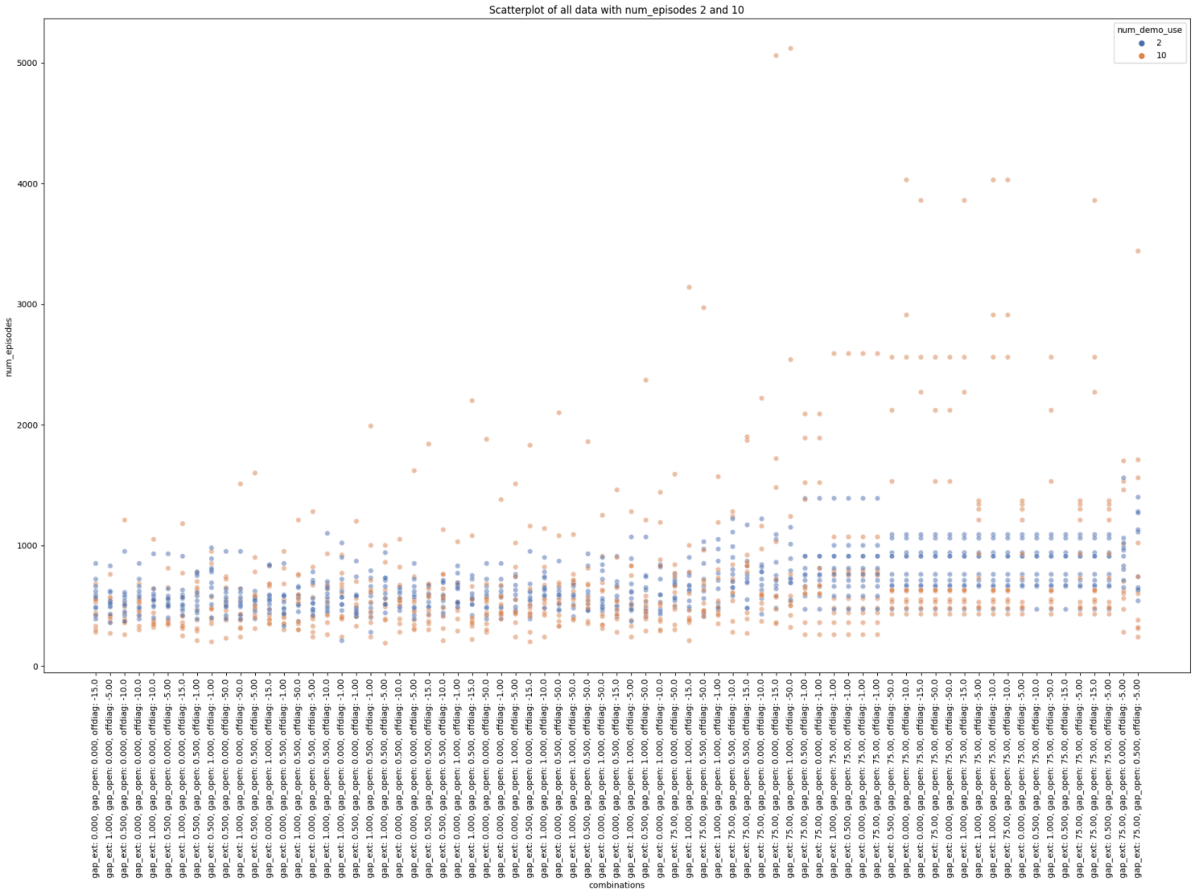
Figure 12: **Scatterplot of all the data.** The parameter-combinations are plotted against the number of episodes. For each combination, ten seeds and four different amounts of demonstrations were used. The figure illustrates the influence of the number of demonstrations to the performance of the RL agent. All rounds with more than 50 demonstrations completed the task within about 10 episodes.



demonstrations always lead to a fast success within about 10 episodes, even for more extreme parameters. Fewer demonstrations required more episodes. Other parameters do not influence the results as much. This can also be observed in figure 15, which shows only some combinations. We found the same trend in the other combinations as well, which are also plotted and can be found in section 7 (Appendix). Based on this trend, we deduced that when enough demonstrations are used, the other parameters are not as important anymore. However, Align-RUDDER has the purpose of solving tasks efficiently even when very few demonstrations are given [23] and therefore we focused on data with two and ten demonstrations.

The results filtered for only 2 and 10 demonstrations are again visualized in a scatterplot

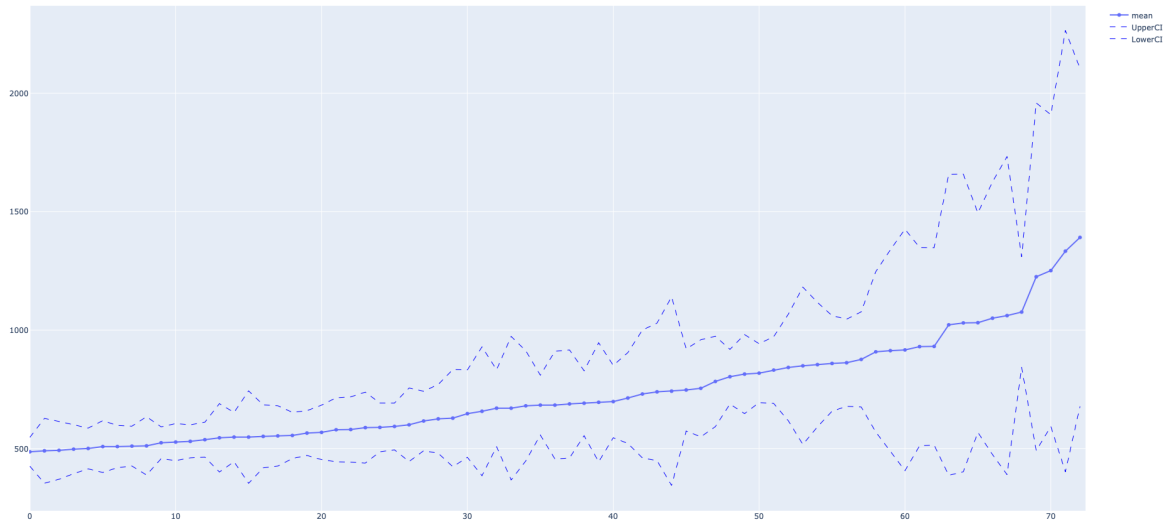
Figure 13: **Scatterplot of demonstration 2 and 10** The parameter-combinations are plotted against the need number of episodes. For each parameter combination, the ten seeds and only two number of demos (2 and 10) are shown.



(figure 13). The combinations in this plot are sorted by the mean number of episodes needed to solve the task, increasing from left to right. As this is still not very intuitive to interpret, figure 14 shows only the mean and confidence interval of each combination. The first ten parameter combinations can also be seen in table 10. One trend we observed is that the more extreme values such as a gap open penalty of 75.0 consistently achieved the highest number of episodes. The same trend was already visible in the figures 15, 18, 19 and 20.

Nevertheless, apart from the aforementioned trend we were not able to make more conclusion by just visually inspecting the data. Additionally, we were not even sure if there is any significant difference between any of the combination as the confidence intervals in figure 14 overlap, even for the worst and best performing combinations. Therefore, we carried out an ANOVA

Figure 14: **Lineplot of all the data.** The x-axis represents the different configurations, each point stands for one configuration. For each parameter combination, the ten seeds and two number of demos (2 and 10) are shown. The seeds and demos are combined by calculating the mean of them. Around the the line the 95% confidence interval is shown.



analysis. For this we assumed normally distributed data. The hypothesis were

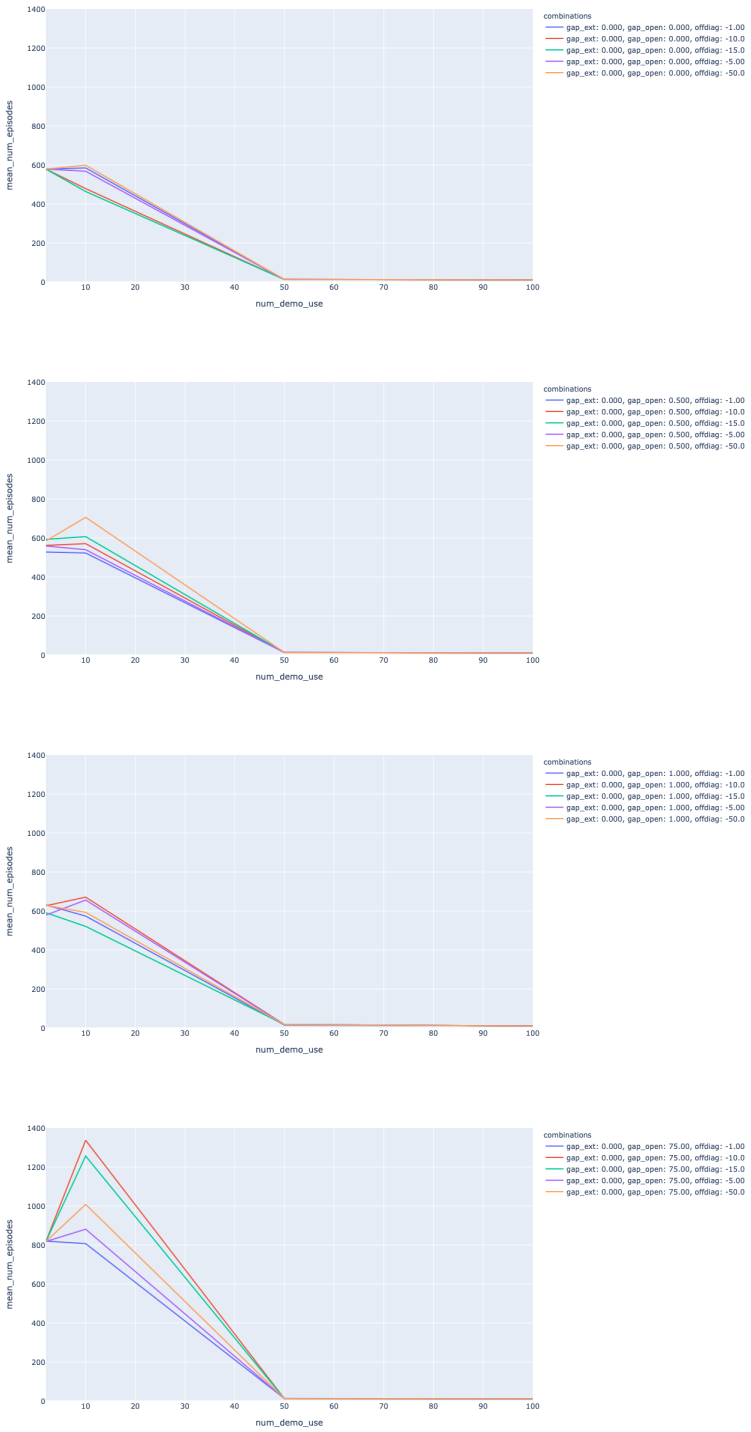
$$H_0 = \mu_1 = \mu_2 = ... = \mu_m H_1 = \text{the means are unequal}$$

m is the number of groups compared by ANOVA

Our null hypothesis for all subsequent ANOVA tests was that all means are equal. This hypothesis was rejected if p-value ≤ 0.05. First we tested the significant difference of all our data (as in figure 12). The ANOVA test over all four numbers of demonstrations used shows that the mean values differ statistically significant with a p-value of 0.004. This confirms our previous assumption that the number of demonstrations used have a high influence on the outcome.

However, as already mentioned previously, we decided to focus more on the lower number of demonstrations. Therefore, we performed an ANOVA analysis for all combinations grouped by the number of demonstrations. The results can be seen in table 11. These show that within a high number of demos, here 50 and 100, the values do not differ statistically significantly. If the RL algorithm is given enough demos, the parameters hardly play a role and the results

Figure 15: In these figures the number of episodes needed was plotted against the number of demonstrations. Each line represents one parameter configuration. The number of episodes value is the average over the ten seeds. In each sub figure only the matrix off-diagonal value differed. And from every sub figure to sub figure the gap open penalty changes. The axis are fixated.



GE	GO	Offdiag	Mean Nr. Episodes	Median Nr. Episodes
0.5	0.0	-10.0	486	505.0
0.5	0.0	-15.0	490	435.0
0.5	0.0	-5.0	492	390.0
1.0	0.0	-15.0	497	470.0
1.0	0.0	-5.0	500	495.0
0.0	0.5	-1.0	508	485.0
1.0	0.5	-50.0	508	505.0
0.5	0.5	-5.0	510	510.0
1.0	0.0	-50.0	511	465.0
0.5	0.0	-50.0	524	515.0

Table 10: **Ten best results** This table shows the first ten results of the lineplot 14. There is no statistical significant difference in-between them, but they are significantly different from the runs on the right side of the plot. These ten combinations performed very good on the task even that only two or ten demonstrations were given.

Num demo use	P-Value	F-Value	Statistical significant difference
2	0.00000	4.36669	True
10	0.00236	1.59269	True
50	0.50811	0.98793	False
100	0.99777	0.56923	False

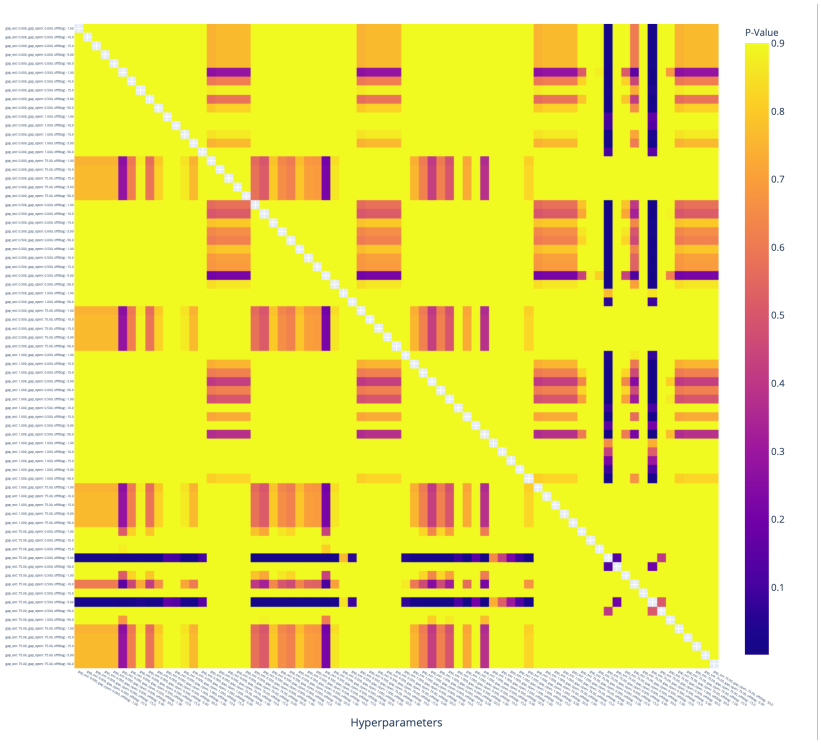
Table 11: **ANOVA test results.** The results of the ANOVA tests of the different demonstrations used.

barely differ.

Furthermore, we performed a two-pair independent t-test between the best value (gap_ext: 0.500, gap_open: 0.000, offdiag: -10.0) and the worst (gap_ext: 0.000, gap_open: 75.00, offdiag: -15.0). The difference between these two values is significant with a p-value of 0.02. and a test statistic of -2.5.

For the two groups with statistical significance (demos 2 and 10) we also carried out a Tukey

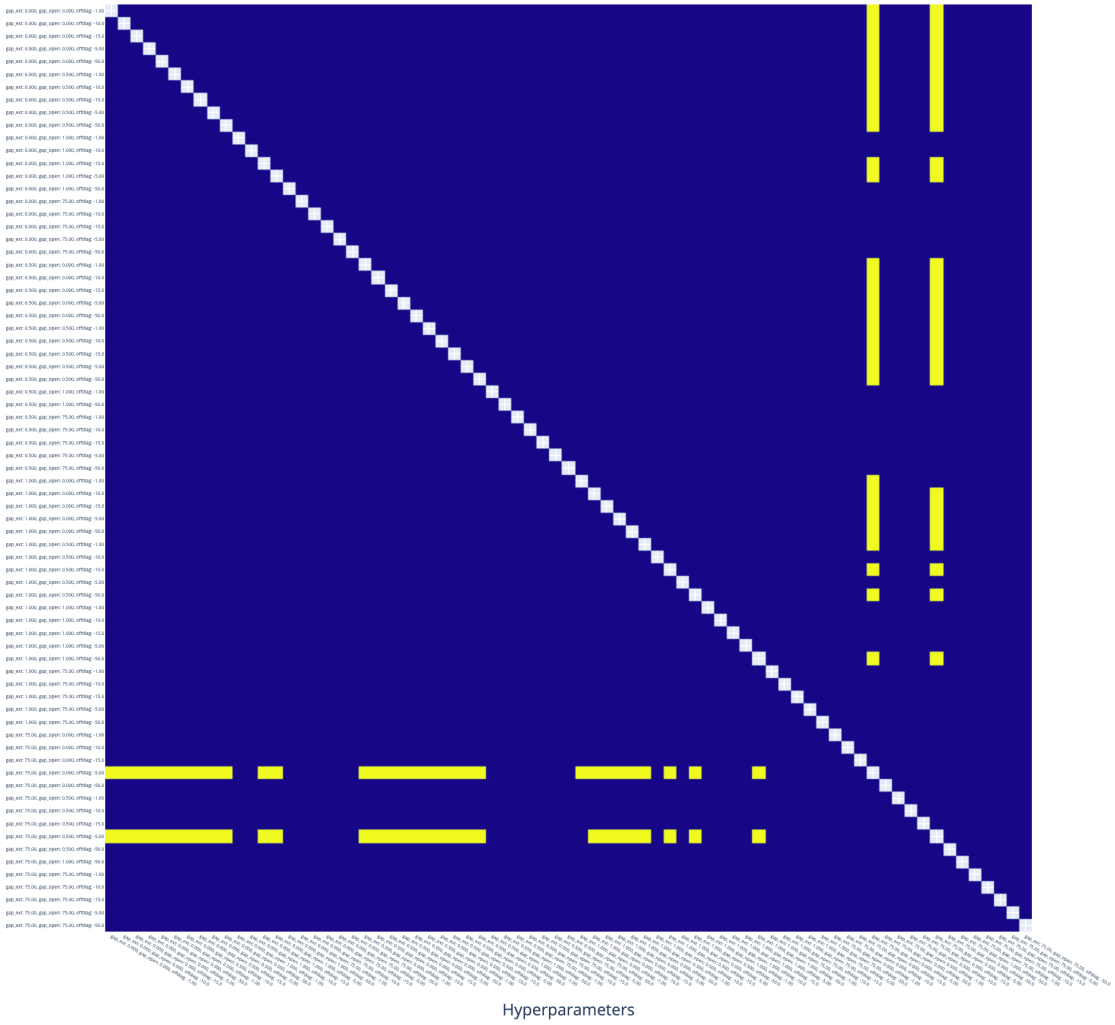
Figure 16: **Heatmap of Tukey Post Hoc Test.** The figure shows the results of the Tukey Post Hoc test as a heatmap colored by the p-value. Null hypothesis was that the means are equal. This can be rejected with a p-value smaller or equal to 0.05, visible in this graph by the color dark purple.



post hoc test, in order identify the better/worse parameter configurations. While ANOVA found at least one significant difference in both datasets (see table 11), the more sensitive Tukey test showed that the ten demo dataset had no statistically significant differences. Differences could be determined for the other dataset. Figure 16 and figure 17 show the result of the post hoc analysis. This shows that there is always a significant difference between the runs with parameter value 75 for gap open or extension and the others. This confirms the observation from figure 14 that the values depicted on the left are significantly different from those on the right. However, we found no significant difference between the other parameter combinations.

Our analysis showed that for runs where 50 and more demonstrations were given, the parameters did not influence the results. For the runs with fewer demonstration, it could be seen that the gap penalties have the most influence. Too high gap penalties (gap open penalty: 75.0,

Figure 17: **Rejected Heatmap of Tukey Post Hoc Test.** The figure shows the results of the Tukey Post Hoc test as a heatmap showing only rejected and non-rejected. Null hypothesis was that the means are equal. This can be rejected with a p-value smaller or equal to 0.05. The data includes only boolean values. The null hypthesis is rejected by points in color yellow



gap extention penalty: 75.0) take more episodes. The scoring matrices did not have a high influence. We recommend the use of a intermediate scoring scheme (-5 to -15 non-diagonal values) and gap penalties between zero and one.

6 Conclusion

The goal of this work was to find the optimal alignment parameters for Minecraft sequences and for the AlinRUDDER algorithm. For that we have carried out two experiments. For both experiments we found parameters that improved the results. However, no clear optimal parameters could be found. We can only give recommendations, because results were statistically insignificant. First, we summarize the parameter search for the alignment of the Minecraft sequences. Secondly, we sum up the parameter search for AlignRUDDER.

6.1 Parameter Search for the Minecraft Task

To conclude the hyperparameter search for optimal alignment of the Minecraft sequences, our investigation demonstrated that gap penalties and matrices affect alignments differently. The choice of suitable gap penalties is essential. Our results indicate that small variations influence the outcome drastically. A gap open penalty of 0.1 and a gap extension penalty of 0.0 performed best. We would suggest using a matrix in a range of -5 to -15 as a scoring scheme. Because no matter which matrix we used, all results appeared comparable. Matrices below five will result in many mismatches, and models with high mismatch penalties will end in many gaps. The range of -5 to -15 is ideal to keep the balance between gaps and mismatches. Moreover, the optimal matrix depends on the sequences themselves. Better parameters could be found if it was possible to get more information about the sequences. The best way would be to identify the sequences that describe the same task and align these tasks separately. With the help of these alignments, a new scoring scheme could be built. A mutability matrix similar to PAM could be created. Nonetheless, we showed that the scoring matrix had little influence on the alignment. Hence, we conclude that it is not always worth the effort to create a dedicated scoring matrix for every task. The parameters represented in this paper will help to find the similarities between the sequences.

6.2 Parameter Search for the AlignRUDDER Task

The concrete parameter search for the Minecraft sequences as well as the parameter testing for the eight room test illustrated that parameters are important. However, in some circumstances the influence of them is hard to observe. We saw that an alignment with ClustalW is sensible to parameters and that slight changes influenced the alignment. Our analysis with the same parameters on the eight-room example showed that a slightly better or worse alignment did not really affect the result as long as enough demonstrations were used. Even with few demos or extreme parameters (e.g. matrix -50, GE: 75.0 or GO: 75.0), the RL algorithm performed well. Nevertheless, smaller gap penalties in a range from zero to one are recommended. Additionally, we advise an intermediate scoring scheme (-5 to -15 non-diagonal values). Since no statistically significant difference could be found between these parameters, we cannot give one optimal configuration of parameters. To receive an optimal scoring scheme, the sequences would need to be analysed in more detail.

List of Figures

1 Example of an MSA performed with ClustalW 5
2 Multiplicand table for amino acids 7
3 Four Steps of ClustalW Alignment 8
4 PAM Numbers 10
5 Scoring Matrix with -1 non-diagonal value 16
6 Calculation of the Consensus Strand 17
7 MSA implementation in AlignRUDDER 19
8 Alignment of sequences of length 1000 to 1400 22
9 Alignment of sequences with GO: 0.1, GE: 0 22
10 Alignment of sequences with 600 to 700 residues with GO: 1 and GE: 1 . . . 24
11 Sequences less than 400 residues GO: 0.1, GE: 0, matrix -4 28
12 Scatterplot of all data 30
13 Scatterplot of demonstration 2 and 10 31
14 Lineplot of all data 32
15 In these figures the number of episodes needed was plotted against the number of demonstrations. Each line represents one parameter configuration. The number of episodes value is the average over the ten seeds. In each sub figure only the matrix off-diagonal value differed. And from every sub figure to sub figure the gap open penalty changes. The axis are fixated. 33
16 Heatmap of Tukey Post Hoc test 35
17 Rejected Heatmap of Tukey Post Hoc Test 36
18 **Gap extension penalty 0.5**. In these figures the number of episodes needed was plotted against the number of demonstrations. Each line represents one parameter configuration. In each sub figure only the matrix off-diagonal value differed. And from every sub figure to sub figure the gap open penalty changes. The axis are fixated. 46

19 **Gap extension penalty 1.0**. In these figures the number of episodes needed was plotted against the number of demonstrations. Each line represents one parameter configuration. In each sub figure only the matrix off-diagonal value differed. And from every sub figure to sub figure the gap open penalty changes. The axis are fixated. 47

20 **Gap extension penalty 75.0**. In these figures the number of episodes needed was plotted against the number of demonstrations. Each line represents one parameter configuration. In each sub figure only the matrix off-diagonal value differed. And from every sub figure to sub figure the gap open penalty changes. The axis are fixated. 48

List of Tables

1 Alignment results of sequences with 1000 to 1400 residues 18
2 ClustalW features 23
3 Top ten of alignments results of sequences in the range of 0 to 300 24
4 Top ten of alignments of sequences in the range of 0 to 400 25
5 Result of sequences in range 1000 to 1600 aligned with NJ 25
6 Alignment of all sequences with GO: 0.1, GE:0 26
7 Alignment of Sequences smaller than 400 residues with GO: 0.1, GE:0 27
8 Alignment of Sequences between 500 and 600 residues with GO: 0.1, GE:0 . 27
9 RUDDER grid search parameter 29
10 Ten best results 34
11 ANOVA test results 34

References

[1] Aiyar A. “The Use of CLUSTAL W and CLUSTAL X for Multiple Sequence Alignment”. In: *Bioinformatics Methods and Protocols. Methods in Molecular Biology*. Ed. by Krawetz S.A. (eds) Misener S. Vol. 132. Totowa, NJ: OHumana Press, 2000. Chap. 11, pp. 221–242.

[2] Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, and Sepp Hochreiter. “RUDDER: Return Decomposition for Delayed Rewards”. In: *CoRR* abs/1806.07857 (2018). arXiv: 1806.07857. URL: http://arxiv.org/abs/1806.07857.

[3] Ramu Chenna, Hideaki Sugawara, Tadashi Koike, Rodrigo Lopez, Toby J. Gibson, Desmond G. Higgins, and Julie D. Thompson. “Multiple sequence alignment with the Clustal series of programs”. In: *Nucleic Acids Research* 31.13 (July 2003), pp. 3497–3500. eprint: https://academic.oup.com/nar/article-pdf/31/13/3497/9487060/gkg500.pdf. URL: https://doi.org/10.1093/nar/gkg500.

[4] Biswanath Chowdhury and Gautam Garai. “A review on multiple sequence alignment from the perspective of genetic algorithm”. In: *Genomics* 109.5 (2017), pp. 419–431. URL: http://www.sciencedirect.com/science/article/pii/S0888754317300551.

[5] ClustalW. *ClustalW help*. http://www.clustal.org/download/clustalw_help.txt. Accessed: 2020-05-1. June 2010.

[6] M. O. Dayhoff and R. M. Schwartz. “Chapter 22: A model of evolutionary change in proteins”. In: *Atlas of Protein Sequence and Structure*. 1978. URL: http://ibi.zju.edu.cn/bioinplant/courses/dayhoffetal1978.pdf.

[7] Karyn Duggan. *EMBL*. Ed. by NandanaEditor Madhusoodanan. Accessed: 2021-01-12. Sept. 2019. URL: https://www.ebi.ac.uk/seqdb/confluence/display/THD/Help%20-%20Clustal%20Omega%20FAQ.

[8] DF. Feng and R.F. Doolittle. “Progressive sequence alignment as a prerequisitetto correct phylogenetic trees”. In: *Journal of Molecular Evolution volume* 25 (1987), 351â□□360.

[9] Dan. Gusfield. “Algorithms on Strings, Trees, and Sequences : Computer Science and Computational Biology”. In: Cambridge [u.a.] : Cambridge Univ. Pressy, 1999. Chap. 15.7.4, pp. 383–384.

[10] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. *MineRL: A Large-Scale Dataset of Minecraft Demonstrations*. 2019. arXiv: 1907.13440 [cs.LG].

[11] S Henikoff and J G Henikoff. “Amino acid substitution matrices from protein blocks”. In: *Proceedings of the National Academy of Sciences* 89.22 (1992), pp. 10915–10919. eprint: https://www.pnas.org/content/89/22/10915.full.pdf. URL: https://www.pnas.org/content/89/22/10915.

[12] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Leibo, and Audrunas Gruslys. “Learning from Demonstrations for Real World Reinforcement Learning”. In: (Apr. 2017).

[13] Ronald A. Howard. “Dynamic Programming”. In: *Management Science* 12.5 (1966), pp. 317–348. eprint: https://doi.org/10.1287/mnsc.12.5.317. URL: https://doi.org/10.1287/mnsc.12.5.317.

[14] Jonathan Hui. *RL - Imitation Learning*. Accessed: 03.03.2021. July 2019. URL: https://jonathan-hui.medium.com/rl-imitation-learning-ac28116c02fc.

[15] *Informationen zu Minecraft*. Accessed: 2021-01-11. Nov. 2020. URL: https://www.minecraft.net/de-de/about-minecraft.

[16] Kazutaka Katoh. *MAFFT version 7*. Accessed: 21.01.2021. 2013. URL: https://mafft.cbrc.jp/alignment/software/.

[17] M.A. Larkin, G. Blackshields, N.P. Brown, R. Chenna, P.A. McGettigan, H. McWilliam, F. Valentin, I.M. Wallace, A. Wilm, R. Lopez, J.D. Thompson, T.J. Gibson, and D.G. Higgins. “Clustal W and Clustal X version 2.0”. In: *Bioinformatics* 23.21 (Sept. 2007), pp. 2947–2948. eprint: https://academic.oup.com/bioinformatics/article-pdf/23/

21/2947/1057599/btm404.pdf. URL: https://doi.org/10.1093/bioinformatics/btm404.

[18] Lihong Li, Thomas Walsh, and Michael Littman. “Towards a Unified Theory of State Abstraction for MDPs.” In: Nov. 2006.

[19] *Minecraft*. Accessed: 11.01.2021. URL: https://minecraft.gamepedia.com/Minecraft.

[20] Saul B. Needleman and Christian D. Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins”. English (US). In: *Journal of Molecular Biology* 48.3 (Mar. 1970), pp. 443–453.

[21] CÃ©dric Notredame, Desmond G Higgins, and Jaap Heringa. “T-coffee: a novel method for fast and accurate multiple sequence alignment11Edited by J. Thornton”. In: *Journal of Molecular Biology* 302.1 (2000), pp. 205–217. URL: http://www.sciencedirect.com/science/article/pii/S0022283600940427.

[22] Stefano Pascarella and Patrick Argos. “Analysis of insertions/deletions in protein structures”. In: *Journal of Molecular Biology* 224.2 (1992), pp. 461–471. URL: http://www.sciencedirect.com/science/article/pii/002228369291008D.

[23] 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”. In: (2020). eprint: 2009.14108.

[24] *Ray v2.0.0.dev0*. Accessed: 23.01.2012. 2021. URL: https://docs.ray.io/en/master/tune/index.html.

[25] Elisha Odemakinde Data Scientist | AI Engineer | Machine Learning Researcher and Elisha Odemakinde. *Model-Based and Model-Free Reinforcement Learning - Pytennis Case Study*. Accessed: 11.01.2021. Dec. 2020. URL: https://neptune.ai/blog/model-based-and-model-free-reinforcement-learning-pytennis-case-study.

[26] Johannes SÃ¶ding. “Protein homology detection by HMMâ□□HMM comparison”. In: *Bioinformatics* 21.7 (Nov. 2004), pp. 951–960. URL: https://academic.oup.com/bioinformatics/article-pdf/21/7/951/749249/bti125.pdf.

[27] Fabian et al. Sievers. “Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega”. In: *Molecular systems biology* 7 (2011), p. 539. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3261699/pdf/msb201175.pdf.

[28] T.F. Smith and M.S. Waterman. “Identification of common molecular subsequences”. In: *Journal of Molecular Biology* 147.1 (1981), pp. 195–197. URL: https://www.sciencedirect.com/science/article/pii/0022283681900875.

[29] R.S. Sutton and A.G. Barto. *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018. URL: https://books.google.co.uk/books?id=uWV0DwAAQBAJ.

[30] Thompson, Julie D., Higgins, Desmond G., Gibson, and Toby J. “CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice”. In: *Nucleic Acids Research* 22.22 (Nov. 1994), pp. 4673–4680. eprint: https://academic.oup.com/nar/article-pdf/22/22/4673/7122285/22-22-4673.pdf. URL: https://doi.org/10.1093/nar/22.22.4673.

[31] Huan Yu and Minghua Deng. “ClustalY: speed up the guide tree building for ClustalW”. In: *Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA’05)*. 2005, 3 pp.–610.

7 Appendix

Figure 18: **Gap extension penalty 0.5**. In these figures the number of episodes needed was plotted against the number of demonstrations. Each line represents one parameter configuration. In each sub figure only the matrix off-diagonal value differed. And from every sub figure to sub figure the gap open penalty changes. The axis are fixated.

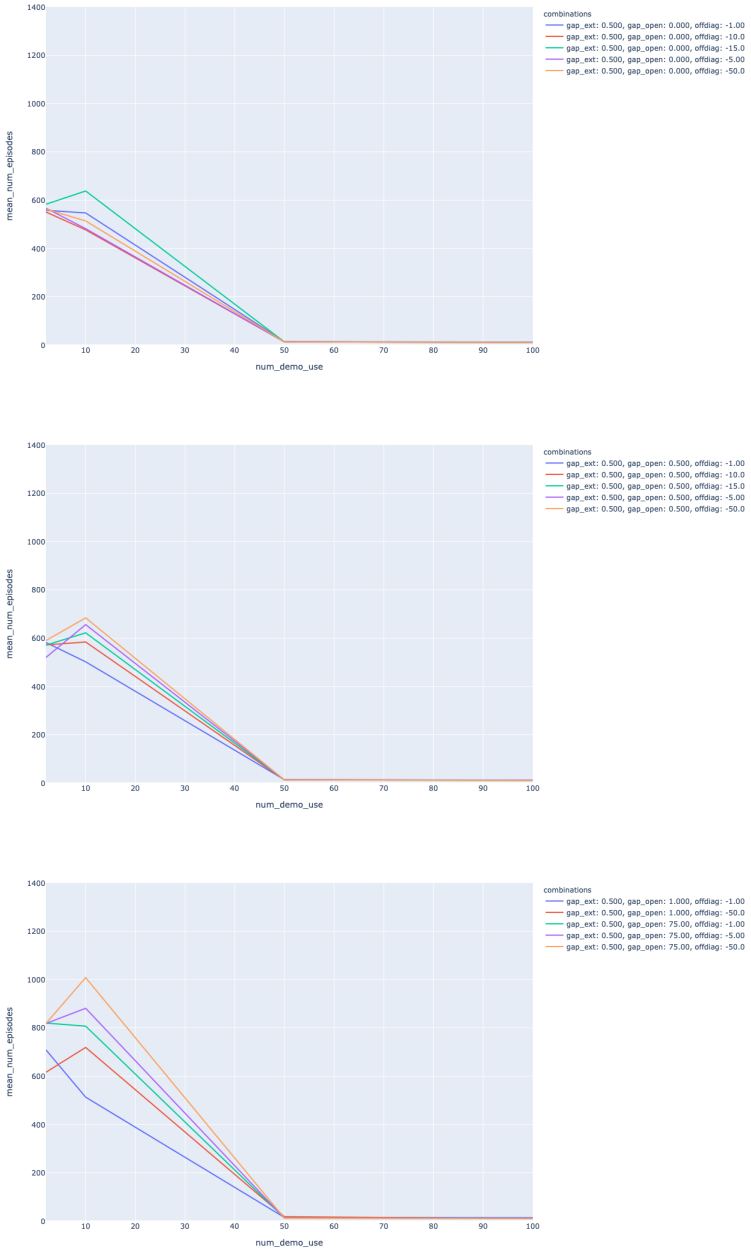


Figure 19: **Gap extension penalty 1.0**. In these figures the number of episodes needed was plotted against the number of demonstrations. Each line represents one parameter configuration. In each sub figure only the matrix off-diagonal value differed. And from every sub figure to sub figure the gap open penalty changes. The axis are fixated.

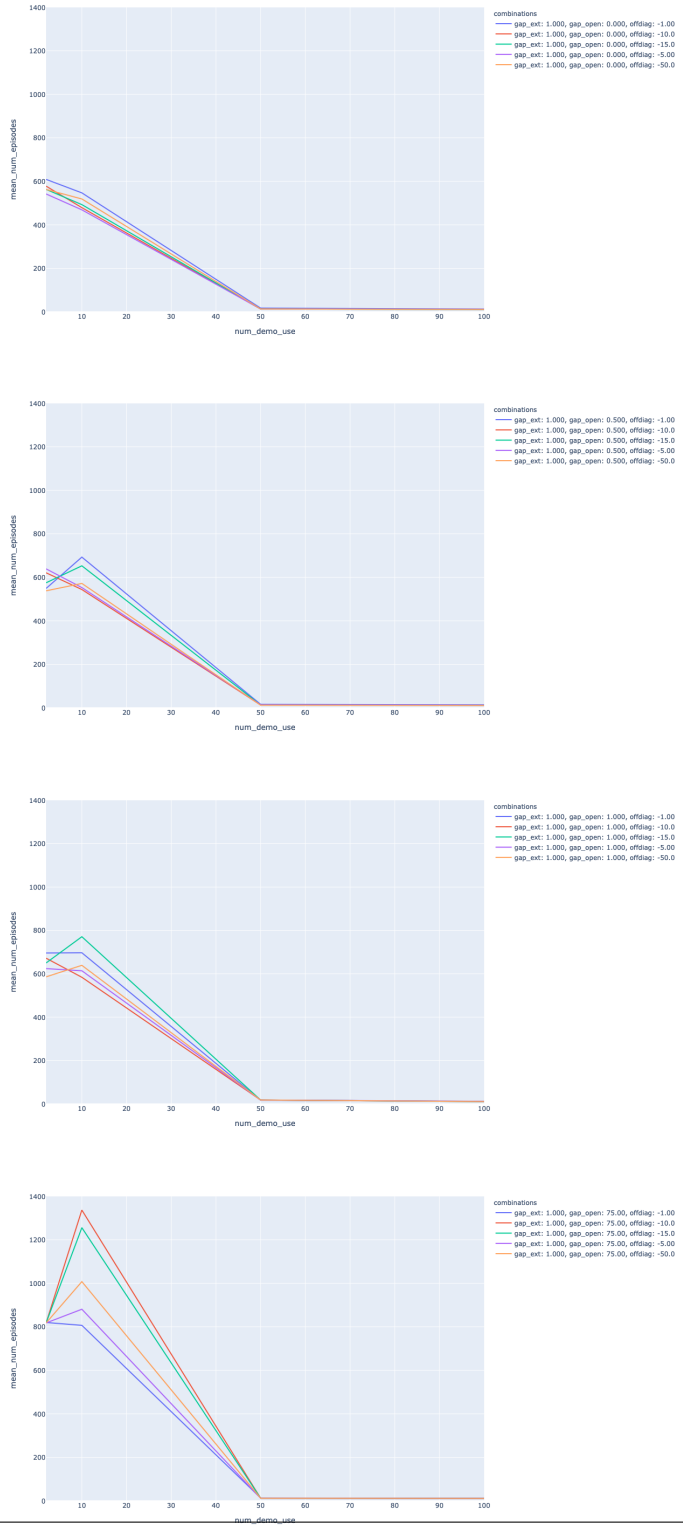


Figure 20: **Gap extension penalty 75.0**. In these figures the number of episodes needed was plotted against the number of demonstrations. Each line represents one parameter configuration. In each sub figure only the matrix off-diagonal value differed. And from every sub figure to sub figure the gap open penalty changes. The axis are fixated.