





Saarland University

Bachelor Thesis

Multiple Sequence Alignment Using Deep Reinforcement Learning

Roman Joeres

supervised by

Univ. Prof. Dr. Verena Wolf
Univ. Prof. Dr. Tobias Marschall

advised by

B.Sc. Timo P. Gros

November 09, 2020

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis. I marked the parts of this thesis where content or phrasing has been taken from others. For drawings, sketches or schemes as well as for graphical depictions I have stated whenever they were created by someone else. I declare that the written and electronic versions of this thesis are fully identical.

Saarbrücken, November 09, 2020

_____

Roman Joeres

Abstract

Multiple sequence alignment is one of the primal problems in biology and bioinformatics. The question of how to correctly align multiple sequences is crucial for many different other fields of research, e.g. gaining information about the evolutionary distance of two or more sequences and therefore about their corresponding species, finding protein targets for drugs, or finding a drug for a certain target protein.

Reinforcement learning, and especially deep reinforcement learning, has become popular in recent years. To name just a few, DRL has shown major success in complex games such as Atari Games, Chess, and Go. The underlying model of reinforcement learning are Markov decision processes.

The multiple sequence alignment can be represented as a Markov decision process in several ways. In this thesis, we focus on the easiest one (introduced by Mircea et al.) and apply several different algorithms from both tabular reinforcement learning and deep reinforcement learning. We compare our results to other work that also tried solving the multiple sequence alignment problem with reinforcement learning, as well as to other state-of-the-art tools that are widely used in bioinformatics.

We find we can outperform the existing reinforcement approaches by far and partially even compete with the state-of-the-art tools.

Acknowledgments

I would like to thank all the people who helped me with this thesis at each stage. My family also deserves my thanks for their support in non-technical matters throughout my studies. Without them, I could not have done this work in the way I wanted to do this.

In the beginning, Verena Wolf and mainly Timo Gros introduced me to Reinforcement Learning and how to solve tasks and to optimize multi-staged processes using deep learning. They supported me in all stages of this work with a lot of advice on how to solve tasks arising during the implementation and evaluation of the algorithms.

And I also thank Tobias Marschall for his biological input and his food for thought. Conversation with him helped a lot to find biological meaning in the results produces by the algorithms used in this thesis.

Introduction

Multiple sequence alignment is one of the most important open fields in bioinformatics because it has applications in nearly every other field of bioinformatics research. The most striking example is phylogenetic trees that are used to describe the evolutionary distance between species. The distance between species can be measured relative to the number of similar or dissimilar patterns between sequences of the species [1]. Computing the similarity of a pattern can be performed on both the protein sequences and the DNA sequences of species, because both give information on evolutionary changes in the sequences.

Another application for multiple sequence alignments is the field of drug design and drug development [2]. In those fields, multiple sequence alignments are often performed between a mixture of sequences. For some sequences of the mixture, the secondary structures or functions are known for the other sequences these informations are unknown. Multiple sequence alignments are then used to align similar regions of those sequences together based on sequences information. These regions are said to have similar secondary structures or similar functionality, so from sequence information, information about the secondary structure or the functionality are inferred [3].

Classical approaches often define a biological fine-tuned scoring function to score alignments. Along this scoring function, the alignments are then optimized to reproduce the true biological relationship between the sequences. Those tools often use classical optimization techniques like dynamic programming. Based on "true", hand-crafted and hand-optimized solutions for alignment problems, the solution of different tools on the same problem can be evaluated and compared [4, 5, 6].

Aside from those classical approaches, some new ideas utilize techniques from machine learning to find good or optimal alignments. Reinforcement learning provides techniques to solve singleplayer or multiplayer games by learning and interacting in a Markov decision process. The main idea in such settings is to have a try-and-error-agent that learns by interacting with the environment [7]. Using these techniques, some astonishing results have been achieved in various fields. Examples are Google DeepMind artificial intelligence for Atari Games [8], AlphaZero programs revolutionized how to play computer Shogi, Chess, and Go [9], and finally, AlphaFold, which won the CASP13 competition [10], a comparison of different protein folding prediction programs that tries to find the correct fold of a protein structure [11].

There are already approaches that apply reinforcement learning to the problem of aligning multiple sequences. They use many different algorithms and Markov decision processes to tackle the problem. The best of these approaches come from Mircea et al. [12], Jafari et al. [13], and Edelkamp et al. [14]. All of those approaches we briefly discuss in chapter 3. Besides the algorithms of the papers, we test many other algorithms from the field of deep reinforcement learning.

The existing papers on reinforcement learning methods for multiple sequence alignment have several drawbacks like no freely available source code, only one optimized score, and poor com-

parison with little significance, because the tests are not very extensive. We discuss those points in chapter 3 but they generate the need for more comparison with stronger significance and other scores to optimize.

None of the papers discusses their results in the context of the optimal alignment for a problem. But this is crucial because the SP-scores used for optimization are very different between the classical approaches. And, except one, none of the papers uses protein sequences to compare the performance, but the classical approaches are specially designed for protein alignments. So, there is a need for a comprehensive analysis of the performance and potential reinforcement learning has in the field of multiple sequence alignments.

In this thesis, we try to give this comprehensive analysis and offer a variety of interesting and promising ideas for future improvements of reinforcement learning approaches on multiple sequence alignments. We focus on the easiest Markov decision process proposed by Mircea et al. [12] for the multiple sequence alignment problem which is maybe not optimal but at the moment for larger problem instances is the only feasible one. With this model, already some algorithms have been tested that we review and compare. In addition to this, we implement other algorithms from reinforcement learning to provide a more broad overview of the performance and possible differences in the performances of different algorithms.

The rest of the thesis is structured as follows. We first introduce Markov decision processes and define some notation. Then we give an introduction to multiple sequence alignments and how the Markov decision process we use is structured. In the fourth chapter, we describe all algorithms from reinforcement learning we use to tackle the problem. In the fifth chapter, we then see how the agents perform on different sequence alignments and compare the results to other tools in this field.

Contents

1 Basics **3**
- 1.1 Markov Decision Process . . . 3
- 1.2 Policy . . . 4
- 1.3 Biological Sequences . . . 5

2 Multiple Sequence Alignment **7**
- 2.1 Relevance . . . 7
- 2.2 Scoring Profiles . . . 8
  - 2.2.1 Sum-of-pairs-score . . . 8
  - 2.2.2 Column Score . . . 10
  - 2.2.3 Q-score . . . 11
  - 2.2.4 TC-score . . . 11
- 2.3 Pairwise Alignments . . . 11
  - 2.3.1 Sequence-to-sequence Alignment . . . 11
  - 2.3.2 Sequence-to-profile Alignment . . . 14
- 2.4 Different Multiple Alignment Techniques . . . 15
- 2.5 Progressive Alignment . . . 17
- 2.6 Iterative Refinement . . . 19

3 Model and Related Work **21**
- 3.1 Markov Decision Process for Multiple Sequence Alignment . . . 21
- 3.2 Previous Work . . . 22
  - 3.2.1 Reinforcement Learning for Multiple Sequence Alignment . . . 22
  - 3.2.2 Reinforcement Learning for Multiple Sequence Alignment with affine Gap Costs . . . 23
  - 3.2.3 Deep Reinforcement Learning for Multiple Sequence Alignment . . . 23
  - 3.2.4 MCTS for the MSA problem . . . 24
  - 3.2.5 RLALIGN and Exploring Reinforcement Learning Techniques in Multiple Sequence Alignment . . . 24

4 Reinforcement Learning **27**
- 4.1 Reward-functions . . . 28
  - 4.1.1 Monte-carlo Return . . . 28
  - 4.1.2 Temporal Difference Learning . . . 28
  - 4.1.3 Lambda Return . . . 28
- 4.2 Learning Techniques . . . 29
  - 4.2.1 Online and Offline Learning . . . 29
  - 4.2.2 On-policy and Off-policy Learning . . . 29

4.2.3 ϵ-greedy learning . . . 29
4.2.4 Discount Factor . . . 30
4.3 Value-function Approximation . . . 30
4.3.1 Tabular Approaches . . . 30
4.3.2 Deep Q-networks . . . 31
4.4 Policy Approximation . . . 32
4.4.1 Policy-gradient Theorem . . . 32
4.4.2 REINFORCE . . . 33
4.4.3 REINFORCE with Baseline . . . 33
4.4.4 Actor-critic Methods . . . 34
4.5 Tree-based Approaches . . . 35
4.5.1 UCT . . . 35

5 Results **37**
5.1 General . . . 37
5.1.1 Sequence Identity . . . 37
5.1.2 Databases . . . 38
5.1.3 Compared Tools . . . 38
5.1.4 Parameter Settings . . . 39
5.2 DNA Datasets . . . 39
5.3 Protein Datasets . . . 41
5.3.1 Alignment Analyse . . . 43
5.4 Runtime . . . 47

6 Conclusion and Future Work **51**
6.1 Conclusion . . . 51
6.2 Future Work . . . 52
6.2.1 Biological . . . 52
6.2.2 Computer Science . . . 52

Appendix **59**

Chapter 1

Basics

In this chapter, we briefly discuss some basics for this work in both fields covered in this thesis. First, we start with the definition and some properties of Markov decision processes and how they can be solved by policies. The last section gives an introduction to biological sequences and some of their properties.

1.1 Markov Decision Process

Most of the environments where reinforcement learning is applied are based on Markov decision processes. These processes are discrete-time stochastic processes that are used to model decision-making environments. Using a Markov decision process, one can solve optimization problems using dynamic programming. In theory, this is feasible for all optimization problems, but to compute the optimal solution of larger instances of optimization problems one needs to approximate the solution because dynamic programming would take too long and is therefore not practical [7].

Definition 1.1 (Markov Decision Process [15], [7])
A Markov decision process can be defined as a quadruple of the form $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$.

1. $\mathcal{S}$: state space containing all possible states of the decision process.
2. $\mathcal{A}$: action space. This contains all available actions of the process and can be defined depending on the state a Markov decision process is in. This means that if not all actions are available in a state, the action space for this state is smaller than the action space for the full Markov decision process.
3. $\mathcal{P}(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$: state transition function describing the probability of reaching the successor state s' from state s by applying action a.
4. $\mathcal{R}(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$: reward function that assigns rewards to state-successor-pairs based on an action.

Definition 1.2 (MDP Notation)
Some further definitions on the notation needed for the definition of the algorithms we use:

1. T: final timestep at the end of an episode
2. t: running index through the timesteps from 0 to T
3. r_t: immediate reward of step t
4. G_t: accumulated return of an environment after time step t

The difference between return and reward is explained in section 1.3.

1.2 Policy

Definition 1.3 (Policy)
A policy is an action plan that selects actions in states, this is done by assigning a probability of performing the action. $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ with $\pi(a \mid s) = \mathbb{P}[A = a \mid S = s]$ gives the probability of performing the action a in state s. As in the definition of the transition function, a policy can be deterministic, so $\mathbb{P}[A = a \mid S = s] = 0$ or 1, or probabilistic.

From a policy two other functions can be derived, the state-value function and the action-value or Q-function.

Definition 1.4 (State-Value Function [16])
The state-value function that is defined as the expected return of the state s under the policy π.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{T-k-1} \gamma^k r_{t+k+1} \mid S_t = s \right]$$

In theory, this equation can be solved optimally according to every policy as one can represent $v_\pi(s)$ using its successor states.

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} \mathbb{P}[s', r \mid s, a] \left[r + \gamma v_\pi(s') \right]$$

With $\mathbb{P}[s', r \mid s, a]$ as the probability of reaching state s' and getting reward r from performing action a in state s. This equation is called the bellman equation for v_π.

Definition 1.5 (Action-Value Function, also called Q-function [17, 7])
The action-value function or Q-function that is defined as the expected return of taking the action a in state s under the policy π.

$$q_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{T-k-1} \gamma^k r_{t+k+1} \mid S_t = s, A_t = a \right]$$

Similar to the bellman equation for v_π presented above, there is also a Bellman equation for q_π to compute the optimal action-value function.

$$q_\pi(s, a) = \sum_{s',r} \mathbb{P}[s', r \mid s, a] \left[r + \gamma \max_{a'} \{ q_\pi(s', a') \} \right]$$

The solution of an environment, so the optimal behaviour of an agent in each state of the environment, is denoted as an optimal policy $\pi_*(a \mid s) = \mathbb{P}[A = a \mid S = s]$, where the $*$ denotes the optimality of the policy. The functions of this policy are denoted as v_* $(= v_{\pi_*})$ and q_* $(= q_{\pi_*})$.

1.3 Biological Sequences

Definition 1.6 (Sequence)
A sequence seq is defined as a string over an alphabet Σ. For protein sequences this alphabet contains 20 letters for the 20 amino acids. For DNA sequences holds $\Sigma_{DNA} = \{A, C, G, T\}$ representing the four nucleic bases.

Those sequences are then aligned with each other. Because aligned sequences have not always the same length, so-called gaps have to be inserted in an alignment.

Definition 1.7 (Profile)
We denote an aligned sequence as seq' and is a string over $\Sigma' = \Sigma \cup \{\backslash - \backslash\}$. The gaps, " $-\backslash$, represent mutations such as insertions and deletions of some bases when altering one sequence into another. A profile contains at least two aligned sequences and must fulfil three properties:

1. $\forall\ seq_1, seq_2 \in profile : |seq_1| = |seq_2|$
2. $\forall\ seq' \in profile : seq' \setminus$ "-" $= seq$
3. $\nexists\ i \in \#cols\ in\ profile : \forall\ seq' \in profile : seq'_i =$ "-"

The rows of the profile represent the sequences $seq'_1, ..., seq'_n$ resulting from the aligning. In the columns, we find the aligned bases or gaps, one per sequence. The bases in a column are called "aligned to each other" and represent evolutionary changes or conservations in the sequence between the according species.

Biological sequences have an orientation. That means, one sequence end is the start, and the other is the end. For this work, the orientations of the sequences are not important, in chapter 5 we refer briefly to this orientation for protein sequences. For those, the ends are called N-terminus and C-terminus according to the chemical structure of protein sequences.

Chapter 2

Multiple Sequence Alignment

Multiple sequence alignment and also pairwise alignments are very important techniques in bioinformatics research to find similarities in biological sequences [18]. Similarities in the biological sense mean that in two or more sequences there are similar patterns of e.g. bases or nucleotides that are conserved from one sequence or species to another. Multiple sequence alignment is a basic technique for many analyses performed in bioinformatics, such as the computation of phylogenetic trees to display the evolutionary origin of species and how they are related to each other [1]. Also, computing protein structures of new proteins make use of these techniques [19, 20] and the process of drug development [2].

We focus in first place on global multiple sequence alignments of DNA sequences, but we can easily extend the presented techniques to other kinds of sequences since there is no sequence specificity in the discussed algorithms like in other approaches. So, we apply the algorithms also to protein alignments to see how well we compete with other protein aligning tools. This trade-off from DNA to proteins is not that easy in terms of prediction quality because applying the same algorithm for multiple sequence alignment to protein-sequences and DNA-sequences might influence the quality of the results in classical approaches a lot. Algorithms like CLUSTAL [4], MAFFT [5] or MUSCLE [6] use sequence-specific information to optimize their scoring function. These tools tend to overfit one type of biological sequence since the properties of DNA and protein sequences are quite different. So, there are many more tools optimized for proteins than for DNA, as proteins are often more interesting because of the more direct relation to drug development. Another reason for this might be the lack of perfectly or correctly aligned or even hand-crafted DNA alignments. There are multiple benchmark databases for multiple protein sequence alignments such as BAliBASE [21], OXBench [22], PREFAB [6] and SMART [23], but not that much for DNA-alignments [24].

2.1 Relevance

As already mentioned, multiple sequence alignments are incorporated in almost every field of research in bioinformatics. Secondary structure prediction for proteins, for example, can be made based on multiple sequence alignments. Knowing the secondary structure of proteins is of enormous interest in ongoing research, as it can find potential functional sites in proteins that can have an influence on the functionality of the proteins. One example is the active site of proteins, where the key processes of a protein take place, such as splitting ligands or binding other proteins to form complexes with new functions [2]. To find similar patterns in structures, one could align the sequences and infer similarly structured for similar parts of the sequences. Another approach

is to align the structure itself to predict the 3D-fold of proteins, called structural alignment. As an example, the algorithm MUSTANG performs alignments to predict the 3D-structure of proteins using not their amino-acids but the position in a three-dimensional space of specific carbon-atoms in the protein's backbone. The actual fold can then be determined based on the known folds of the co-aligned sequences [3].

Another way to use multiple sequence alignments to gain information about proteins is to find out which function in a cell a protein has. By performing multiple sequence alignments, similar proteins from other species can be identified that might have the same function in similar pathways. This approach to finding functions of unknown proteins is based on the idea that proteins with similar structures often have similar functions [25]. Here, the way to extract the functionality of the unknown sequence works similar to the computation of the folding based on the co-aligned sequences.

Also in genetic research on a level below proteins and their sequences, multiple sequence alignment has applications. So, we can use multiple sequence alignments for the construction of phylogenetic trees, which can then infer evolutionary distances between species. We can compute these distances from the number of changes that occurred in the evolution of a sequence [1].

2.2 Scoring Profiles

To align two or more sequence and to interpret the results of an alignment process, we have to score the resulting profile. That means we have to evaluate the quality of a profile to be able to compare two different alignments of the same sequences and to reason which one is the better one. For this propose different scores have been developed and each of them has advantages and disadvantages as we discuss in the following.

Example 2.1 (Alignment)
The example we use to demonstrate the scorings consists of the three sequences $seq_1 = ATTG$, $seq_2 = ATCCG$ and $seq_3 = ATGT$. The exemplary alignment

$$\begin{array}{cccccc} A & T & T & - & G & - \\ A & T & C & C & G & - \\ A & T & - & - & G & T \end{array}$$

is used to apply different scorings to.

2.2.1 Sum-of-pairs-score

The sum-of-pairs-score measures the number of perfectly aligned pairs of bases in relation to the total number of aligned pairs where each type of pair is weighted differently. This score is applicable for both pairwise alignments and multiple alignments. In its formula

$$SP = \sum_{i=1}^{N} \sum_{j=1}^{n-1} \sum_{k=i+1}^{n} sp\left(c_i^j, c_i^k\right),$$

i iterates over all columns of an alignment and the j and k index the bases in the alignment, so that the sum iterates column-wise over all pairs of bases aligned to each other. In more detail: c_i^j is the j-th base of the i-th sequence. The function sp returns the score of two sequences and can be defined arbitrarily. As mentioned above the pairwise alignment we use is based on this scoring, the choice of sp and its definition has a huge influence on the resulting alignments, since

this directly defines what a good match of two bases is, and indirectly how a good alignment look like. In this work we used

$$sp\left(c_i^j, c_i^k\right) = \begin{cases} 0 & \text{if } c_i^j = c_i^k = \text{"}-\text{"} \\ 2 & \text{if } c_i^j = c_i^k \\ -1 & \text{if } c_i^j = \text{"}-\text{" or } c_i^k = \text{"}-\text{"} \\ -2 & \text{if } c_i^j \neq c_i^k \end{cases}$$

as Mircea et al. [12], because we want to improve the reinforcement learning approach of the paper and not the alignment algorithm. So, we fix the parameters to get comparable results. From the definition of $sp(*,*)$ it follows that we have to maximize the SP-score of an alignment. One could define s also as a measure of the distance and give credit for aligning gaps to bases. Then the gap-score (align gap against sequence) would be higher than the match-score (align two equal bases) and one has to minimize the SP-score of the alignment. But this is not part of we discuss more deeply and only named for completeness.

This score is also used in state-of-the-art tools we later compare the results to. Those tools use a much more fine-tuned formula, taking biological sense into account. In CLUSTALW this is among others done by sequence weighting based on the (dis-)similarity of a sequence to other sequences. The higher the dissimilarity or divergence of a sequence, the higher the sequence weight. These weights are used as multiplication factors for the sequences. There are several other improvements to this plain SP-scoring function presented above [4].

In the reference literature of reinforcement learning for multiple sequence alignment, two different scoring structures for the gaps are used. First, the linear gap costs, we already use in the function above and second, affine gap costs that more likely insert fewer, but therefore larger gaps than many small gaps. The scoring of the individual gaps has a large influence on the scoring of alignments and also on the definition of what a good alignment is.

Linear Gap Costs

When using the idea of linear gap costs, each part of a gap is penalized with the same weight. So, there is no difference if a newly inserted gap extends an already existing gap of opens a new one. This way is easy to compute but has the disadvantage that evolutionary coherent gaps are more likely compared to many small gaps.

Example 2.2 (SP-score of alignment)
Using this idea in the score above, the above exemplary given alignment of seq_1, seq_2 and seq_3 would get a score of $6 + 6 - 4 - 2 + 6 - 2 = 10$.

A	T	T	$-$	G	$-$
A	T	C	C	G	$-$
A	T	$-$	$-$	G	T
6	6	-4	-2	6	-2

Affine Gap Costs

The disadvantage of scoring many small gaps as well as fewer, larger gaps can be cured using affine gap costs. That means that a gap has not always the same score, but the score for a gap depends on the type of gap. The gaps, opening a new, larger gap receives a *gap-opening* penalty

and gaps that only extend an already existing gap, receives a *gap-extension* penalty. Using this idea, we can now (only for this example) redefine the sp-function

$$sp\left(c_i^j, c_i^k\right) = \begin{cases} 0 & \text{if } c_i^j = c_i^k = \text{" - "} \\ 2 & \text{if } c_i^j = c_i^k \\ -1 & \text{if } c_i^j = \text{" - "} \text{ or } c_i^k = \text{" - "} \text{ (opening)} \\ -0.5 & \text{if } c_i^j = \text{" - "} \text{ or } c_i^k = \text{" - "} \text{ (extending)} \\ -2 & \text{if } c_i^j \neq c_i^k. \end{cases}$$

Example 2.3 (SP-score with affine gap costs of alignment)
With this function, the alignment above would get a score of $6 + 6 - 4 - 1.5 + 6 - 2 = 10.5$.

A	T	T	$-$	G	$-$
A	T	C	C	G	$-$
A	T	$-$	$-$	G	T
6	6	-4	-1.5	6	-2

There are some other ways to compute gap costs, but they are not relevant for our work. The affine gap costs are not used in our work, but in a reference work, and for completeness we should have a look at another technique of scoring gaps within the SP-score. In the following work, we always refer to the SP-score with linear gap costs.

2.2.2 Column Score

Beside the SP-score, there are some other ways and metrics to score an alignment. One of those we also use for optimization is the *column score* (C-score). This cures one disadvantage of the SP-score that is very relevant for optimizing the score in reinforcement learning. In some reinforcement learning settings, it is helpful to know the maximal value a scoring or evaluation function can take. The need for this upper bound becomes more clear, when we discuss the results and especially the performance of the UCT-algorithm. But for now it is sufficient to know that, given some input sequences that are to align, there is no good upper bound for the SP-score of the optimal alignment regarding the SP-score of the input sequences.

This challenge can be overcome by the column score

$$C(\textit{profile}) = \frac{\#\text{perfectly aligned columns}}{\#\text{columns in profile}}.$$

Example 2.4 (C-score with affine gap costs of alignment)
With this function, the alignment above would get a score of $\frac{1+1+0+0+1+0}{6} = 0.5$.

A	T	T	$-$	G	$-$
A	T	C	C	G	$-$
A	T	$-$	$-$	G	T
1	1	0	0	1	0

One can differ two major types of scoring alignments. The first we discuss in the previous two sections can be computed only based on the alignment itself. No other information except the alignment is needed to compute the SP-score and the C-score. The second type can only be computed relative to a "true" alignment. "True" alignment means verified and hand-crafted alignments that are considered being the true solution to a sequence alignment problem. This

means researchers believe those alignments are the true alignments, how evolution formed the sequences out of each other, even though those alignments are not those with the highest SP-score or C-score.

The Q-score and the TC-score are defined and used according to the definition by Robert C. Edgar [6] and computed used the QSCORE program of Edgar [26].

2.2.3 Q-score

The Q (quality)-score

$$Q(profile, reference) = \frac{\#\text{correctly aligned pairs in profile}}{\#\text{aligned pairs in reference}}$$

is a more detailed score, because it sums over pairs instead of columns as in the TC-score. Therefore, the Q-score is always greater or equal to the TC-score.

This quotient puts the number of correctly aligned pairs in relation to the total number of pairs in the reference. This is like the SP-score where also mismatches or gaps can be counted positively. Correct aligned in this context means that the same pair of bases is present in both, the computed profile and the reference alignment.

2.2.4 TC-score

The TC (total column)-score is computed in the following way

$$TC(profile, reference) = \frac{\#\text{correctly algined columns in profile}}{\#\text{columns in reference}}$$

This score puts the number of correct columns in relation to the number of columns in the reference alignment. As the Q-score is similar to the SP-score but with a reference, this TC-score is similar to the C-score above where not the rate of perfectly aligned columns is computed, but the rate of correct aligned columns in the profile, compared to the reference.

These last two scores can be used to compare different alignment to see how well they approximate the "true" or reference alignment. These scores cannot be approximated in the learning process, as in an application setting they cannot be computed, because the reference alignment should be unknown in order to compute approximations for it. If the true alignment would be known, there is no need to compute approximations. The SP-scores used in the state-of-the-art tools are designed to find nearly optimal alignments in terms of Q-score and TC-score [6].

2.3 Pairwise Alignments

Pair-wise alignments are the key algorithms for this thesis, because they are fast and optimally to compute. As referred to in section 2.5, we only need to perform pairwise alignments, either of two sequences (sequence-to-sequence alignment) or of a sequence against a block of already aligned sequences (sequence-to-profile alignment).

2.3.1 Sequence-to-sequence Alignment

The algorithm we use is based on the idea of S. B. Needleman and C. D. Wunsch [27] which later has been adapted by T. F. Smith and M. S. Waterman [28] to compute an optimal alignment

algorithm of two sequences. In this section we shortly discuss how the algorithm works and show an example alignment using the two sequences $seq_1 = ATTG$, $seq_2 = ATCCG$.

The algorithm make use dynamic programming and computes in a way the scores of all possible alignments between all prefixes of the two sequences that are then stored in a table.

At first we have to select a scoring function, which is here the one presented above.

1. The table we fill within this algorithm consists of as many rows and columns as both sequences have.

   Example 2.5 (Example alignment)
   Aligning seq_1 and seq_2 from above, at the beginning the table looks like

			c_1^1	c_1^2	c_1^3	c_1^4
		-	A	T	T	G
	-					
c_2^1	A					
c_2^2	T					
c_2^3	C					
c_2^4	C					
c_2^5	G					

2. The next step is to fill the cells in the column and row with the gap. These cells are filled as one would align many gaps again the according prefix of a sequence.

   Example 2.6 (Example alignment)
   Filled upper and left edges in the example alignment.

			c_1^1	c_1^2	c_1^3	c_1^4
		-	A	T	T	G
	-	0	-1	-2	-3	-4
c_2^1	A	-1				
c_2^2	T	-2				
c_2^3	C	-3				
c_2^4	C	-4				
c_2^5	G	-5				

   The filling of the gap-fields implicitly follow already the function used to fill the remaining cells

$$\text{For } i,j > 0 : c\left(c_1^j, c_2^j\right) = max \begin{cases} c\left(c_1^{i-1}, c_2^{j-1}\right) + sp\left(c_1^i, c_2^j\right) \\ c\left(c_1^{i-1}, c_2^j\right) + sp\left(c_1^i, -\right) \\ c\left(c_1^i, c_2^{j-1}\right) + sp\left(-, c_2^j\right) \end{cases}$$

   Here $c(*,*)$ denotes the value of the cell of the two arguments and $sp(*,*)$ represents the SP-score of the two input bases. c_*^0 for this purpose is defined as gap.

3. The last step is to fill the entire table following applying the formular to each cell as soon as the three needed adjacent cells are filled

Example 2.7 (Example alignment)
The finally filled matrix with the scores of all possible alignments.

			c_1^1	c_1^2	c_1^3	c_1^4
		-	A	T	T	G
	-	0	-1	-2	-3	-4
c_2^1	A	-1	2	1	0	-1
c_2^2	T	-2	1	4	3	2
c_2^3	C	-3	0	3	2	1
c_2^4	C	-4	-1	2	1	0
c_2^5	G	-5	-2	1	0	3

The resulting alignment score is always the value in the down-right cell, here marked green and has the value 3. So, the SP-score of the resulting optimal alignment of seq_1 and seq_2 is 3. To get the alignment, we track back, from which cell the score in the actual cell can come from. That means, which of the rules in $c(*,*)$ has been applied to get a score in a cell. For example, the green value 3 is reached from (c_1^3, c_2^4) by aligning c_1^4 with c_2^5. From that it follows that the alignment ends with (...G, ...G) and the next cell to track back is (c_1^3, c_2^4). Here, there are two options: start from (c_1^2, c_2^3) and perform a mismatch or from (c_1^3, c_2^3), by aligning a gap to c_2^3. To our knowledge, there is no solution for those conflicts, so we resolve them by assigning priorities to each direction an alignment can come from. For both, sequence-to-sequence and the sequence-to-profile alignment later, we use the order LEFT, DIAGONAL, UP. So, if possible, we track the alignment back to the left direction. That means, if possible and optimal, a gap is aligned to the sequence on top. If this is not possible, we try the diagonal alignment of a (mis-)match otherwise we align a gap to sequence on the left. These computations can also be performed during the computation of the alignment-matrix and stored in a second matrix which makes the evaluation of the remaining matrix more easy.

The order defined in the last paragraph has some advantages. First, the algorithm to align a profile against a sequence is implemented such that the profile always is at the top of the alignment matrix, so going up as the last option ensures that the profiles gets a gap if and only if there is no other option. And going left before diagonal comes from the scoring matrix and its weighting of gaps and mismatches. Remember that aligning a gap gets a penalty of -1 but a mismatch of -2, so this indicates that mismatches are worse than gaps, so we choose the above mentioned order.

Example 2.8 (Example alignment)
Evaluation of the example alignment.

			c_1^1	c_1^2	c_1^3	c_1^4
		-	A	T	T	G
	-	0	-1	-2	-3	-4
c_2^1	A	-1	2	1	0	-1
c_2^2	T	-2	1	4	3	2
c_2^3	C	-3	0	3	2	1
c_2^4	C	-4	-1	2	1	0
c_2^5	G	-5	0	1	0	3

Table 2.1: Scoring table of the aligning

			c_1^1	c_1^2	c_1^3	c_1^4
		-	A	T	T	G
	-		←	←	←	←
c_2^1	A	↑	↖	←	←	←
c_2^2	T	↑	↑	↖	←	←
c_2^3	C	↑	↑	↑	←	←
c_2^4	C	↑	↑	↑	↖	←
c_2^5	G	↑	↑	↑	↖	↖

Table 2.2: Direction table of the aligning

The resulting alignment contains one gap in the shorter sequence and has a SP-score of $2 + 2 - 1 - 2 + 2 = 3$.

A	T	$-$	T	G
A	T	C	C	G
2	2	-1	-2	2

One interesting property mentioned already above is that each cell in the left table represents the optimal SP-score of the alignments of both prefixes up to the considered cell.

Example 2.9
The optimal alignment of AT (c_1^1, c_1^2) and ATCC (c_2^1, c_2^2, c_2^3, c_2^4) would have a score of $2+2-1-1 = 2$.

A	T	$-$	$-$
A	T	C	C
2	2	-1	-1

The runtime is in $O(n^2)$, where n is the length of the longer sequence. This is because of the table one has to fill, which represents all possible alignments of the two sequences. The space usage in this algorithm is also quadratic in the sequence length, because the at least one table has to be stored. Using the Hirschberg algorithm, the space-usage can be reduced to $O(n)$ [29].

2.3.2 Sequence-to-profile Alignment

The sequence-to-profile alignment algorithm extends the Needleman-Wunsch algorithm explained above. Since this way of aligning a sequence to a profile is not described in any reference paper, we have to find an appropriate way to define this algorithm. The setting is quite similar to the algorithm we discuss in the section on sequence-to-sequence alignment, but there are still some differences to take care of.

One is that there are multiple sequences on one side to align and only one on the other side. So, we have to adjust the scoring system to take this imbalance of sequences into account. In order to keep the property described above (each cell denotes the optimal SP-score of the prefix-alignments), we design the scoring function in a way that in each cell the optimal SP-score of an alignment of the prefixes of the sequence and the profile is denoted.

1. We again have to define the scoring function, here denoted as a scoring matrix of the profile against all DNA-bases. So, this scoring function depends on the position of the profile, where to align a base and the base to align at that position.

Example 2.10
Using the profile from the previous section, we come up with a scoring table that represents a function.

	-	p^1	p^2	p^3	p^4	p^5
-	0	0	0	-2	-4	0
A	-2	6	-2	-4	-6	-2
C	-2	-2	-2	0	-2	-2
G	-2	-2	-2	-4	-6	6
T	-2	-2	6	-4	-2	-2

This table can be derived by computing the SP-score of a potential alignment of a base against a single column of the profile. At position p_1 the profile as an internal SP-score of two because of the match of the two As. By aligning another A against this position, the score is increased to 6 ($= 3 * 2$, as there are three distinct pairwise matches of As scored with two each).

Example 2.11
We discuss how the algorithm works, again using a small example. The profile above aligned against the sequence $seq_3 = ATGT$. For this propose, we have to fill another table similar to the table above.

		-	p^1	p^2	p^3	p^4	p^5
	s_1		A	T	-	T	G
	s_2		A	T	C	C	G
	-						
c_2^1	A						
c_2^2	T						
c_2^3	G						
c_2^4	T						

Table 2.3: Scoring tabe of the sequence-profile alignment

2. Based on this scoring function and the algorithm already described above, one could fill the table of the alignment.

   Example 2.12
   When filling the table, we can again compute the corresponding table with the directions as in the last section

   From these two table the following profile with an SP-score of $6 + 6 - 4 - 2 + 6 - 2 = 10$ results.

2.4 Different Multiple Alignment Techniques

As mentioned above the problem of finding optimal pairwise alignments is feasible in quadratic runtime and space usage that can be reduced to be a linear usage in space [29]. Both measures are relative to the number of sequences. The technique of computing a complete alignment matrix using dynamic programming can be easily extended to multiple sequence alignments. But the

		-	p^1	p^2	p^3	p^4	p^5
	s_1		A	T	-	T	G
	s_2		A	T	C	C	G
	-	0	0	0	-2	-6	-6
c_2^1	A	-2	6	6	4	0	0
c_2^2	T	-4	4	12	10	6	6
c_2^3	G	-6	2	10	8	8	12
c_2^4	T	-8	0	8	6	6	10

Table 2.4: Scoring table of the aligning

		-	p^1	p^2	p^3	p^4	p^5
	seq_1		A	T	-	T	G
	seq_2		A	T	C	C	G
	-		$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
c_3^1	A	$\uparrow$	$\nwarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
c_3^2	T	$\uparrow$	$\uparrow$	$\nwarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
c_3^3	G	$\uparrow$	$\uparrow$	$\uparrow$	$\leftarrow$	$\nwarrow$	$\nwarrow$
c_3^4	T	$\uparrow$	$\uparrow$	$\uparrow$	$\leftarrow$	$\nwarrow$	$\uparrow$

Table 2.5: Direction table of the aligning

A	T	T	$-$	G	$-$
A	T	C	C	G	$-$
A	T	$-$	$-$	G	T
6	6	-4	-2	6	-2

runtime and the usage of space of this extension increases exponentially with the increasing number of sequences. So, normally for finding the optimal alignment of n sequences one needs time in $O(k^n)$ where k is the maximal sequence length. The result of this time consumption is that this alignment technique is only in theory feasible. Therefore, different algorithms to reduce the search space or to approximate the optimal alignment have been developed, of which we shortly discuss a few. Indeed, the complexity of multiple sequence alignments with SP-score is proven to be NP-complete and therefore for larger problem instances not feasible in an acceptable amount of time [30].

In 1988, Carrillo and Lipman published an idea to reduce the search space in multiple sequence alignments using a branch-and-bound algorithm. The algorithm is based on Dijksta's shortest path algorithm and interprets the n-dimensional tables that would arise in an optimal multiple sequence alignment as a directed graph with weighted edges. The result is optimal, but the search time can be reduced [31]. This might help with sequences that are quite different, because many branches can be left out, but for very similar sequences the runtime still converges to $O(k^n)$.

Another idea, starting from the optimal alignments, is to use divide-and-conquer methods to reduce the search space and to find the optimal alignment in acceptable time. The main idea is to split a whole multiple alignment into many smaller alignments that can be solved very fast (if the sequence-lengths are around 3-5 sequences, the number does not matter that much) and then merge the sub-alignments into larger ones to finally get the optimal alignment. But this leads to the new problem of finding positions where to cut the alignments. This problem is also NP-complete and therefore this idea is also not feasible [32].

The last three ideas lead to optimal solutions but are quite slow. To overcome this slowness is to not find the optimal solution but to approximate the optimal alignment. One algorithm of those performing an approximation is the so-called center-star algorithm. The main-idea of this is to find a sequence that minimizes the number of mutations to all other sequences. It is proven that this center-star score approximates the SP-score up to a factor of 2; the algorithm is a so-called 2-approximation of the SP-score. 2-approximation means that the score of the center-star algorithm is at least half as good as the optimal score for the same alignment problem [33].

The algorithms of most of the state-of-the-art algorithms use are progressive alignments and iterative alignments presented in the next two sections in more detail. There are some other techniques using hidden-Markov-models [34], genetic algorithms [35] or simulated annealing [36] but those are not of interest for us and they are not related to what we discuss here.

2.5 Progressive Alignment

The core idea of progressive alignments is to reduce the multiple sequence alignments that are not feasible to many pairwise alignments that can be solved very fast. So, given a set of sequences or pre-aligned profiles of two or more sequences, we now do not align all sequences at once but select two of the elements in the list and align them into another alignment that replaces its parents in the set for the next alignment step. By doing so, the number of elements in the list that are to align is progressively reduced, until there is only one element in the list that contains all input sequences and is considered as the result of the algorithm.

The choice on which pair of sequences or profiles to align next can be done based on similarity of sequences. This similarity guided alignment choice can be computed before the main alignment process starts and result in so-called guide trees as presented below. Or it is done before each alignment step by computing the similarity of the elements to align next and choosing the most similar pair.

Example 2.13
Guide-tree of the alignment we computed in the previous sections.

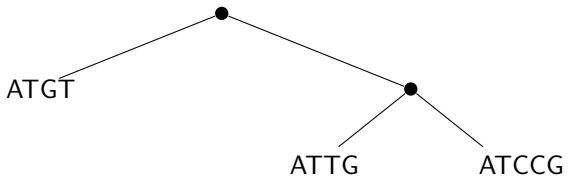


Figure 2.1: Guide-tree for the multiple sequence alignment we computed in section 2.3

Example 2.14
Guide-tree for alignment with 7 sequences as used in classical approaches.

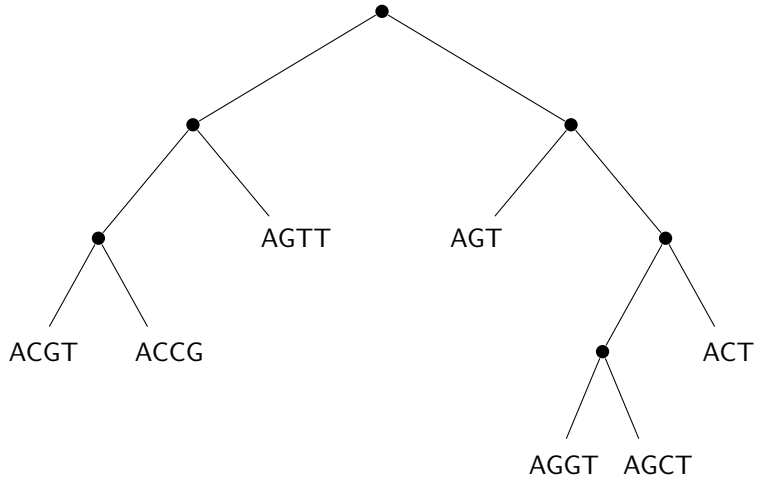


Figure 2.2: More complex example of a guide tree that might be the schematic output of CLUSTAL, MAFFT, or MUSCLE.

Those guiding trees use profile-to-profile alignments to resolve inner nodes that introduce a need for this. We use a special form of progressive alignments is performed, where profile-to-profile alignments are not needed. This is done by restricting the trees to be in a way an order of the sequences in which the sequences then are aligned. So, we do not have to create a tree, but generate orders of the input sequences and search for the best one.

Example 2.15
Guide-tree for alignment with 7 sequences as produced by out tool.

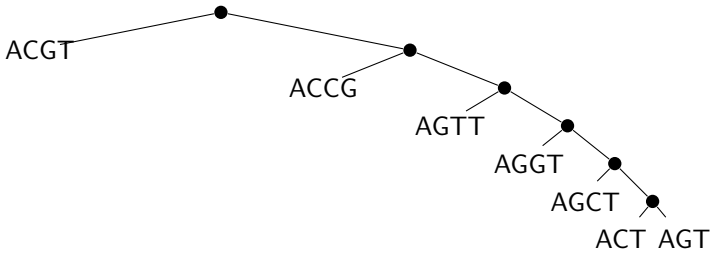


Figure 2.3: The only type of guide-trees for reinforcement learning guided progressive alignments are stairs.

One drawback of progressive alignments is the fact that a sequence is considered only once. That means, a sequence that is aligned, cannot be realigned. Feng and Doolittle call this principle once a gap, always a gap [1]. One advantage of having the possibility to realign sequences again and again is that those realignments could improve the quality of the resulting alignments and therefore the performance of the entire algorithm. But this realign offers a new problem. Whereas, progressive alignment is by construction of the algorithm forced to terminate with

the possibility of realigning, this natural termination no longer exists and therefore another termination criterion must be found [4]. This idea of realigning sequences is summarized under the term iterative alignment or iterative refinement, which we handle in the next section.

Another problem with progressive alignment is that by the selection process, the resulting profile can be caught in a local minimum. In the ideas presented in the previous section, there is in theory always a guarantee that the algorithm reaches or at least approaches the global optimal alignment and one can give a bound for the distance to the global optimum. This guarantee cannot be given in progressive alignments. This is also due to the above mentioned problem of once a gap, always a gap. Misalignments made early in the alignment procedure cannot be corrected to better alignments. To solve this problem, one could again use iterative refinement techniques to optimize former alignments.

2.6 Iterative Refinement

As explained above, iterative refinements aim to fix one of the biggest drawbacks of progressive refinement, the option to realign and therefore to fix errors made in early alignment steps. One of the tools performing iterative refinement as a main step of its algorithm is MUSCLE. In a first step, MUSCLE produces a guide-tree based on the k-mer distance of each pair of input sequences. A k-mer is a k bases long substring of a sequence. This first tree is then used to compute a first progressive alignment which is improved in a second progressive alignment step [6]. In this second step MUSCLE performs mainly the same steps as in for the first alignment but the distance measure based on which the alignment tree is created is switched from the k-mer distance to the Kimura distance that is more accurate than the k-mer distance but needs more time to be computed since for the Kimura distance the two sequences has to be aligned [37]. From this second tree and the first alignment, a second better alignment is created that will undergo an iterative refinement procedure to further cure errors made while aligning along the trees. Therefore, from the Kimura distance tree an edge is deleted, two new profiles along the subtrees are computed and at the end matched. If this newly created alignment is better than the one before, the new one is kept. This procedure is performed until a user-defined stop criterion is fulfilled [6].

Our way of iterative refinement is different. After receiving an alignment of the input sequences from the progressive alignment step, single sequences can be realigned against the remaining block of sequences. As stop criterion, the number of sequences is used. So, if the task is to find the optimal global multiple sequence alignment of 5 sequences, in the refinement step, at most 5 realignments are performed, but the algorithm has the option to stop the realigning after realigning less than 5 sequences. The performance of this approach can be seen in chapter 5.

Chapter 3

Model and Related Work

In this chapter, we will discuss the model of multiple sequence alignments as a Markov decision process we take from the paper on which this work base on. In the second section of this chapter, we focus on the different approaches that used reinforcement learning, not necessarily deep reinforcement learning to improve solutions in multiple sequence alignment.

3.1 Markov Decision Process for Multiple Sequence Alignment

In this section, we discuss the definition of the states, actions, and functions of the Markov decision process that underly this thesis. Furthermore, we discuss some other properties of the model. These definitions mostly stick to the definitions of the main reference paper of Mircea et al. [12].

We begin with the state space. As we discuss in the section on progressive alignments, we are searching for optimal orders to align the sequences. So, each state represents an order of a subset of the sequences. The only initial state is the one representing the empty set or empty order that contains no sequence. Goal states are those that contain orders of sequences with no duplicate as defined in the section on progressive alignments. As the definition of goal states indicates there is another group of states besides valid, initial, and goal states that ends the Markov decision process or is at least invalid. These invalid terminal states contain duplicates of sequences in their orders and lead directly to a negative reward. The number of possible states is given as

$$|\mathcal{S}| = \frac{n^{n+1} - 1}{n - 1}$$

which can be inferred from the paper of Mircea et al. [12]. This description becomes more clear in figure 3.1 and after defining the actions and the transition function.

The action space comprises all available actions, i.e. sequences that are to align. So, in each state, the agent can choose to align every sequence independent of the validity of the resulting state. As each state has exactly one sequence of actions to get to this state, we can write a state as the sequence of its actions.

The transition function is completely deterministic. The result from choosing in a state a sequence of your choice leads always to the same next, successive order that is computed by putting the index of the selected sequence at the end of the order of the starting state of a

transition. This represents the alignment of the chosen sequence to the actual profile from the starting state of a transition.

To visualize these properties of the Markov decision process for multiple sequence alignment, we can have a look at example 3.1. As it is completely deterministic and the states grow with each action, we can plot the state space as a tree with the initial state as root and the terminal states as its leaves. Here, a state is indexed with the according actions.

Example 3.1
State space of the model

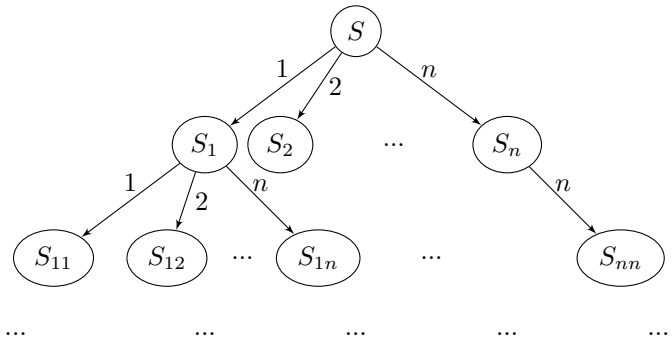


Figure 3.1: Schematic overview over the model we use to decribe multiple sequence alignments.

Finally, there is the reward function to finish the definition of the Markov decision process.

$$R(s, s') = R(s') = R(a_1, ..., a_n)$$
$$= \begin{cases} 0 & \text{if } n = 1 \\ -\infty & \text{if } \exists 1 \leq i, j \leq n, i \neq j : a_i = a_j \\ sp(s') & \text{otherwise} \end{cases}$$

where $sp(s), s \in \mathcal{S}$ is defined as the SP-score of the profile resulting from an alignment of the sequence in the order the state s indicates.

This model has some advantages and properties that make the learning process easier. First, there are no loops in the state space as each action adds a sequence to the state that cannot be removed. So, the state-space can be represented as a tree. Second, all actions are applicable in any state. Some of those actions lead to invalid states, but this is penalized by the reward function that fulfills the Markov property, i.e. is memory-less. And third, the Markov decision process is deterministic, which makes it easier to learn an optimal policy.

3.2 Previous Work

In this section, we discuss the work that previously applied reinforcement learning to multiple sequence alignments.

3.2.1 Reinforcement Learning for Multiple Sequence Alignment

The first paper is the most relevant paper on which most of the other papers in this field base [12]. The researchers define the Markov decision process we use in a slightly modified way as we discuss above. This work is already based on another work of the same research group [38].

They use the Q-learning algorithm in its tabular form, which we discuss in the reinforcement learning section and also implement for comparison. As action selection mechanism ϵ-greediness extended with a look-ahead search is used. For the look-ahead search to find the best sequence to align in a state, all possible actions are tested; i.e., they are performed and scored and the best one is selected.

The evaluation of this paper is performed on reference datasets containing DNA sequences. Most of those datasets are from other papers publishing approaches to tackle the multiple sequence alignment problem. The new approach outperforms CLUSTAL and MAFFT based on those DNA references, which is not an optimal setting for CLUSTAL and MAFFT. Those two state-of-the-art references are not designed and optimized for DNA sequences. As explained in chapter two, this might have a huge impact on the performance of the algorithms. Both should also be compared on protein sequence data to obtain a better assessment of the real quality of the reinforcement learning tool. In chapter 5 we will see a comprehensive comparison of reinforcement learning and classical approaches on both DNA and protein data. Unfortunately, this comparison on additional protein sequences cannot be conducted on this paper as the code is not publicly available.

3.2.2 Reinforcement Learning for Multiple Sequence Alignment with affine Gap Costs

This paper is from the same research group that also wrote the paper in section 3.2.1. The major difference is that for this paper the gaps in the alignments are not scored linearly but affine [39]. We discuss the difference between those two scoring systems in chapter 2 in more detail.

The results of this approach improve the results from [12] only slightly. The comparison between the two papers is based on only two datasets of DNA sequences, and no other tool like MUSCLE or MAFFT is used to compare the results against the state-of-the-art approaches [39]. Again, it would be interesting to see how this approach performs on protein data.

3.2.3 Deep Reinforcement Learning for Multiple Sequence Alignment

Based on the work we discuss in section 3.2.1 and in section 3.2.2, Jafari et al. presented an implementation of deep reinforcement learning algorithms. They use deep neural networks with long short-term memory cells as networks in deep Q-learning and an actor-critic approach [13].

The comparison of the results is again performed on the datasets already used in the paper [12]. So, the same problems and drawbacks with the comparion we mention in section 3.2.1 arise again. A second, interesting aspect is the individual performance of Q-learning and the actor-critic approach separately. In the paper [13], there is no clear distinction between the results of both approaches.

And, as above, the code is not available, so one cannot use it for additional testing on protein data or other sequence data.

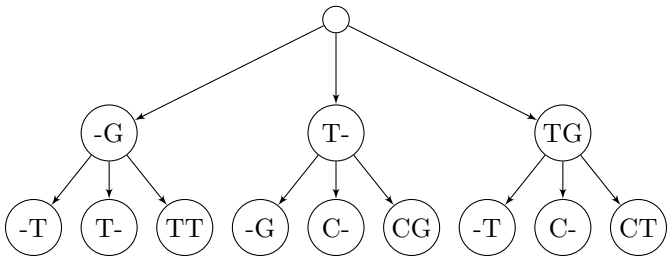
The next two papers we discuss are not based on the model proposed by Mircea et al. but use some different, more complex Markov decision processes to represent the multiple sequence alignment problem and make it “solvable” with reinforcement learning techniques. Both Markov decision processes have the advantage to not care about the aligning of sequences, as we will discuss in the according to sections.

3.2.4 MCTS for the MSA problem

The first of paper with a different model is a special one because it uses neither neural networks nor Q-learning [14]. Instead, Edelkamp et al. use a special form of monte-carlo tree-search. We also use a technique based on Monte-Carlo Tree-Search to solve the multiple sequence alignment problem, but we stick to the model presented by Mircea et al. [12].

Edelkamp et al. present another model to find the optimal alignment. They think of the way to find the optimal alignment as an enumeration of all possible alignments and finding the best one according to a certain scoring function. Since enumerating all alignments is not practical for increasing numbers of sequences, all alignments are represented in a tree. This representation is column-wise, which means, starting with an empty node, the children in this node are all possible alignments continuing in this node [14].

Example 3.2
Schematic first nodes of the search tree suggested by Edelkamp et al. [14]. This example is based on the alignment of two sequences starting with TC... and GT... . The idea behind this tree can easily be extended to alignments of more sequences



The results of this approach are achieved on protein sequence data from BAliBASE version 2. Unfortunately, there is no comparison with other tools like the already named CLUSTAL or MAFFT.

3.2.5 RLALIGN and Exploring Reinforcement Learning Techniques in Multiple Sequence Alignment

The last work we discuss in this section is the master thesis of Ramakrishnan on "Exploring Reinforcement Learning Techniques in Multiple Sequence Alignment" [40] and a paper on the tool RLALGIN that Ramakrishnan implemented in the thesis [41]. We discuss those two topics as one because they are directly related to each other. The work is based on the MDP used in the PHYLO game, which allows people to align sequences themselves to find the optimal one [42]. In terms of the size of the state space, this is even more complex than the one discussed above, but more intuitive.

The model is based on the idea of shifting bases to the left or to the right to form alignments. Bases can only be shifted to a field where a base is located. So, the bases are not allowed to switch their position within the sequence with removed gaps.

By definition, this idea does not need an alignment algorithm as it aligns the sequences on its own, but this increases the complexity of the resulting Markov decision process dramatically as there are many more states and actions and learning becomes much slower than the approach we perform.

Example 3.3
Schematic table of the model inferred from the PHYLO game with arrows representing possible actions. Each row is a sequences, very similar to the model we use. The arrows in the cells represent gaps and how to move the neightboring cells representing nucleic bases or amino acids.

$\leftarrow$	A	T	T	$\leftrightarrow$	G	$\rightarrow$	$-$
$\leftarrow$	A	T	C	C	G	$\rightarrow$	$-$
$\leftarrow$	A	T	$\rightarrow$	$\leftarrow$	G	T	$\rightarrow$

Table 3.1: Exemplary model of the profile we use in the example in chapter 2.

Unfortunately, this idea is only tested on artificial data and not against other tools or on real datasets or benchmarks, so the informative value of the analysis is very limited and should be increased if possible. One reason for this way of testing is that an alignment of a few sequences with about 20 bases is not feasible anymore. As an example, a model for two sequences with 6 bases each, the agent trained 5 hours ([40] p.4). For this tool, only the code for aligning two sequences of length 4 is available. So, we cannot conduct a comparison with this tool.

One interesting idea of this approach is that it can be used for generalization. In the paper, Ramakrishnan et al. applied it successfully to scale from sequence lengths of 6 to 20. A second nice ability of this model is that it can be used to train the agents stepwise. This means it is first trained on smaller problem instances. In the paper, they used two sequences of length 6, trained the agents, and retrained for two sequences of length 10. The results of this do not show any differences to normal training on two sequences of 10 bases each.

Chapter 4

Reinforcement Learning

Reinforcement learning is a field of machine learning that has made significant progress in recent years. The main impulse for the recent rise of interest came from a paper published by Google DeepMind project in 2013 that introduced major improvements [8].

DeepMind is also one of the first that applied deep reinforcement learning with striking success to a problem of bioinformatics. Not to multiple sequence alignment as done is this work, but to the problem of protein folding. It is about the way, how proteins are folded, which is as described in the section on multiple sequence alignment very important for many clinical tasks [11].

Reinforcement learning is beside supervised and unsupervised learning the third big paradigm in machine learning. In contrast to supervised learning, reinforcement learning does not depend on the existence of datasets. Classical tasks for supervised learning are regression and classification on data. The network is learning a function that is used to represent the data. Therefore, a labeled dataset is needed; for example, the MNIST dataset contains 60 thousand images to classify ten different digits. Because of its trial-and-error character, reinforcement learning is different and is not mimicking the dataset labels [43]. As it is often applied to round-based games, reinforcement learning learns from the reward of an action that might not be given immediately [15].

Reinforcement learning is a trial-and-error approach that learns from the returns received from the environment as depicted in figure 4.1. From each action the agent performs in the environment, it observes rewards that are used in reward functions to determine the quality of the selected actions.

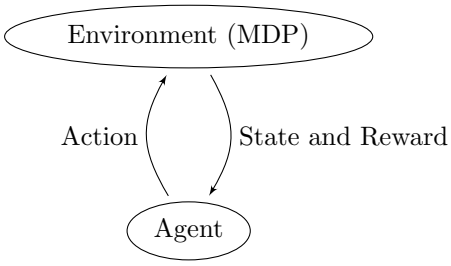


Figure 4.1: Schematic view of a Markov decision process with the interactions between a learning agent and the environment.

For this thesis, we have to distinguish the terms reinforcement learning and deep reinforcement learning. When talking about deep reinforcement learning, we refer to algorithms that use neural networks to learn a policy in an environment. So, tabular approaches and monte-carlo tree-search are the only non-deep reinforcement learning algorithms we use.

As described in the previous chapter, the Markov decision process for multiple sequence alignment is episodic and therefore has a fixed end which means that in this chapter we only discuss the episodic cases of reinforcement learning and its algorithms.

4.1 Reward-functions

Reward functions address the problem of assigning rewards to the actions in the correct amount.

There is a very important difference between the terms reward and return. A reward is what the agent gets from the environment after each action. The return on the other hand is what a learning agent assigns to each step as a performance measure of that action to refine the performance of that agent. The return can be equal to the reward, but can also be a weighted sum of some of the rewards. How exactly the return is computed is described in the following, as there are many different ways.

4.1.1 Monte-carlo Return

First, we discuss the monte-carlo return,

$$G_t = \sum_{i=t}^{T} \gamma^{t-i} R_{i+1}$$

that uses the discounted rewards from all future rewards. So, it is the sum of the discounted rewards up to the actual time step. This kind of return is easy to compute and applicable in almost every setting, but it has some drawbacks. As it sums up to the last timestep, it can only be computed if the episode has ended. That means online learning with monte-carlo returns is not possible, because the return cannot be computed while the episode is still running [7].

4.1.2 Temporal Difference Learning

To make return functions applicable to online learning, temporal difference learning is developed. In TD-learning one has to choose a hyper-parameter n that controls to number of time-steps ahead of an action to take into account to compute the reward for an action following

$$G_t = \sum_{i=0}^{n} \gamma^{i} R_{t+i}$$

.

Here $t + n$ might be larger than T, so R_t for $t > T$ is defined as 0.

The most basic form with $n = 0$ is called TD(0). So, the return is equal to the immediate reward. TD(0) has the advantage of an immediate update compared to MC, but it does not take later payoffs into account. If n increases the temporal difference return approximates the monte-carlo return and takes those mentioned later payoffs into account. The technique of scaling n between 0 and T is called n-step bootstrapping and is used to better graduate between TD(0)-learning and MC-learning. MC-learning can be seen as learning with TD(∞). With this, one can make the learning more online, but one still has to wait until the n steps after an action has passed to update the policy [17].

4.1.3 Lambda Return

Sometimes n-step bootstrapping is not accurate enough, as the return computation is restricted to use integers to grade between the TD(0) and MC. Another technique to finely grade between those two extremes is the use of the lambda-return with or without eligibility traces. Eligibility traces can be used online, even though the computation of the lambda-return would last longer and need more input that only the immediate reward. Since eligibility traces are not used in

this thesis, we do not handle them here, but we use the lambda-return as an alternative return-function for offline learning with

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}$$

and $G_{t:t+n}$ defined as

$$G_{t:t+n} = \sum_{i=1}^{n}\gamma^{i-1}R_{t+i} + \gamma^n\hat{v}(s_{t+n})$$

where $\hat{v}(s_{t+n})$ is the actual evaluation of the state; so, the result of a value-estimation of a learned state value function of the state s_{t+n} [17, 44].

4.2 Learning Techniques

4.2.1 Online and Offline Learning

In reinforcement learning, there are some categories to distinguish the algorithms. The first pair is online and offline learning. In an online learning setting, the agent is updated while the episode is still running. So, while performing actions to create the episode, the agents' policy is changing, which can make the learning much faster, but also more unstable as the policy is changed more often. On the other hand, in offline learning, the agents' policy is updated based on already finished episodes. This setting is more stable than online learning but also slower, as it takes longer to make the same number of update steps as in an online setting. As the episodes in reinforcement learning for multiple sequence alignment are not that long, we use only offline learning [7].

4.2.2 On-policy and Off-policy Learning

The second pair of categories is that of on-policy and off-policy learning. In this distinction, it is not about when to update the policies but how to update the policies. In on-policy learning, the agent uses one policy to learn an optimal policy and to generate the episodes to find the optimal policy. Therefore, the same policy has to be non-optimal in order to be exploratory but is getting optimal. The advantage of on-policy learning is the simplicity as one has to handle only one policy and does not have to take care of how to handle two policies at the same time. The described divergence of the goals the policy is heading to is already the problem with online learning. A policy can not get optimal and at the same be non-optimal to generate exploratory episodes. So, the solution can be off-policy learning, which means using two policies at the same time. One to generate the episodes, the so-called behavior policy, and one to become optimal, the target policy. As it is in later learning states (after numerous episodes) not helpful to learn from nearly randomly generated episodes generated from a naive policy, there are some ways how to update the behavior policy using the target policy. In most of the learning settings, on-policy learning is sufficient and leads to good results as shown in the chapter 5 [7]. Only for deep Q-networks, we use off-policy learning as reasoned in section 4.2.2.

4.2.3 ϵ-greedy learning

Another important learning technique is the ϵ-greedy learning. This is a technique to balance exploration and exploitation as an additional option besides off-policy learning. We use it in two ways.

One is the classical way in deep Q-learning as in the Atari-paper to select a random action not necessarily following the actual policy. But this is only used in value-function approximation methods, as they tend to explore too little. In algorithms that directly approximate the policy, there is already randomness in the action selection process.

The other way to use ϵ-greedy action selection is applicable to all agents and inspired by the reference paper. There is a technique used called a look-ahead search. That means, in each state, all available actions are checked whether they are the best ones, i.e. lead to the highest immediate reward. This is done by scoring the resulting profiles after applying each action, and that action with the highest score is selected [17].

4.2.4 Discount Factor

One last important parameter important to mention is the so-called discount-factor γ, which is used to decrease the reward of the future states to not the stuck in infinite loops when an agent jumps cyclic between two or more states without reaching a goal state. By this discount-factor, the reward of behavior is reduced, and it is optimal to find the shortest path to the solution.

4.3 Value-function Approximation

This section mainly focuses on the details of Q-learning that is implemented in the thesis in two ways. There is an agent using a Q-table to approximate an appropriate policy and an agent using a neural network to approximate the Q-function for faster and better learning. Q-learning describes algorithms used to learn the optimal Q-function and to infer the optimal policy.

We will only approximate the Q-function of a policy, but there is the possibility to also approximate the value-function, called value-iteration. The value-function is optimized using the according Bellman-equation from definition 1.4. After training the function approximation, the actions are selected using

$$\arg\max_{a \in \mathcal{A}} \left\{ \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') \left[\mathcal{R}(s, s') + \gamma \mathcal{V}(s') \right] \right\}.$$

We discuss this algorithm not in further detail, as it is not important for the work and just mentioned for completeness. In theory, this algorithm would reach the optimal state-value function $v_*(s)$ if the value iteration is performed an infinite number of times. So, it is guaranteed to converge to the optimal policy and therefore to optimal behavior if it is trained long enough.

4.3.1 Tabular Approaches

Maybe the most intuitive way to approximate functions is to represent the target function as a table of its inputs. In the previous sections, we used the term Q-learning as learning the Q-function of a policy. There is also an algorithm called Q-learning, which learns a Q-function in an off-policy fashion. But this is not what the thesis makes use of, we use on-policy learning in the tabular case of value-function approximation. The algorithm is called SARSA, a short form of state-action-reward-state-action, as the named terms are the main parts based on which the Q-function is learned [45].

In tabular SARSA the state space and the action space are enumerated and stored in a table where each cell represents the Q-value of the according state and action. As presented in section 4.1, there are several options to compute the reward given an episode. We implement all of them in an offline fashion.

Algorithm 1: Tabular SARSA with different Returns

Input: Learning Rate α, Discount Factor γ, λ-return parameter λ, step-size n
Initialite $Q(a, s)$ random for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
Initialize π ϵ-greedy with respect to Q
for *number of episodes* **do**
  Generate episode s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T following π
  for $t \in [0, T-1]$ **do**
    if *Lambda-Return* **then**
      $G = (1-\lambda) \sum_{n=1}^{T} \left[\lambda^{n-1} \sum_{i=1}^{n} \gamma^{i-1} r_{t+i} + \gamma^n \hat{v}(s_{t+n}) \right]$
    else if *Temporal-Difference* **then**
      $G = \sum_{i=0}^{n} \gamma^i r_{t+i}$
    else if *Monte-Carlo Return* **then**
      $G = \sum_{i=t}^{T} \gamma^{t-i} r_i$
    end
    $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[G - Q(s_t, a_t) \right]$
  end
end

4.3.2 Deep Q-networks

The major drawback of the tabular Q-learning presented above is that the convergence time and especially the needed space increases with a growing state space. This is due to the low sampling ratio compared to the size of the state space. To overcome this challenge one can use neural networks to learn to approximate higher dimensional value functions.

In this section, we reimplement the algorithm presented by Mnih et al. [8]. This is the only algorithm we implemented using off-policy learning and a prioritized experience replay buffer.

The training data are generated by interaction with the environment. As a consequence, the samples within one episode of data generation are correlated. If these correlated data are used together to train the agent, it mights mimic the correlation of the data. This is because the learning techniques assume independent and identically distributed data. To avoid this, one has to break the correlation of the samples. Therefore, a storage, called replay buffer, is used. This store keeps training samples from the episodes generated while training and is used to sample uniform data to learn from [46]. To further improve this technique, the buffer can be prioritized. That means, the single samples saved in the buffer are sampled not uniformly but with a probability relative to the information a sample contains. With this technique, samples that might improve the agent the most are selected more often. So, the samples have to be weighted according to their loss, therefore new samples get high values to be used early for training. After a sample was used for learning, its weight is recomputed according to the loss the sample produced in the learning process [47].

As in the tabular case, also for the deep Q-network algorithm, all three presented reward functions are available.

Algorithm 2: Deep Q-Learning with different Returns, prioERB

Input: differentiable action-value function $\hat{q}(s, a, \boldsymbol{\theta})$, Learning Rate α, Discount Factor γ, λ-return parameter λ, prioritized experience replay buffer, update-count k

Initialize $\boldsymbol{\theta}$ random (e.g. following $\mathcal{N}(0, 1)$) `// behaviour policy`

$\boldsymbol{\theta}^- = \boldsymbol{\theta}$ `// target policy`

Initialize π ϵ-greedy with respect to $\hat{q}_{\boldsymbol{\theta}}$

for *number of episodes* **do**

  Generate episode s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T following π

  for $t \in [0, T-1]$ **do**

    if *Lambda-Return* **then**

      $$G = (1-\lambda) \sum_{n=1}^{T} \left[\lambda^{n-1} \sum_{i=1}^{n} \gamma^{i-1} r_{t+i} + \gamma^n \hat{v}(s_{t+n}) \right]$$

    else if *Temporal-Difference* **then**

      $$G = \sum_{i=0}^{n} \gamma^i r_{t+i}$$

    else if *Monte-Carlo Return* **then**

      $$G = \sum_{i=t}^{T} \gamma^{t-i} r_i$$

    end

  end

  Store tuples of $(s_t, a_t, r_{t+1}, s_{t+1})$ in the replay buffer

  $(s, a, r, s') =$ sample from replay buffer

  $$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha \left[r + \gamma \underset{a'}{max}(s', a', \boldsymbol{\theta}^-) - \hat{q}(s, a, \boldsymbol{\theta}) \right] \nabla_{\boldsymbol{\theta}} \hat{q}(s, a, \boldsymbol{\theta})$$

  if *every k episodes* **then**

    $\boldsymbol{\theta}^- = \boldsymbol{\theta}$ `// update target policy`

  end

end

4.4 Policy Approximation

In this section, we discuss the basics of policy approximation using policy gradients. The basic idea of the algorithms in this section is not to approximate and learn state-value functions or action-value functions that result from a policy. Instead, these algorithms directly learn an approximation of the optimal policy.

Approximating policies instead of value-functions derived from those has an advantage in the model we use. A policy approximation can better approximate deterministic policies, because the algorithms in section 4.3 are always ϵ-greedy and therefore select a non-optimal action with probability ϵ [7].

4.4.1 Policy-gradient Theorem

This theorem reasons why the policy approximation works and gives the formula for the update of the parameters of the policy. The function used for optimization measures the performance of a policy parametrization. This is done using the state value function derived from the policy following the actual parametrization

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0)$$

As mentioned, $v_{\pi_{\boldsymbol{\theta}}}$ is the true value function of the policy $\pi_{\boldsymbol{\theta}}$ determined by the actual parametrization $\boldsymbol{\theta}$. By maximizing J the state-values of the policy are maximized, so the agents observes the maximal reward from the states [48, 7].

The policy gradient theorem gives based on this definition the gradient of the performance J based on the parameters $\boldsymbol{\theta}$ as

$$\nabla J(\boldsymbol{\theta}) = \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a \mid s, \boldsymbol{\theta})$$

Here, $\mu(s)$ is a policy distribution over all states that determines how crucial errors in certain states are [48].

4.4.2 REINFORCE

The REINFORCE algorithm uses the policy gradient theorem to approximate a policy and is the most basic algorithm to use for policy approximation [7]. The algorithm and its update rule for the policies parameters are implemented offline as the episodes in our model of multiple sequence alignments are not long.

Algorithm 3: REINFORCE: monte-carlo policy-gradient Control [49]

Input: differentiable policy $\pi(a \mid s, \boldsymbol{\theta})$, Learning Rate α, Discount Factor γ
Initialize policy parameter $\boldsymbol{\theta}$ random (e.g. following $\mathcal{N}(0,1)$)
for *number of episodes* **do**
  Generate episode s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T following π
  for $t \in [0, T]$ **do**
    $G = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln[\pi(a_t \mid s_t, \boldsymbol{\theta})]$
  end
end

4.4.3 REINFORCE with Baseline

This algorithm generalizes the policy gradient theorem used above to the form

$$\nabla J(\boldsymbol{\theta}) = \sum_s \mu(s) \sum_a [q_\pi(s,a) - b(s)] \nabla \pi(a \mid s, \boldsymbol{\theta})$$

where $b(s)$ a baseline function is that can be a constant or also a function, e.g. a learned state-value function. This baseline does not affect the expected value of the update applied to the parameters but reduces its variance [49]

Algorithm 4: REINFORCE with Baseline

Input: differentiable policy $\pi(a \mid s, \boldsymbol{\theta})$, differentiable state-value function $\hat{v}(s, \boldsymbol{w})$, Learning Rates $\alpha^{\boldsymbol{\theta}}$ and $\alpha^{\boldsymbol{w}}$, Discount Factor γ

Initialize policy parameter $\boldsymbol{\theta}$ random (e.g. following $\mathcal{N}(0,1)$)

Initialize state value weights $\boldsymbol{w}$ random (e.g. following $\mathcal{N}(0,1)$)

for *number of episodes* **do**

  Generate episode s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T following π

  for $t \in [0, T]$ **do**

    $G = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$

    $\delta = G - \hat{v}(s_t, \boldsymbol{w})$

    $\boldsymbol{w} = \boldsymbol{w} + \alpha^{\boldsymbol{w}} \delta \nabla \hat{v}(s_t, \boldsymbol{w})$

    $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta G \nabla \ln(\pi(a_t \mid s_t, \boldsymbol{\theta}))$

  end

end

4.4.4 Actor-critic Methods

Actor-critic methods combine the advantages of policy-gradient methods and value-function approximation by learning both simultaneously. As the name suggests, actor-critic methods can be split into two major parts. First, the actor selects actions according to a learned policy as in policy gradient methods, and second, the critic, evaluating or criticizing the actions the agent performs [7].

Since already REINFORCE with baseline learns two functions (policy and state-value function), one can argue that this is an actor-critic algorithm but we stick to the definition of Sutton and Barto that in actor critics the critic has to influence the updating of the actor directly and not indirect as in the algorithm of REINFORCE with baseline presented above [7].

For completeness, we also implement an actor-critic algorithm, namely the asynchronous advantage actor-critic algorithm presented by Mnih et al. in 2016 [50]. It uses a Q-function and an advantage-function.

Algorithm 5: Advantage Actor-Critic (A2C)

Input: differentiable policy $\pi(a \mid s, \boldsymbol{\theta})$, differentiable state-value function $\hat{v}(s, \boldsymbol{w})$, Learning Rates $\alpha^{\boldsymbol{\theta}}$ and $\alpha^{\boldsymbol{w}}$, Discount Factor γ

Initialize policy parameter $\boldsymbol{\theta}$ random (e.g. following $\mathcal{N}(0,1)$)

for *number of episodes* **do**

  Generate episode s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T following π

  $Q = \hat{v}(s_T, \boldsymbol{w})$

  for $t \in [T-1, 0]$ **do**

    $Q = r_{t+1} + \gamma Q$

    $\boldsymbol{w} = \boldsymbol{w} + \alpha^{\boldsymbol{w}} (Q - \hat{v}(s_t, \boldsymbol{w}))^2 \nabla \hat{v}(s_t, \boldsymbol{w})$

    $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t (Q - \hat{v}(s_t, \boldsymbol{w})) \nabla \ln(\pi(a_t \mid s_t, \boldsymbol{\theta}))$

  end

end

4.5 Tree-based Approaches

One could address Markov decision processes by using planning and do not use learning techniques. One technique that uses planning instead of learning is monte-carlo tree-search. Because its naive form is not powerful enough, we use an improved version.

The basic principle of pure monte-carlo search to find an appropriate action in a given state is to enumerate all possible actions in a state and to simulate random rollouts from the resulting state, to sum up, the final rewards of this, and finally to select the action that maximizes this sum over the rollout rewards [51, 52]. This naive algorithm has two major disadvantages as the algorithm does not balance exploration and exploitation very well. This can be seen in two ways.

Algorithm 6: Vanilla-MCTS

Input: State $s \in \mathcal{S}$, number_of_rounds
Output: Best action in the input state
for $a \in \mathcal{A}$ **do**
  v_a = rollout(s, a, number_of_rounds)
end
return $\underset{a}{max}\{(v_a)\}$

First, as it explores each action the same number of times, actions that would lead to an immediate or at least forced win or lose are rolled out the same number of times as those that do not lead to such an early decision and are therefore more interesting to explore that exploitation of directly finishing actions.

This observation leads to the second drawback. The interesting parts are not well explored, as there is only a fixed number of rollouts for these regions.

4.5.1 UCT

These issues of vanilla monte-carlo tree-search are fixed in the UCT algorithm which stands for upper confidence bounds applied to trees. It uses the technique of upper confidence bounds to guide the tree search into regions where the agent is not so confident about the evaluation of the position. That means when it comes to action selection in a node, the agent does not select randomly but using decision rule balancing exploration and exploitation

$$\underset{a}{max}\left\{\underbrace{v_a}_{\text{Exploration}} + C * \underbrace{\sqrt{\frac{\ln n}{n_a}}}_{\text{Exploitation}}\right\}.$$

Here, v_a is the average reward a node gives, n is the total number of rollouts performed in the whole tree, and n_a is the number of rollouts through this node. The left-hand side of the addition is used to give credit to actions that lead to a high reward that should therefore be more exploited. The square-root gives credit to actions that are only explored a little and one cannot be very confident about the v_a of this node since this might have a higher variance than other nodes. The C in between is a hyperparameter balancing the exploratory, right side, and the exploiting left side. This formula defined by this formula is called the tree policy as it selects nodes as long as the search is within the explored tree [53, 54].

During a search, the algorithm builds up a search tree based on the nodes that the algorithm explored. In the implementation, a node has three major states. First, it is the child of an already explored node, so it exists but is not explored. As soon as the search selects this node (it maximized the formula above for its parent node against its siblings) it is set to be explored with the next (in total second) visit. Whenever the search in a simulation reaches a leaf node

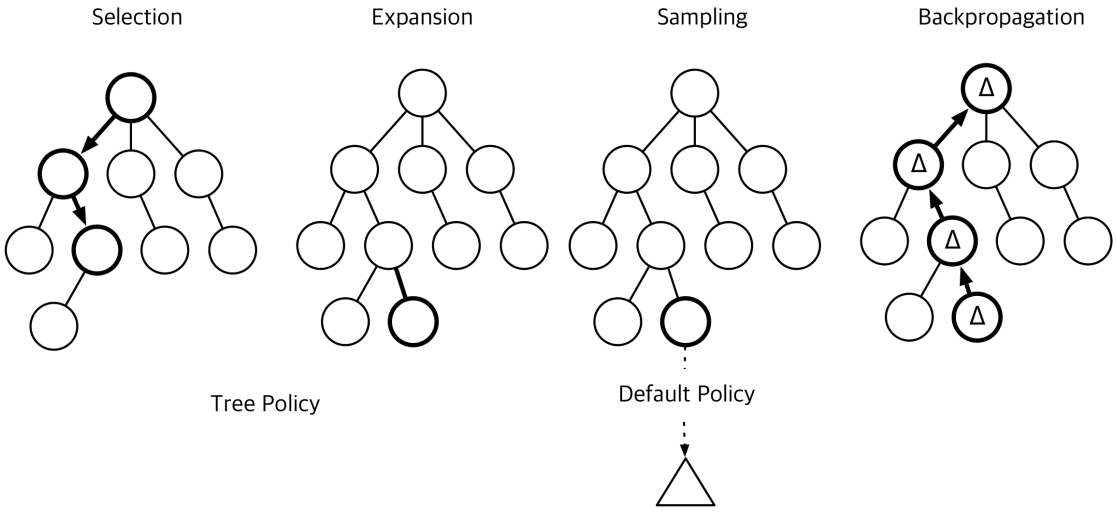


Figure 4.2: Schema of UCT algorithm with a little different naming. The default policy in this figure refers to the rollout policy in out terminology. Also the Sampling is a new phrase, we call it Rollout.

that cannot or is not to be explored, a rollout is performed. That means a default or rollout policy takes action as long as there are actions to select, a terminal state is reached or another stop criterion is fulfilled. This rollout policy takes over from the policy as soon as the search leaves the explored, known tree [53, 54].

Challenges with SP-score

The UCT-formula has a drawback we see in the next chapter. The exploitatory part of the UCT rule, the first summand, is very sensitive to the value of the SP-score. The exploratory part, the second summand, is designed to be in the interval [0,1]. Because we chose $C = 1$ for the hyperparameter, the formula is takes often a value that is highly influenced by the average SP-score (v_a). So, if one path seems to be clearly better than another, this path is always selected, even though it might not be optimal. This means the exploration in the UCT rule is dominated by the expected return. In the first part of the DNA reference comparison, this effect is not that visible, because either the search spaces are very limited or the order does not matter that much as in the hepatitis C virus dataset. To fix this issue, we adjust the SP-score according to estimated bounds, following the idea of Schadd et al. [55].

The adjustment works better as the results get improved by the estimation of a minimum and a maximum. To estimate the maximal score, we use the center-star algorithm as described in chapter 2. As it is proven that the center-star score is at least half as good as the optimal alignment with the scoring function; we multiply the score with 2 and get a correct upper bound. The minimum is then estimated as the minimum score of 10 alignments following random permutations of the sequences.

Chapter 5

Results

In this chapter, we discuss the results of the algorithms we implement in this thesis. First, we compare the algorithms based on the sequence data from the paper of Mircea et al. [12]. This comparison is extended with the newest versions of CLUSTAL, MAFFT, and MUSCLE.

In a second step, we discuss results computed by the three tools compared to a deep Q-learning agent using the λ-return based on sequence data from the BAliBASE database in version 4.0 [56].

We chose the deep Q-learning agent because it is fast and performs well on the DNA sequence data as discussed in section 5.2. Out of the three possible reward structures for deep Q-learning, the λ-return is chosen because this return is not used in any other approach.

The reason for the second comparison is to get a wider overview of the performance of reinforcement leanning algorithms compared to state-of-the-art tools. The second reason for the comparison on BAliBASE is the type of sequences. BAliBASE is a database of protein sequences that introduces a new aspect into the comparison.

5.1 General

All comparisons are done using both scores, C-score and SP-score. Besides that, we test the alignments of BAliBASE sequence alignments also using Q-score and TC-score. Unfortunately, this additional evaluation is not possible for the DNA reference data as there are no "correct" alignments of those sequences given in the paper [12].

5.1.1 Sequence Identity

As we will use the term sequence identity, we briefly discuss how it is defined in this thesis. As in sequence alignments, we want to find the maximal number of equal bases or amino-acids between two sequences. For that propose, we can apply the Needleman-Wunsch algorithm with a specific scoring function. The modified SP-score for this task is defined as

$$sp\left(c_i, c_j\right) = \begin{cases} 1 & \text{if } c_i = c_j \\ 0 & \text{otherwise.} \end{cases}$$

The final score gives the maximal number of perfect matches between the two considered sequences. This is guaranteed by giving only credit if the compared components of the sequences are the same. Remember, the case of $c_i = c_j =$ "-" does not need to be considered, as it is

forbidden to have such columns by the rules of alignments. To make a comparable score out of this absolute number, we divide it by the length of the alignment. The identities of a sequence file are given in a matrix that in theory should be symmetrical. Due to the way we implemented the pairwise sequence alignment, this cannot be guaranteed.

5.1.2 Databases

We now have a look at the two main databases used for sequence comparison in this thesis, the BAliBASE [56] and the OxBench dataset [22].

BAliBASE: This is a benchmark database build up in several subsets. These subsets contain groups of sequences with similar properties, like one orphan sequence in a group of similar sequences (reference set 2). The third reference subset, for example, contains many sequence families to be aligned within one sequence file. In total, there are 10 reference sets of which we use the first (sequences of variability in identity and length), the fourth (sequences with N-terminus and C-terminus extensions), and the fifth (insertions in the sequence, so the sequences are of very different sequence lengths). For each sequence file in BAliBASE, there is always a hand-crafted and validated solution containing the biologically correct alignment of the sequences [57]. From all reference sets of BAliBASE we focused on the first five, because those are included in BAliBASE since the first version and are also used in other papers for evaluation [6, 14]. From those five sets we selected all sequence files with at most ten sequences and an average length of the sequences of at most 500. These bounds are chosen because on problem instances with longer sequences or more sequences the execution of reinforcement learning algorithms takes too long. As a result, we do not use the reference sets two and three because their sequences files were filtered out by the bounds.

OxBench: The OxBench dataset is an alternative dataset of multiple sequence alignments. The multiple sequence alignments in this dataset are extracted from the 3Dee database of structural domains and after some preprocessing steps grouped into different sets that have certain properties in common. Examples are small families, pairwise or multiple family alignments where two or more groups of sequences have to be aligned. The OxBench database focuses on the aligning of families and not that much on different factors that influence the alignment quality as the BAliBASE does in its subsets for extensions, insertions, or orphan sequences [22].

5.1.3 Compared Tools

Second, we will shortly go through the settings of comparison aside from the databases, i.e. the tools used as references and the algorithms we implement in this thesis.

CLUSTAL: Version 1.2.4 from December 16th, 2016 (Clustal Omega). As for the other three tools, no parameters except the input and output are provided to the program. So, it is executed in default mode.

MAFFT: Version 7.271 from July 3rd, 2020. This tool is also used in default mode, so with no options set that influence the search. In this case, MAFFT selects the fast and progressive *FFT-NS-2* option ([5] p.4).

MUSCLE: Version 3.8.31 [6]. Finally, for MUSCLE we specify no search-influencing parameters and run it in default mode.

DRL-Tools: In the first step, we use ten different reinforcement learning agents or reward structures to find the best one on the reference data. In a second step, the best agent from those 10, namely the deep Q-learning agent using the λ-return, is used to compute the alignments on BAliBASE test files. The results are then compared to the alignments of CLUSTAL, MAFFT, and MUSCLE.

Networks: The networks we used for the deep reinforcement learning agents are mostly the same over all agents. The basic structure is a feed-forward neural network with three hidden layers of sizes 256, 128, and 64. The size of the output layer is adapted as needed. If the networks approximate a state-value function, it is only one output node. In case of an action-value function or a policy to approximate the network has as many output nodes as sequences in the problem instance. The activation functions in the nodes differ. For value function approximating networks, the first layer and the third layer use ReLUs, whereas the second layer uses a Softmax function. For policy networks we use ReLU in the first two layers and Softmax in the last layer and the output layer.

For deep Q-learning we use the action-value function networks, for the REINFORCE implementations and A2C we use a policy network and a state-value function network as the baseline.

5.1.4 Parameter Settings

The ten agents include 3 with different reward structures for the SARSA algorithm and the deep Q-learning, respectively. The configurations for deep Q-learning using neural networks are the same, except that a neural network replaces the Q-table in the SARSA-algorithm. For both, we use $\alpha = 0.1$, $\gamma = 0.9$, and search support, which means the agent can only select actions that lead to valid states. So, sequence reselection is disabled on the sequence selection level and must not be learned. In the case of TD-learning, we use 0-step TD-learning (TD(0)). For the λ-return, we use $\lambda = 0.1$. In addition to this and to speed up the learning process, in the beginning, ϵ-greedy action selection is used. This ϵ-greediness starts with $\epsilon = 1$ in the first action selection and within the first 10% of the learning process reduced to $\epsilon = 0$. So, the first sequence is random, whereas the major part of the learning is depending on the learners' action selection.

Besides these six function-approximating agents, three policy-approximating agents are tested, namely REINFORCE with and without a baseline and the actor-critic algorithm (A2C) discussed in chapter 4. In those three algorithms we use $\alpha = 0.01$, $\gamma = 0.99$, and search support as in the function-approximation algorithms. For REINFORCE with baseline and A2C, the baseline or critic is implemented as a neural network approximating the state-value function.

As a computer to execute all programs, we use an Ubuntu Server, Version 18.04.4 LTS (Bionic Beaver) with 8 kernels and 4 GHz clocking.

5.2 DNA Datasets

In this second section, we discuss the results of our implementations compared to the references from other reinforcement learning approaches and compared to the three classical tools. These datasets are taken from the reference paper of Mircea et al. [12].

The tables A.4 and A.5 show the scores of progressive alignments based on the DNA datasets optimizing for both scores.

As we see, the reinforcement learning agents are mostly equally good. This is also visualized in figure 5.1. The equal performance is because of the way all of them work like. An additional factor is that the number of sequences in the sequence files is low. In the beginning, the agents are choosing random actions guided by a policy to be improved while learning. So, the sequences chosen in each step are nearly random, and the exploration of the search space is performed very widely. As for 3 sequences, there are only 6 possible orders to align the sequences. Those are quickly visited so that the search and therefore the learning is stopped. We discuss this in more detail in section 5.4. The scores of the brute-force alignments support this hypothesis of seeing in the entire search space and selecting one optimal order. This observation holds for both scores, the C-score and the SP-score.

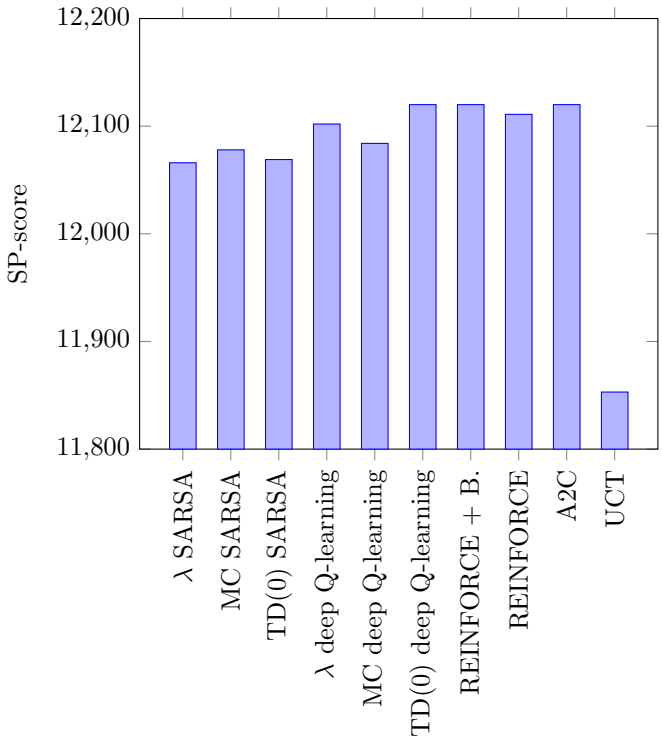


Figure 5.1: Comparison of the SP-score of different agents on the OxBench 429 dataset. The low SP-score of the UCT agents is handled in section 5.2.

The three datasets with more than 3 sequences are the OxBench dataset 429 containing 12 sequences, the hepatitis C virus dataset (10 sequences), and the papio anubis dataset (5 sequences). The search spaces for the first two of those are not entirely visited as those are too large. The search space of papio anubis is special, it consists of 120 different profiles and is explored by some algorithms as we see in table A.33. Nevertheless, for the hepatitis C virus and papio anubis alignments the agents perform equal because the sequences have identities above 90% (see table A.1 and table A.2), so the order of how to align the sequences does not matter that much.

The small differences in the C-score of alignments might arise from rounding errors. This can be seen for the alignments of papio anubis sequences, dataset 1, and OxBench 429 in table A.4.

SP-score Comparison

Comparing the SP-scores for the DNA datasets to the reinforcement learning algorithm from Mircea et al. [12] is difficult because they and Jafari et al. [13] did not optimize for the SP-score. So, we look at the SP-scores of the tools optimized for the C-score. From this, it is obvious that the reinforcement learning agents optimizing the SP-score outperform the SP-scores of C-score-optimized alignments.

To get a fair comparison for SP-score, we can additionally look at the SP-scores of the agents optimized for the C-score. Those scores are given in table A.6. The SP-scores of our agents are similar to the scores we get when optimizing for the SP-score. This is due to the fact that the

datasets contain only 3 sequences. So, the search space can be visited entirely. Another factor is that the not optimized score is used as a tiebreaker in case of an equal main-score. That means, if there is an alignment order that optimizes both SP-score and C-score, this is the optimal alignment for both settings even though there are other orders that lead to the same optimal main-score. Especially for the OxBench 429 data, it is interesting to see that there seem to be multiple orders to align the sequences leading to the same C-score but highly different SP-scores. This can also be seen in figure 5.2.

Additional Data

For the second part of this first comparison and to get a slightly more differentiated view on the comparison, we chose two more files of sequence data from the OxBench Database and three files of protein data to get a first impression of how reinforcement learning agents perform on protein and other DNA data. The results of this can be seen in the tables A.7 and A.8.

Also in these comparisons, the reinforcement learning agents outperform the three classical tools: CLUSTAL, MUSCLE, and MAFFT. And again, all algorithms perform nearly equally. The differences in SP-scores between the reinforcement learning agents and the classical approaches can be explained easily. The state-of-the-art approaches also optimize an SP-score, but this SP-score is computed more gradually and biological more useful than our score. Some additional features we discussed in chapter 2.

MCTS

In figure 5.1 we see a worse performance of the UCT agent compared to all other agents. This problem can be found in other comparisons as well. The scores drop for 3 out of 5 sequence files by at least 180 points (10% of the total score). An approach to overcome this challenge is discussed in section 4.5.1.

The comparison of the results between the normal and the adjusted UCT agents can be seen in table 5.1. The quality of the resulting alignments is not optimal, especially the alignments for the OxBench datasets 641t2 and 34. This smaller gap in the performance between the adjusted UCT and the optimal progressive alignment might be due to the quality of the bounds. If the center-star finds an optimal or nearly optimal alignment for progressive alignments, the upper bound with being twice the score, and the range [0, 1] is nearly reduced to [0, 0.5] which shrinks the performance of the UCT agent.

5.3 Protein Datasets

Already in the second comparison step of the previous section, we discussed some protein results and they look promising. Deep reinforcement learning outperformed the three classical approaches on the data. To validate and double-check this result, we perform a more comprehensive analysis on more protein data from the BAliBASE.

The presented approach for the multiple sequence alignments takes much more training time than the classical approaches as we will discuss later in this chapter. Because of this longer computation time, we analyze only those sequences from the first 5 reference sets that have less than 10 sequences and an average number of bases or amino acids below 500. Which datafiles these are can be seen in tables A.9, A.10, and A.11.

The comparison in this field will be a bit different from the one before. Instead of analyzing each sequence file on its own, we average in groups of sequence files. In these groups, we compute the average score for each agent. As the computation of these alignments takes much longer than

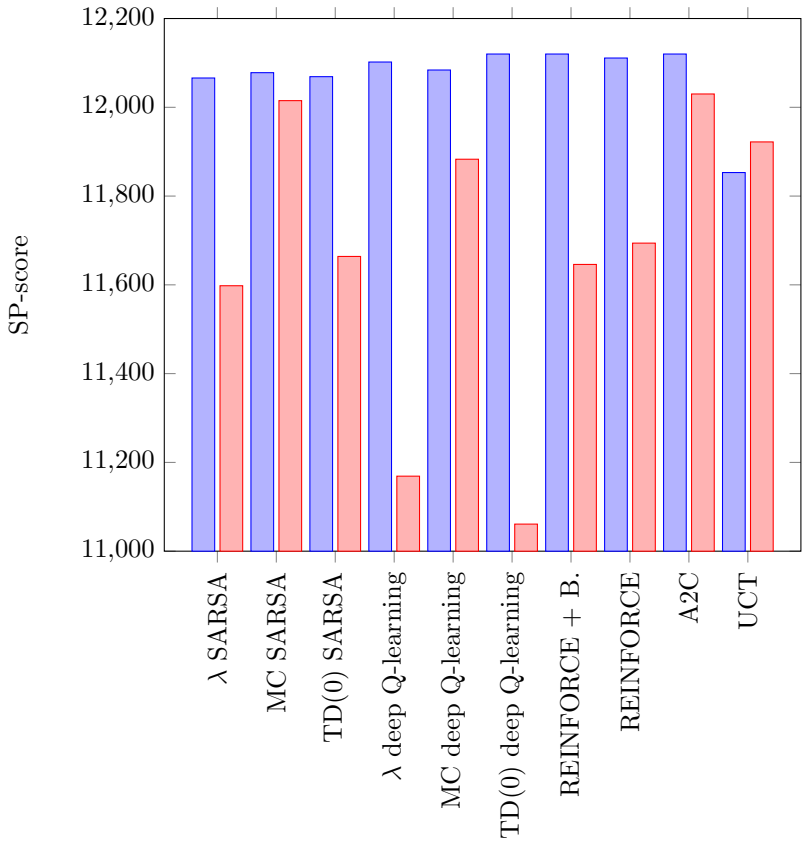


Figure 5.2: Comparison of the SP-score of different agents on the OxBench 429 dataset. The SP-scores of agents that optimize the SP-score are shown in blue, the SP-scores of agents that optimized the C-score are shown in red.

for the few sequences above, we only look at the performance of the deep Q-learning agent using the λ-Return and $\lambda = 0.1$.

In the following, we discuss the average scores for the different groups within the BAliBASE dataset. The ratio columns in the tables give information about how many alignments one of the reinforcement learning algorithms computed better than the classical approaches. "Win Ratio" gives the ratio of performing better then all the classical tools. The "Draw Ratio" gives the ratio that one of the deep Q-learning agents performs better than at least one of the classical tools.

The groups are divided as in the BAliBASE Benchmark. Additionally, we divide the reference sets 1.1 and 1.2 into two sets of sequence lengths below 100 nucleotides and a set of sequences up to a length of 500 nucleotides. The sequences in the reference set 1.1 have an average identity level of 20-25% per database file, whereas the sequences in 1.2 reach a level around 25-30%.

In the tables A.12 and A.13 the comparison of the average Q-scores and TC-scores are denoted. These scores are not optimized by the agents, as these scores can only be computed knowing the optimal alignments of the sequences. The classical approaches use special adaptions of the SP-score that is biologically optimized to mirror the optimal alignments. In contrast, we use a simple scoring function that gives only credit for matches, mismatches, and gaps. We discuss ways how this could be improved in the chapters 2 and 6, and also in the next chapter.

Benchmark	Data				References	
	Type	# Seqs	ØLen	BF	Normal	Adjusted
Hep-C-Virus	DNA	10	212	-	18627	18627
Papio Anubis	DNA	5	1092	18881	18848	18875
Oxbench 469	DNA	3	332	672	639	672
Oxbench 429	DNA	12	171	-	11853	12006
LGM-Dataset	DNA	3	39	348	342	348
RLO-Dataset	DNA	3	93	488	488	488
Dataset 1	DNA	3	129	167	167	167
OxBench 414	DNA	6	154	1974	1743	1965
OxBench 415	DNA	4	126	603	603	603
OxBench 433	PEP	3	186	353	353	307
OxBench 641t2	PEP	3	271	1053	870	887
OxBench 34	PEP	6	215	1490	1147	1208

Table 5.1: SP-score comparison of UCT agents for normal setting and an adjustment of the SP-score to be scaled to a range between 0 and 1.

So, it is not surprising that reinforcement learning approaches perform dramatically worse than CLUSTAL, MUSCLE, and MAFFT.

The results comparing the SP-scores and C-scores, namely the tables A.14 and A.15, of the agents are way better than the comparison of Q-scores and TC-scores. But when we look at the average SP-scores in the reference sets 4 and 5, we see a dramatic difference between the scores. Remembering the results from the reference sets, we did not expect such major differences.

Also, the C-score comparison shows that C-score optimizing agents performs a little better than the SP-score optimizing agent, but is still worse than the classical approaches. Here, the differences are smaller than in the SP-score comparison. A reason for this might be the inappropriateness of the C-score, especially for protein alignments. The more sequences an alignment contains, the more mismatches per column arise just because more sequences might introduce a mismatch in a single column. This means, the SP-score evaluates the alignment quality more graduated.

Unfortunately, in this comparison not all files from the BAliBASE reference sets are tested. This is because of the length and count of the sequences which both slows the alignments extremely down. So, the runtime of our algorithms is too long. We analyze this in section 5.4.

5.3.1 Alignment Analyse

To find reasons for the gap in the performance between the reference set of sequences and the BAliBASE alignment, we take a closer look at the alignments for some of the datasets found by the classical approaches and by the reinforcement learning agents.

From the tables A.9, A.10, and A.11 in comparison to table A.3 we see that in BAliBASE, the sequence lengths within a sequence file vary much more than in the DNA data form Mircea et al. [12]. This can easily be seen from the "Range"-column in each table. For the DNA data the range is on average 15, mostly influenced by three benchmark files. The median range in this set is 6. Out of all BAliBASE sequence files, we analyze only one file that has a range below 6. A second very good measure for the high divergence of sequence lengths is the standard deviation of those sequences. The standard deviation of the sequences sizes within a file is also much higher than in the DNA sequence set (around 50 (reference set 1.1), 80 (reference set 1.2),

and 40 (reference sets 4 and 5) in the BAliBASE reference sets compared to 5 in the reference data from OxBench [22] and Mircea at al. [12]).

For the analyzed alignments, we provide the identity matrix in the tables A.17 to A.29. Those sequence files are chosen based on their properties of the sequences and the compared performance of the algorithms. We will look at files with better alignment by reinforcement tools as well as from the classical tools. But we also investigate the structure of the sequences as the reference sets of BAliBASE distinguish them well into different groups that share important properties. The analyzed sequences are from 3 different reference sets, set 1 which is itself divided into two separate sets of different identities, $\leq 25\%$ (RV11) and $\geq 25\%$ (RV12).

When we refer to the term "structure to align", we mean, that the identity matrices show subgroups of the sequences that have higher internal identity. Examples for this will be marked in the tables. This is based on the idea, that similar sequences should be aligned first which is, for example, used in the scoring function of CLUSTAL [4].

Reference Set 1.1

The insights into the actual alignments help to understand how the sequences are aligned and what ideas for further optimization are. On the sequences from BB11001 (A.17 and figure 5.3), BB11002, (A.18), and BB11003 (A.19) the classical approaches outperformed the reinforcement learning agents by far. From the identity matrices, we see that none of the three matrices indicate a guide tree to use for alignments. For the first two alignment trees, no structure in the alignment trees can be seen. The same holds for BB11025 (A.21). The table indicates no clear structure on how to align the sequences in the matrix. So, the way the sequences are aligned looks similar to the first three datafiles above.

What is striking about the three problems is that in the alignments of the reinforcement learning agents the gaps are heavily split. This means there are many small, even only one or two positions long gaps, and then only two or three bases come before the next gap opens. This phenomenon can be seen in nearly all files from BAliBASE we analyze. The reason for this is simple and rooted in the very simple SP-scoring function we use. We use a formula with linear gap costs and not with affine costs. The difference is the weighting of the gaps based on their position within an entire gap, and we discussed this in more detail in chapter 2.

In the example of BB11001 (see figure 5.3), we see that the state-of-the-art agents outperform the reinforcement learning agents on the Q-score and TC-score, whereas the latter agents perform far better on the scores they are optimized for. This is not always the case as we see the tables A.14 and A.15.

The next problem instance we take a closer look at is the file BB11013. In this file, the C-score optimizing agent performs second-best in terms of the Q-score and was the best in terms of the TC-score. Again, there is no indication for an alignment order based on the sequence identities. This file has 5 sequences of an average length of 71 with one long sequence. The C-score agent aligns the longest in the second step and the classical tools even in the first. Contrary to the C-score optimizing agent, the SP-score optimizing aligns the longer sequences at the end. So, it seems to be a good choice to align longer sequences at first or at least early.

So far, we analyze alignment from the reference set RV11 with low sequence identity. In the next set, RV12, this similarity is a bit higher. As we can see from the tables A.12 and A.13 the Q-scores and TC-scores are also much higher. So, the algorithms, classical and reinforcement ones, can handle them better.

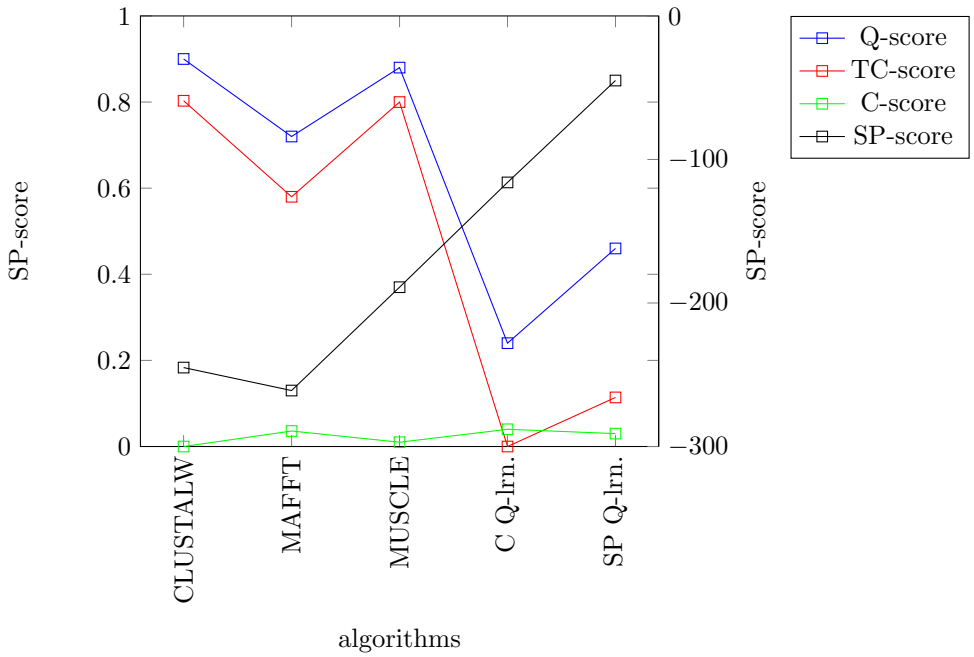


Figure 5.3: Comparison of the four scores on all agents for BB11001.

Reference Set 1.2

The first file we analyze in more detail is BB12020, table A.22, with only four sequences and a similarity matrix that indicates to align the first two and the last two sequences in the first step and then both profiles against each other. While the classical tools do so, this type of alignment is not possible for the reinforcement algorithms as they can only align one sequence against a profile, which results in stairs in the alignment as discussed in chapter 2. But both reinforcement learning algorithms we use align the sequences 3 and 4 in the first stepfollowed by the two other sequences. Astonishingly, the alignments of the reinforcement algorithms are more compact than in the reference set 2 and perform clearly better than CLUSTAL, which aligns the two groups described above in a shifted way with large gaps at the beginning of the end of each pair of sequence.

In the BAliBASE files BB12024 (A.23) and BB12040 (A.25), we see similar behavior to the first analyzed reference set. The reinforcement algorithms introduce way too many small gaps comprising only one or two gaps. So, the classical tools outperform the reinforcement learning algorithms by far. In both, the similarity matrix introduces no clear structure on how to align the sequences.

This is different in BB12036 (A.24 and figure 5.4). In this file, the sequences 1-3 and 5-7 are more similar in their smaller groups and therefore CLUSTAL and MAFFT align these subgroups at first and merge them than with the remaining sequence in a second step. A similar issue we have in BB20020, where two subgroups with higher internal similarity can be identified. This alignment in two subgroups can not be mirrored by the reinforcement learning alignments. Because of the used MDP the agents are not able to build complex trees but simple orders of how to align sequences.

As in the first reference subset, we see in this second reference subset that the classical

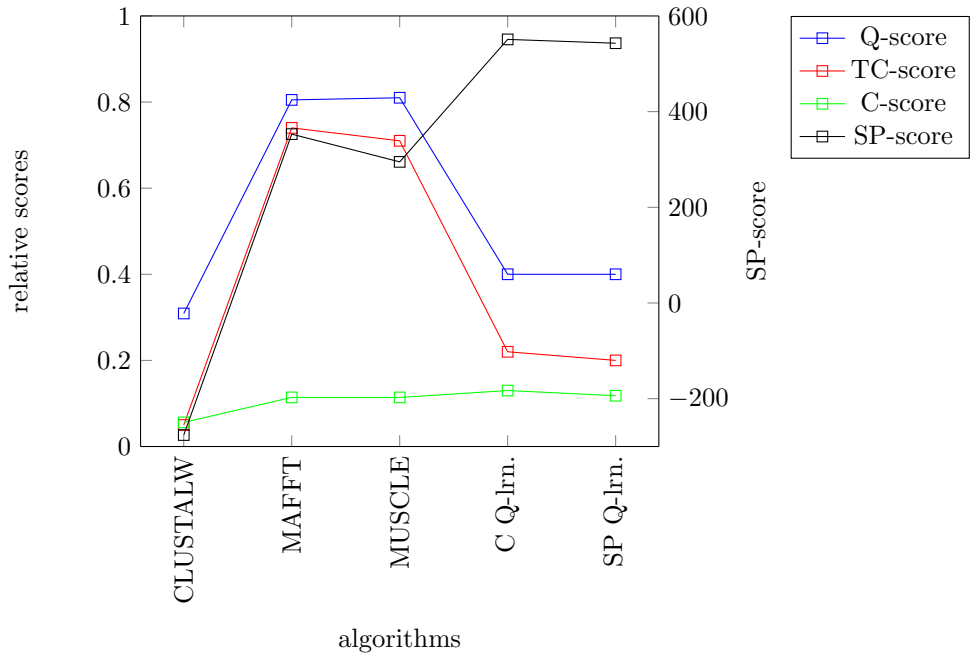


Figure 5.4: Comparison of the four scores on all agents for BB12020.

approaches perform better than the reinforcement learning algorithms. The primary reason for this is the SP-score we use to optimize the alignments. This scoring uses linear gap costs and not affine gap costs. It might improve the results very much to adapt the scoring in a way that uses affine gap costs. In the following, we inspect the only alignments from the reference sets 4 and 5 we computed for this analysis.

Reference Set 4

The first two files from reference set 4, BB40010 (A.26) and BB40014 (A.28), are similar. As all sequence files in this reference set, they comprise a few long sequences and multiple smaller ones. Compared to the shorter sequences, the longer ones have extensions at the N-terminal and C-terminal ends of the sequence, so the smaller ones are aligned in the middle of the longer sequences. In the classical approaches, the longer sequences are mostly aligned at the end, whereas the reinforcement algorithms align them earlier. And as in the previous reference sets, similar sequence groups are aligned early in CLUSTAL, MUSCLE, and MAFFT whereas the reinforcement learning algorithms cannot find their groups and disregard them. As a result, they perform very poorly and cannot compete with classical tools.

The last alignment from the reference set 4, BB40018 (A.28), is different as two longer sequences are aligned offset. Additionally, the similarities between the sequences are low, so there is no sign for a grouping. Again, we see many smaller gaps in the alignments of the reinforcement learning algorithms.

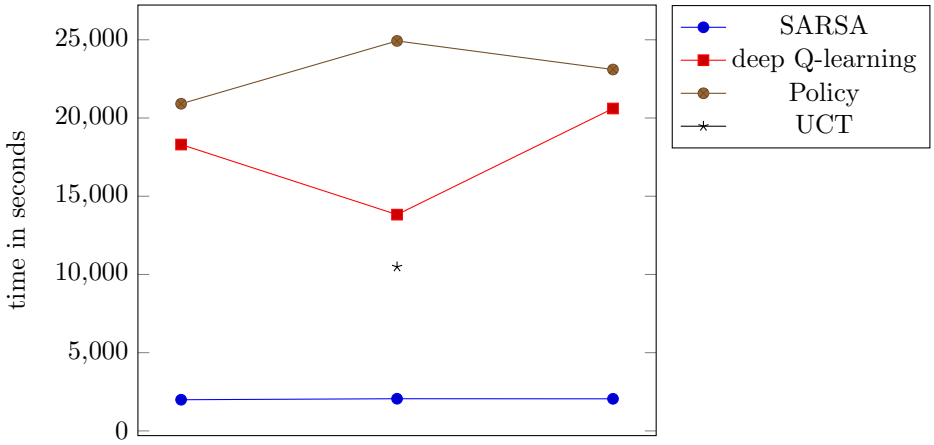


Figure 5.5: Exemplary comparison of the runtime of the different agents on the OxBench 429 dataset. The lines of the SARSA agents and deep Q-learning agents show from left to right the runtime using the λ-return, the MC-return and the TD-return. The policy curve shows from left to right the runtime of REINFORCE with baseline, REINFORCE without baseline and A2C.

Reference Set 5

Other than in reference set 4 the sequences from reference set 5 have internal insertion instead of N-terminal and C-terminal extensions. So, the smaller sequences have to be aligned with a larger gap in the middle, whereas the longer sequences define more or less the length of the alignments. The only alignment in this section we look at is BB50004 (A.29). From the table, we see that there are two subgroups to align first because of their higher internal similarity. Those groups are sequences 1, 3, and 4, and sequences 2, 5, and 6. The classical alignment algorithms perform again clearly better, and the known problem of many small gaps occurs again in the reinforcement learning alignments.

The most mentioned problem with the alignments is the gaps that too numerous and too small to have a reasonable biological meaning. The solution to use might be affine gap costs that should merge many small gaps into a few bigger ones. Something similar has been used by Mircea et al. for their alignments on the few DNA datasets [39]. We assume major improvements while the paper shows only minor improvements. The reason for our view is the difference between the sequences we want to treat with this alternative gap handling. As mentioned earlier in this chapter, the lengths of the DNA sequences within one sequence file do not vary that much compared to the protein sequences. So, in alignments, fewer gaps are inserted, which means the handling of the gaps does not change the resulting alignments that much as in alignments with more gaps. On the other hand, the protein sequences contain more gaps in their aligned form and have more options to place the gaps.

5.4 Runtime

The final aspect of this analysis of the reinforcement algorithms is the runtime of the algorithms. In contrast to tools like CLUSTAL, MAFFT, and MUSCLE, the presented agents perform many more alignments. The reason for those additional alignments is rooted in the idea of reinforcement learning and its trial-and-error character. To learn an agent has to train for many episodes in

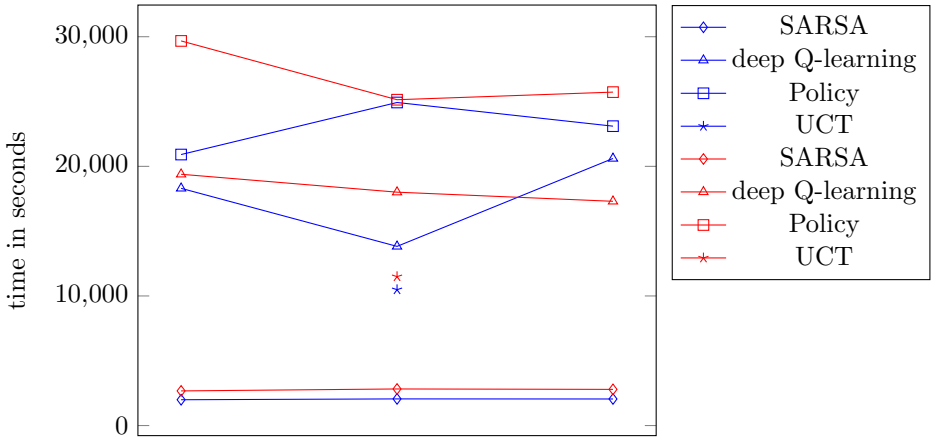


Figure 5.6: Exemplary comparison of the runtime of the different agents on the OxBench 429 dataset. The red lines show the runtime optimizing the C-score and the blue lines show the runtime when optimizing the SP-score.
The lines of the SARSA agents and deep Q-learning agents show from left to right the runtime using the λ-return, the MC-return and the TD-return. The policy curve shows from left to right the runtime of REINFORCE with baseline, REINFORCE without baseline and A2C.

its environment. Transferred to multiple sequence alignment, the agent has to perform many alignments to find out which are good and which are not that good.

The resulting runtimes can be seen in table A.30 and also in figure 5.5 for optimizing the C-score and in table A.31 for optimizing the SP-score. The runtime differs not very much between the two different scores to optimize as exemplary shown in figure 5.6. As explained, the reinforcement learning algorithms are much slower than the state-of-the-art tools. And compared to the references we discussed in chapter 3, the runtimes are similar, but difficult to compare. Mircea et al. stop their time measurement as soon as the convergence arise. So, not the complete training is conducted but only a few episodes until a stop criterion that depends on convergence was fulfilled [12].

The algorithms we implement train for 5000 episodes or until all possible alignments have been found. The additional criterion introduces a more practical aspect, as learning is not necessary anymore if already all possible alignments have been found. From this some discrepancies arise, e.g. the deep Q-learning agents show highly different runtimes on the LGM dataset and on the RLO dataset. This is because the agent found the last alignment very late. If this remaining alignment is found at the end of the training the training is conducted much longer compared to trainings where all possible alignments are found early in the training process.

This phenomenon explains the different performances on the smaller datasets with 3 sequences. Differences in runtime on the three larger datatsets can also be explained in a very similar. For the three datasets with over 3 sequences, we additionally analyse the explorations of the different agents. The tables A.32 (hepatitis C virus), A.33 (papio anubis), and A.34 and figure 5.7 (OxBench 429) show the number of entries per permutation length. So, how many different permutations of a certain length an agent has analyzed during the learning process. For the datasets with 3 sequences this would not be informative, because all agents find all permutations and the tables contain no information about the point in time in the learning process when the permutations are found. The minimal number of sequences in a profile shown in the

tables is two because at least two sequences are needed to perform an alignment. The maximal number is one less than the maximal number of sequences that can be aligned in a dataset. This is a trick we use in the implementation because the last number in an order of numbers is clear and therefore neither need to be learned nor need to be specially selected. So, we get rid of a not necessary learning step and have smaller tables and networks.

From those three tables, we get some interesting insights. First, we see that the SARSA algorithms are less exploratory than the other algorithms and also deep Q-learning explores not that much. On the other hand, the three policy agents are much more exploratory. The algorithms for the policy agents are because of the action selection mechanism more exploratory than SARSA and deep Q-learning, because policy agents estimate probabilities for the actions and select along them, whereas SARSA and deep Q-learning selects actions greedily after ending of the ϵ-greediness. Also, the UCT agent is also straight in its action selection. This can also be seen in figure 5.8. The number of profiles explored in an UCT execution is between the values for deep Q-learning and policy approximation. Second, there is no notable difference in exploration between the two scores to optimize. In neither algorithm, the degree of exploration differs by more than the factor 2 and is mostly equal between the different optimization settings of the same algorithm. And finally, the policy agents converge for longer sequences to maximum of 5000 and 6000 different profiles that are explored during the search. The reason for this is simple, because the algorithms are only allowed to perform 5000 or 6000 alignments, respectively. That this bound is reached might be an indication to rise the number of alignments to perform in the training so that the policy agents can explore the search space with larger profiles better.

From these results, we can see that algorithms that use are good to optimize a specific score and can outperform the classical algorithms such as CLUSTAL, MUSCLE, and MAFFT based on these scores. They are also good, when the order of how to align those sequences does not really matter because they are similar or of equal length. But for more complex alignment settings like multiple sub-families within one alignment or highly divergent lengths as we see in BAliBASE, the algorithms do not perform that well and can find only a few better alignments than the classical approaches. Also, evaluating a score that the agent is not optimized for is not a pleasant setting for reinforcement learning (i.e. evalutating Q-score or TC-score while optimizing for the SP-score or C-score). So, there is much work to do until the reinforcement learning alignments compete with state-of-the-art tools.

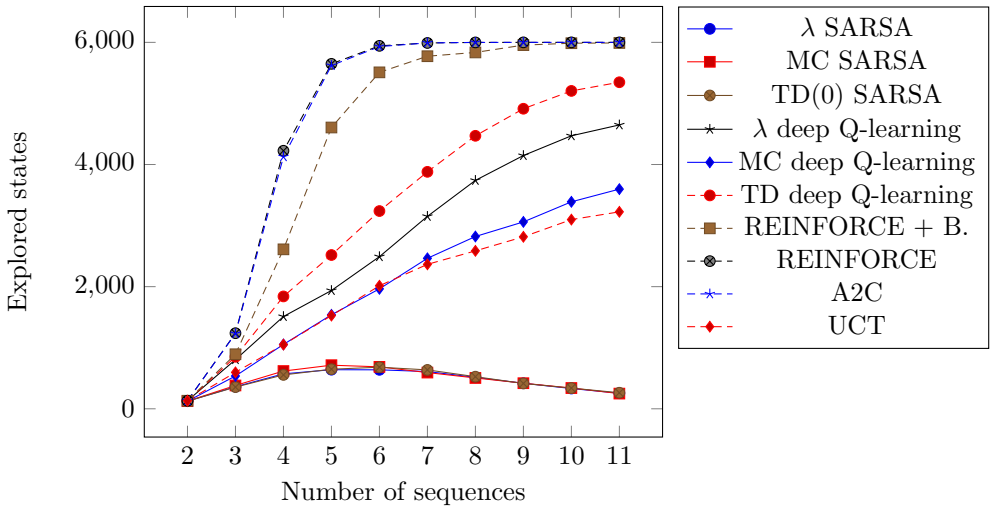


Figure 5.7: Exemplary comparison of the exploration of the different agents on the OxBench 429 dataset. The lower line consists of all three SARSA agents.

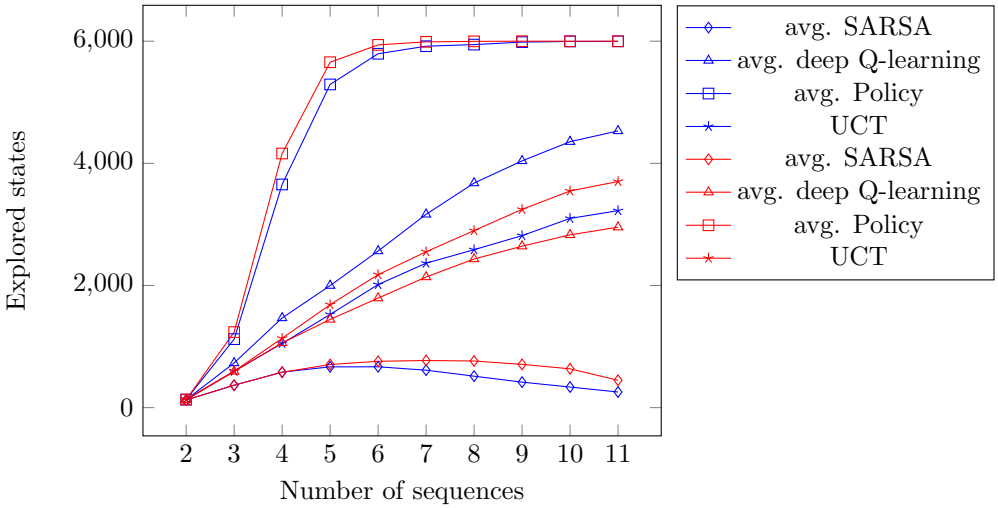


Figure 5.8: Exemplary comparison of the exploration of the different agents on the OxBench 429 dataset. The red lines show the runtime optimizing the C-score and the blue lines show the runtime when optimizing the SP-score. The values are averages over the three algorithms per category, except UCT.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work, we applied multiple algorithms from the field of reinforcement learning to the biological problem of multiple sequence alignments. We performed an extensive comparison with the three state-of-the-art tools, namely CLUSTAL, MUSCLE, and MAFFT, and with other reinforcement learning algorithms that were applied to multiple sequence alignments.

We used model from Mircea et al. [12] for simulating the progressive alignment process that is also the base of the three compared classical tools. Therefore, we implemented ten different algorithms from the field of multiple sequence alignment. These agents were applied to several problem instances and compared the results to other tools that also use reinforcement learning. We could easily reproduce the results from the papers this work is based on. Other than these previous research papers that applied reinforcement learning to multiple sequence alignments, we conducted a more comprehensive comparison on DNA data as well as on protein sequence files from the BAliBASE reference.

In chapter 5, we saw that reinforcement learning performs well on sequences that have almost the same length. Unfortunately, for some of those problem instances we used, no biologically "correct" or hand-crafted solutions are available. So, we cannot compute Q-scores and TC-scores to see how well reinforcement learning performs in terms of biological correctness. Also on the additional data from the OxBench dataset, our agents outperform the classical tools in terms of the SP-score and C-score. But when it comes to sequences form the BAliBASE whose lengths are more different, the alignment quality sinks as there are multiple small gaps interrupt the sequences in the final alignments. On most of the alignments, all state-of-the-art tools clearly outperformed our agents.

But the major drawback of reinforcement learning in multiple sequence alignments is the time that is needed to compute the alignments. Classical approaches solve the BAliBASE alignments within seconds or milliseconds; whereas the computation of our agents last for hours on the same sequence data. The reason for this is primarily based on the number of alignments a reinforcement agent computes compared to the number of alignments a classical alignment algorithm does. For example, CLUSTAL performs basically one progressive alignment that is then improved by a few iterative refinement steps, no matter how many sequences are to align. On the other hand, a reinforcement agent performs $500 \cdot$ "the number of sequences to align" alignments which are not necessarily different but in total that are often tens of thousands of pairwise alignments to compute, either of two sequences of a sequence and a profile.

So, the results of reinforcement learning in the field of multiple sequence alignment are very

promising, especially for a few sequences, DNA sequences, and sequences of similar lengths. These requirements for successful reinforcement learning agents are fulfilled by the dataset from [12], but not by the sequences from the BAliBASE. In order to improve the results on those sequences, further research is needed.

6.2 Future Work

In this following section, we discuss some interesting work to improve and expand the fields of application for reinforcement learning based alignments of multiple sequences.

6.2.1 Biological

In order to reach better Q-scores and TC-scores, one can use other variants of the SP-score for optimization. Actually, we use an SP-score with linear gap costs that lead to many single, small gaps in the alignment and multiple spread nucleotide islands. To prevent this, affine gap costs are very important.

Further improvements to the SP-score are the scores of the classical approaches that are also based on the SP-score but take many biological features of the sequences into account that are used to weight gap positions more sensitive or to weight sequences in the SP-Scoring. Some of those are described in chapter 2 and taken from the CLUSTAL scoring function. So, it would be very interesting to see if reinforcement learning is able to outperform the classical tools using their own scoring.

Another biologically interesting aspect of future work for reinforcement learning based multiple sequence alignments would be to explore, which other types of alignments could be addressed. So, for example, this technique could be applied to structural alignments and compete with MUSTANG. This approach is very easy to apply to different tasks, because one basically has to exchange the scoring function and the input to the learner.

6.2.2 Computer Science

The most obvious field of improvement, is to optimize hyperparameters. It is very time expensive to find the optimal values. Improving hyperparameters would especially include the values α, γ, and λ as those are the most influencing ones. Other hyperparameters influence the ϵ-greediness of the algorithm, so how long and to which value the ϵ is decreased.

Apart from this, the network structure used in the algorithms plays an important role. We actually use relatively tiny networks with only a few hidden layers. It would be better to find structures that match the number of input nodes better and more flexible. This means a mid-size network is too large for small instances of only a few sequences. This means, training a smaller network would be faster because it has fewer nodes. On the other hand, a mid-size network might be too small for larger problems with many more sequences. So, a network with too few nodes might be not able to learn relations between the input and the output.

Another idea regarding the networks is so-called long short-term memory- (LSTM-) cells that are used by Jafari et al. [13]. Other than the usual network cells, those LSTMs offer additional learning features and might increase the learning and convergence speed and quality of the alignments.

Some simple and more comprehending additional features are online learning for faster convergence of the learning process. Even though it is more variable for larger state spaces, it might be more efficient for larger state spaces as the learning process starts earlier and is performed more often.

The same holds for on-policy learning and off-policy learning. This is not investigated for all the agents we use. The only agents performing off-policy learning Q-learning agents, all others learn on the same policy they also use for episode generation. So, it would be interesting to see if this has an effect on the efficiency of the learning process.

Another very interesting feature is also the use of eligibility traces that are an additional alternative to the lambda return in Q-learning. This is also a method to speed up the learning process as it makes the λ-return usable in online learning.

As we briefly discuss in chapter 5, the performance of the adjusted UCT algorithm is heavily based on the balancedness of the exploratory and exploitatory part of the UCT-decision-rule. The better the estimated bounds for the alignment score match the interval within the score while learning falls in, the better the balancedness of the two parts and the better the performance of the UCT agent. So, finding better approximation for the bounds of the SP-score might lead to better results from the UCT agent.

Beside those improvements of the algorithms we use, one could change the complete setting we learn in. We briefly discuss other Markov decision processes than the one we use in this work. Especially the one presented by Edelkamp et al. could be interesting for other reinforcement learning algorithms than Nested MCTS. AlphaZero for example performed very well in tree-based models like the one presented by Edelkamp et al. [14, 9].

As we mentioned multiple times during this discussion, it would be interesting to align sequences in subgroups like a tree. Therefore, the network output has to be converted into a tree representation. Doing this adds the ability to solve multiple sequence alignments in subgroups to the reinforcement learning agents. This is very useful to solve alignments from the reference set 2 of the BAliBASE efficiently, as those consist of multiple distinct subgroups. One drawback of this is that only complete trees can be aligned and scored. This makes it nearly impossible to apply online learning to this because there are no intermediate steps to score.

Based on this idea, iterative alignments can be performed with a second network structure that selects edges from an input tree to be removed and recreated, just like in MAFFT. This network needs comprehensive information about the tree that has to be refined.

Most of the previous ideas try to improve the convergence and therefore the learning speed of the agents. Another idea to get better results is to add more information to the networks that decide which sequence to align next. This information can be given as a similarity matrix of the sequences to a convolutional layer that is later merged into the net. This information about similarities would help the agent to find similar sequences that should probably be aligned first.

Another type of information to input into the network is the sequence lengths. So, it can be learned by the networks to align longer sequences at first or at the very end.

Bibliography

[1] Da-Fei Feng and Russell F Doolittle. Progressive sequence alignment as a prerequisitetto correct phylogenetic trees. *Journal of molecular evolution*, 25(4):351–360, 1987.

[2] Qingliang Li, Tiejun Cheng, Yanli Wang, and Stephen H Bryant. Pubchem as a public resource for drug discovery. *Drug discovery today*, 15(23-24):1052–1057, 2010.

[3] Arun S Konagurthu, James C Whisstock, Peter J Stuckey, and Arthur M Lesk. Mustang: a multiple structural alignment algorithm. *Proteins: Structure, Function, and Bioinformatics*, 64(3):559–574, 2006.

[4] 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.

[5] Kazutaka Katoh, Kazuharu Misawa, Kei-ichi Kuma, and Takashi Miyata. Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic acids research*, 30(14):3059–3066, 2002.

[6] Robert C Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–1797, 2004.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[9] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[10] John Moult, Jan T Pedersen, Richard Judson, and Krzysztof Fidelis. A large-scale experiment to assess protein structure prediction methods. *Proteins: Structure, Function, and Bioinformatics*, 23(3):ii–iv, 1995.

[11] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[12] Ioan-Gabriel Mircea, Iuliana Bocicor, and Gabriela Czibula. A reinforcement learning based approach to multiple sequence alignment. In *International Workshop Soft Computing Applications*, pages 54–70. Springer, 2016.

[13] Reza Jafari, Mohammad Masoud Javidi, and Marjan Kuchaki Rafsanjani. Using deep reinforcement learning approach for solving the multiple sequence alignment problem. *SN Applied Sciences*, 1(6):592, 2019.

[14] Stefan Edelkamp and Zhihao Tang. Monte-carlo tree search for the multiple sequence alignment problem. In *Eighth Annual Symposium on Combinatorial Search*. Citeseer, 2015.

[15] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[16] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE Conference on Decision and Control*, volume 1, pages 560–564. IEEE, 1995.

[17] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[18] Julie D Thompson, Benjamin Linard, Odile Lecompte, and Olivier Poch. A comprehensive benchmark study of multiple sequence alignment methods: current challenges and future perspectives. *PloS one*, 6(3):e18093, 2011.

[19] Liisa Holm and Chris Sander. Mapping the protein universe. *Science*, 273(5275):595–602, 1996.

[20] Richard H Lathrop and Temple F Smith. Global optimum protein threading with gapped alignment and empirical pair score functions. *Journal of molecular biology*, 255(4):641–665, 1996.

[21] Julie D Thompson, Patrice Koehl, Raymond Ripp, and Olivier Poch. Balibase 3.0: latest developments of the multiple sequence alignment benchmark. *Proteins: Structure, Function, and Bioinformatics*, 61(1):127–136, 2005.

[22] GPS Raghava, Stephen MJ Searle, Patrick C Audley, Jonathan D Barber, and Geoffrey J Barton. Oxbench: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC bioinformatics*, 4(1):47, 2003.

[23] Chris P Ponting, Jörg Schultz, Frank Milpetz, and Peer Bork. Smart: identification and annotation of domains from signalling and extracellular protein sequences. *Nucleic acids research*, 27(1):229–232, 1999.

[24] Hyrum Carroll, Wesley Beckstead, Timothy O'Connor, Mark Ebbert, Mark Clement, Quinn Snell, and David McClellan. Dna reference alignment benchmarks based on tertiary structure of encoded proteins. *Bioinformatics*, 23(19):2648–2649, 2007.

[25] Ciro Leonardo Pierri, Giovanni Parisi, and Vito Porcelli. Computational approaches for protein function prediction: a combined strategy from multiple sequence alignment to molecular docking-based virtual screening. *Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics*, 1804(9):1695–1712, 2010.

[26] RC Edgar. Qscore: A quality scoring program that compares two multiple sequence alignments, 2015.

[27] 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, 1970.

[28] Michael S Waterman, Temple F Smith, and William A Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20(3):367–387, 1976.

[29] Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[30] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.

[31] Humberto Carrillo and David Lipman. The multiple sequence alignment problem in biology. *SIAM journal on applied mathematics*, 48(5):1073–1082, 1988.

[32] Jens Stoye. Multiple sequence alignment with the divide-and-conquer method. *Gene*, 211(2):GC45–GC56, 1998.

[33] Dan Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of mathematical biology*, 55(1):141–154, 1993.

[34] Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[35] Cédric Notredame and Desmond G Higgins. Saga: sequence alignment by genetic algorithm. *Nucleic acids research*, 24(8):1515–1524, 1996.

[36] Jin Kim, Sakti Pramanik, and Moon Jung Chung. Multiple sequence alignment using simulated annealing. *Bioinformatics*, 10(4):419–426, 1994.

[37] Motoo Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of molecular evolution*, 16(2):111–120, 1980.

[38] M Bocicor, I Mircea, and G Czibula. A novel reinforcement learning based approach to multiple sequence alignment. *Information Sciences*, 2014.

[39] IOAN-GABRIEL MIRCEA, MARIA-IULIANA BOCICOR, and ANDRA DÎINCU. On reinforcement learning based multiple sequence alignment. *Studia Universitatis Babes-Bolyai, Informatica*, 59(2), 2014.

[40] Ramchalam Kinattinkara Ramakrishnan, Jaspal Singh, and Mathieu Blanchette. Rlalign: A reinforcement learning approach for multiple sequence alignment. In *2018 IEEE 18th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 61–66. IEEE, 2018.

[41] Ramchalam Kinattinkara Ramakrishnan. *Exploring Reinforcement Learning Techniques in Muliple Sequence Alignment*. PhD thesis, McGill University Libraries, 2018.

[42] Alexander Kawrykow, Gary Roumanis, Alfred Kam, Daniel Kwak, Clarence Leung, Chu Wu, Eleyine Zarour, Luis Sarmenta, Mathieu Blanchette, Jérôme Waldispühl, et al. Phylo: a citizen science approach for improving multiple sequence alignment. *PloS one*, 7(3):e31362, 2012.

[43] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[44] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in neural information processing systems*, pages 703–710, 1994.

[45] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[46] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[47] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[48] Peter Marbach and John N Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

[49] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[50] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[51] Ingo Althöfer. On board-filling games with random-turn order and monte carlo perfectness. In *Advances in Computer Games*, pages 258–269. Springer, 2011.

[52] Bernd Brügmann. Monte carlo go. Technical report, Citeseer, 1993.

[53] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[54] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[55] Maarten PD Schadd, Mark HM Winands, H Jaap Van Den Herik, Guillaume MJ-B Chaslot, and Jos WHM Uiterwijk. Single-player monte-carlo tree search. In *International Conference on Computers and Games*, pages 1–12. Springer, 2008.

[56] Balibase4. http://www.lbgi.fr/balibase/. last accessed: 2020-09-30.

[57] Julie D. Thompson, Frédéric Plewniak, and Olivier Poch. Balibase: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics (Oxford, England)*, 15(1):87–88, 1999.

Appendix

	Name	1	2	3	4	5	6	7	8	9	10
1	Z75850	1.0	0.97	0.96	0.97	0.97	0.96	0.97	0.94	0.92	0.95
2	Z75841	0.97	1.0	0.99	1.0	1.0	0.99	1.0	0.97	0.95	0.97
3	Z75852	0.96	0.99	1.0	0.99	0.99	1.0	0.99	0.96	0.94	0.96
4	Z75854	0.97	1.0	0.99	1.0	1.0	0.99	1.0	0.97	0.95	0.97
5	Z75858	0.97	1.0	0.99	1.0	1.0	0.99	1.0	0.97	0.95	0.97
6	Z75851	0.96	0.99	1.0	0.99	0.99	1.0	0.99	0.96	0.94	0.96
7	Z75853	0.97	1.0	0.99	1.0	1.0	0.99	1.0	0.97	0.95	0.97
8	Z75859	0.94	0.97	0.96	0.97	0.97	0.96	0.97	1.0	0.94	0.94
9	Z75860	0.92	0.95	0.94	0.95	0.95	0.94	0.95	0.94	1.0	0.94
10	Z75861	0.95	0.97	0.96	0.97	0.97	0.96	0.97	0.94	0.94	1.0

Table A.1: Similarity matrix for the hepatits C virus type 1b sequence data from [12].

	Name	1	2	3	4	5
1	U35624	1.0	0.9	0.91	0.84	0.83
2	U35625	0.9	1.0	0.9	0.84	0.83
3	U35626	0.91	0.9	1.0	0.84	0.83
4	U35627	0.84	0.84	0.84	1.0	0.89
5	U35628	0.83	0.83	0.83	0.89	1.0

Table A.2: Similarity matrix for the papio anubis sequence data from[12].

Name	Seq. Identity	Seq. Count	Sequence Lengths			
			Median	Average	Range	Std. Dev.
Hep-C-Virus	0.97	10	212	211.9	1	0.30
Papio-Anubis	0.86	5	1092	1093	10	4.29
OxBench469	0.54	3	336	332	18	7.87
OxBench429	0.73	12	174	171	30	7.14
LGM	0.707	3	93	93	2	0.82
RLO	0.647	3	129	129.33	1	0.47
Dataset 1	0.753	3	39	39.33	1	0.47
OxBench 414	0.567	6	157.5	154.5	27	8.96
OxBench 415	0.598	4	126	126	0.0	0.0
OxBench 433	0.39	3	60	62	6	2.83
OxBench 641t2	0.55	3	201	198.67	11	4.78
OxBench 34	0.34	6	88	92.83	35	12.21
Average	0.64	5.08	225.63	226.13	11.83	4.18

Table A.3: Properties of the first dataset with DNA sequences from [12] and some protein sequence data from the OxBench dataset to test the performance on protein sequence data.

Benchmark	Data			References					SARSA		
	BF	# Seqs	ØLen	RL [12]	DRL [13]	CLUSTAL	MAFFT	MUSCLE	λ	MC	TD
Hep-C-Virus		10	212	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93
Papio Anubis	0.83	5	1092	0.82	0.82	0.82	0.82	0.83	0.83	0.83	0.83
Oxbench 469	0.46	3	332	0.45	0.37	0.37	0.35	0.36	0.46	0.46	0.46
Oxbench 429		12	171	0.2	0.2	0.18	0.19	0.18	0.22	0.21	0.2
LGM-Dataset	0.66	3	39	0.66	0.66	0.62	0.62	0.64	0.66	0.66	0.66
RLO-Dataset	0.65	3	93	0.65	0.65	0.63	0.62	0.64	0.65	0.65	0.65
Dataset 1	0.74	3	129	0.73		0.66	0.62	0.62	0.74	0.74	0.74

Benchmark	Data	Q-learning			Policy Approximation			Tree based
	BF	λ	MC	TD	REINFORCE + Bsl.	REINFORCE	A2C	UCT
Hep-C-Virus		0.93	0.93	0.93	0.93	0.93	0.93	0.93
Papio Anubis	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
Oxbench 469	0.46	0.46	0.46	0.46	0.46	0.46	0.46	0.46
Oxbench 429		0.22	0.21	0.23	0.21	0.20	0.21	0.20
LGM-Dataset	0.66	0.66	0.66	0.66	0.66	0.66	0.66	0.66
RLO-Dataset	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
Dataset 1	0.74	0.74	0.74	0.74	0.74	0.74	0.74	0.74

Table A.4: C-scores on datasets from [12]. The scores are computed as progressive alignments by different agents. In green the interesting bests are marked and in red are the comparatively bad alignments.

Benchmark	Data			References					SARSA		
	BF	# Seqs	ØLen	RL [12]	DRL [13]	CLUSTAL	MAFFT	MUSCLE	λ	MC	TD
Hep-C-Virus		10	212	18627	18627	18627	18627	18627	18627	18627	18627
Papio Anubis	18881	5	1092	18719	18860	18827	18860	18878	18875	18881	18881
Oxbench 469	672	3	332	565	565	464	549	555	672	672	672
Oxbench 429		12	171	8668	10218	9575	10218	10965	12066	12078	12069
LGM-Dataset	348	3	39	345	345	345	345	348	348	348	348
RLO-Dataset	488	3	93	486	486	480	471	479	486	486	486
Dataset 1	167	3	129	167		149	134	137	167	167	167

Benchmark	Data	Q-learning			Policy Approximation			Tree based
	BF	λ	MC	TD	REINFORCE + Bsl.	REINFORCE	A2C	UCT
Hep-C-Virus		18627	18627	18627	18627	18627	18627	18627
Papio Anubis	18881	18881	18881	18875	18875	18875	18833	18848
Oxbench 469	672	672	672	672	672	672	672	639
Oxbench 429		12102	12084	12120	12120	12111	12120	11853
LGM-Dataset	348	348	348	348	348	348	348	348
RLO-Dataset	488	488	488	488	488	488	488	488
Dataset 1	167	167	167	167	167	167	167	167

Table A.5: SP-scores on datasets from [12]. The scores are computed as progressive alignments by different agents. In green the interesting bests are marked and in red are the comparatively bad alignments.

Benchmark	Data			References					SARSA		
	BF	# Seqs	ØLen	RL [12]	DRL [13]	CLUSTAL	MAFFT	MUSCLE	λ	MC	TD
Hep-C-Virus		10	212	18627	18627	18627	18627	18627	18627	18627	18627
Papio Anubis	18881	5	1092	18719	18860	18827	18860	18878	18875	18881	18869
Oxbench 469	167	3	332	565	565	464	549	555	654	663	663
Oxbench 429		12	171	8668	10218	9575	10218	10965	11598	12015	11664
LGM-Dataset	348	3	39	345	345	345	345	348	342	348	348
RLO-Dataset	488	3	93	486	486	480	471	479	486	486	486
Dataset 1	167	3	129	167		149	134	137	167	167	167

Benchmark	Data	Q-learning			Policy Approximation			Tree based
	BF	λ	MC	TD	REINFORCE + Bsl.	REINFORCE	A2C	UCT
Hep-C-Virus		18627	18627	18627	18627	18627	18627	18627
Papio Anubis	18881	18881	18860	18881	18881	18881	18881	18627
Oxbench 469	672	672	663	672	672	672	672	639
Oxbench 429		11169	11883	11061	11646	11694	12030	11922
LGM-Dataset	348	348	348	348	348	348	348	348
RLO-Dataset	488	488	488	488	488	488	488	488
Dataset 1	167	167	167	167	167	167	167	167

Table A.6: SP-scores on datasets from [12]. The scores are computed as progressive alignments by different agents that optimized the C-score and those SP-scores are a side product of this optimization. In green the interesting bests are marked and in red are the comparatively bad alignments.

Benchmark	Data				References			SARSA		
	BF	Type	# Seqs	ØLen	CLUSTAL	MAFFT	MUSCLE	λ	MC	TD
OxBench 414	0.38	DNA	6	154	0.31	0.32	0.33	0.38	0.38	0.38
OxBench 415	0.40	DNA	4	126	0.31	0.31	0.33	0.40	0.40	0.40
OxBench 433	0.32	PEP	3	186	0.29	0.29	0.28	0.32	0.32	0.32
OxBench 641t2	0.45	PEP	3	271	0.42	0.42	0.38	0.45	0.45	0.45
OxBench 34	0.06	PEP	6	215	0.03	0.03	0.03	0.06	0.06	0.05

Benchmark	Data	Q-learning			Policy			Tree
	BF	λ	MC	TD	REINFORCE + B	REINFORCE	A2C	UCT
OxBench 414	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.38
OxBench 415	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40
OxBench 433	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32
OxBench 641t2	0.45	0.45	0.45	0.45	0.45	0.45	0.45	0.45
OxBench 34	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.03

Table A.7: C-scores for additional data for first analysis using different agents in progressive alignment setting. In red are the comparatively bad alignments.

Benchmark	Data				References			SARSA		
	BF	Type	# Seqs	ØLen	CLUSTAL	MAFFT	MUSCLE	λ	MC	TD
OxBench 414	1974	DNA	6	154	1800	1806	1855	1980	1980	1980
OxBench 415	603	DNA	4	126	513	513	534	603	603	603
OxBench 433	353	PEP	3	186	268	290	24	353	353	353
OxBench 641t2	1053	PEP	3	271	659	723	442	1053	1053	1053
OxBench 34	1490	PEP	6	215	355	377	-206	1210	1215	1210

Benchmark	Data	Q-learning			Policy			Tree
	BF	λ	MC	TD	REINFORCE + B	REINFORCE	A2C	UCT
OxBench 414	1974	1980	1980	1974	1980	1980	1980	1743
OxBench 415	603	603	603	603	603	603	603	603
OxBench 433	353	353	353	353	353	353	353	353
OxBench 641t2	1053	1053	1053	1053	1053	1053	1053	870
OxBench 34	1490	1215	1218	1223	1223	1221	1223	1208

Table A.8: SP-scores for additional data for first analysis using different agents in progressive alignment setting. In red are the comparatively bad alignments.

Name	Seq. Identity	Seq. Count	Sequence Lengths			
			Median	Average	Range	Std. Dev.
BB11001.tfa	0.24	4	85.5	86.25	8	2.95
BB11002.tfa	0.20	8	83	98	141	43.04
BB11003.tfa	0.24	4	475	470	102	41.79
BB11004.tfa	0.24	4	419	421	66	26.66
BB11006.tfa	0.23	8	251.5	244.75	97	34.67
BB11007.tfa	0.24	9	411	418.33	72	24.91
BB11008.tfa	0.17	4	288	305	436	195.35
BB11009.tfa	0.19	4	215	216	240	113.22
BB11010.tfa	0.24	4	492	491.5	2	0.87
BB11011.tfa	0.23	5	216	201	82	31.62
BB11012.tfa	0.24	4	369.5	364	77	28.04
BB11013.tfa	0.21	5	74	71.6	50	18.14
BB11015.tfa	0.25	4	305	308.5	30	12.22
BB11016.tfa	0.23	8	452.5	470.5	413	114.07
BB11017.tfa	0.24	4	250	252.75	17	6.94
BB11019.tfa	0.25	10	349.5	348.4	97	27.02
BB11020.tfa	0.24	9	219	219.22	36	13.21
BB11021.tfa	0.24	4	114	117.25	37	14.91
BB11022.tfa	0.21	4	102	118	142	56.29
BB11023.tfa	0.24	7	309	309.86	176	59.21
BB11024.tfa	0.23	4	395.5	407	93	35.05
BB11025.tfa	0.20	4	82	82.75	39	17.85
BB11026.tfa	0.18	7	98	240	830	283.49
BB11027.tfa	0.23	7	209	232.57	257	82.65
BB11028.tfa	0.22	10	177.5	158.8	118	48.46
BB11029.tfa	0.22	4	94.5	102	57	22.33
BB11032.tfa	0.23	8	283	294.75	177	53.91
BB11034.tfa	0.24	8	471.5	491.62	328	97.93
BB11035.tfa	0.24	5	85	97.4	67	24.95
BB11036.tfa	0.24	8	370.5	370	138	44.07
BB11038.tfa	0.23	8	335	364.25	353	106.99
Average	0.235	6	260.71	270.1	154.13	54.28

Table A.9: Properties of the sequence data we use from the reference set 1.1 from BAliBASE v4.0.

Name	Seq. Identity	Seq. Count	Sequence Lengths			
			Median	Average	Range	Std. Dev.
BB12002.tfa	0.26	6	213	208	66	20.46
BB12003.tfa	0.28	8	66.5	71.25	27	11.00
BB12005.tfa	0.28	9	202	209	37	12.06
BB12006.tfa	0.28	4	233	232	22	7.97
BB12009.tfa	0.21	5	72	112.6	134	54.75
BB12010.tfa	0.27	7	357	348.29	120	37.11
BB12012.tfa	0.25	4	324	373.5	250	102.60
BB12014.tfa	0.23	9	69	97.22	205	61.36
BB12016.tfa	0.17	5	124	494.2	987	471.93
BB12017.tfa	0.28	7	477	472.86	43	15.34
BB12020.tfa	0.26	4	127.5	125.5	11	4.50
BB12021.tfa	0.30	6	73	75.67	14	5.50
BB12022.tfa	0.18	5	157	250.4	396	180.39
BB12024.tfa	0.30	4	241	243.25	39	14.04
BB12025.tfa	0.24	4	210	202	42	17.10
BB12028.tfa	0.24	8	276.5	294.5	272	83.67
BB12032.tfa	0.28	9	65	66.11	24	7.19
BB12033.tfa	0.25	7	470	454.71	370	120.43
BB12034.tfa	0.28	4	279	276.25	19	7.26
BB12036.tfa	0.30	7	217	209.57	32	13.16
BB12038.tfa	0.23	10	259.5	492.4	2091	610.57
BB12040.tfa	0.26	5	129	132.2	40	15.52
BB12041.tfa	0.27	7	110	113.57	37	11.65
Average	0.268	6.26	206.61	241.52	229.48	81.98

Table A.10: Properties of the sequence data we use from the reference set 1.2 from BAliBASE v4.0.

Name	Seq. Identity	Seq. Count	Sequence Lengths			
			Median	Average	Range	Std. Dev.
BB40010.tfa	0.27	9	70	109	147	58.80
BB40014.tfa	0.27	9	507	445.22	311	119.24
BB40018.tfa	0.19	9	124	342.33	959	336.55
Average	0.243	9	233.67	298.95	472.33	171.53
BB50004.tfa	0.33	9	415	429.56	119	41.73

Table A.11: Properties of the sequence data we use from the reference sets 4 and 5 from BAliBASE v4.0.

Group	Count	Win Ratio	Draw Ratio	CLUSTAL	MUSCLE	MAFFT	C Value	SP Value
RV11 100	26	0.0	0.0	0.46	0.4	0.4	0.08	0.06
RV11 500	5	0.0	0.2	0.38	0.37	0.37	0.11	0.16
RV12 100	19	0.0	0.05	0.79	0.77	0.77	0.32	0.3
RV12 500	4	0.0	0.0	0.84	0.82	0.82	0.3	0.25
RV4	3	0.0	0.0	0.75	0.67	0.67	0.04	0.06
RV5	1	0.0	0.0	0.94	0.94	0.94	0.05	0.1

Table A.12: Comparison of the Q-score of Q-learning against classical approaches.

Group	Count	Win Ratio	Draw Ratio	CLUSTAL	MUSCLE	MAFFT	C Value	SP Value
RV11 100	26	0.0	0.0	0.23	0.19	0.19	0.02	0.01
RV11 500	5	0.2	0.2	0.23	0.15	0.15	0.03	0.03
RV12 100	19	0.0	0.05	0.59	0.57	0.57	0.18	0.16
RV12 500	4	0.0	0.0	0.59	0.53	0.53	0.05	0.08
RV4	3	0.0	0.0	0.33	0.18	0.18	0.01	0.0
RV5	1	0.0	0.0	0.85	0.81	0.81	0.01	0.14

Table A.13: Comparison of the TC-score of Q-learning against classical approaches.

Group	Count	Win Ratio	Draw Ratio	CLUSTAL	MUSCLE	MAFFT	C Value	SP Value
RV11 100	26	0.85	0.88	-3568.12	-3637.88	-3637.88	-3098.85	-2647.46
RV11 500	5	1.0	1.0	-791.0	-712.2	-712.2	-434.8	-335.2
RV12 100	19	0.53	0.63	-1347.16	-1149.89	-1149.89	-2591.42	-2079.47
RV12 500	4	0.0	0.0	535.5	833.25	833.25	-808.5	-791.25
RV4	3	0.33	0.33	-3290.0	-3394.0	-3394.0	-5712.0	-5003.67
RV5	1	0.0	0.0	5737.0	5918.0	5918.0	-7742.0	-6657.0

Table A.14: Comparison of the SP-score of Q-learning against classical approaches. In green the interesting quotes are marked.

Group	Count	Win Ratio	Draw Ratio	CLUSTAL	MUSCLE	MAFFT	C Value	SP Value
RV11 100	26	0.12	0.31	0.02	0.02	0.02	0.02	0.01
RV11 500	5	0.6	0.8	0.01	0.02	0.02	0.02	0.01
RV12 100	19	0.16	0.16	0.06	0.07	0.07	0.04	0.03
RV12 500	4	0.0	0.0	0.06	0.07	0.07	0.03	0.0
RV4	3	0.0	0.0	0.02	0.03	0.03	0.0	0.0
RV5	1	0.0	0.0	0.08	0.08	0.08	0.0	0.0

Table A.15: Comparison of the C-score of Q-learning against classical approaches. In green the interesting quotes are marked.

Data	Name		BB11001	BB11002	BB11003	BB11013	BB11025	BB12020	BB12024	BB12036	BB12040	BB40010	BB40014	BB40018	BB50004
	Ref.	C	0.01	0.01	0.02	0.0	0.01	0.12	0.15	0.1	0.08	0.03	0.04	0.0	0.08
		SP	-218	-1856	-1403	-726	-398	341	544	822	-28	5103	-2722	-14295	5600
Classical Algorithms	CLUSTAL	Q	0.9	0.46	0.59	0.05	0.09	0.31	0.93	0.96	0.93	0.81	0.64	0.79	0.94
		TC	0.8	0.11	0.39	0.0	0.0	0.05	0.87	0.9	0.83	0.31	0.28	0.41	0.85
		C	0.0	0.01	0.02	0.01	0.01	0.06	0.16	0.1	0.08	0.03	0.04	0.0	0.08
		SP	-245	-2372	-1113	-814	-408	-276	489	791	-63	5350	-1005	-14215	5737
	MAFFT	Q	0.72	0.37	0.49	0.26	0.17	0.81	0.88	0.89	0.77	0.76	0.62	0.64	0.94
		TC	0.58	0.02	0.29	0.03	0.08	0.74	0.78	0.74	0.56	0.17	0.22	0.15	0.81
		C	0.04	0.01	0.04	0.0	0.03	0.11	0.16	0.1	0.09	0.04	0.04	0.0	0.08
		SP	-261	-2125	-1164	-622	-369	353	585	733	-59	5613	-1554	-14241	5918
	MUSCLE	Q	0.88	0.52	0.54	0.07	0.1	0.81	0.9	0.94	0.93	0.78	0.61	0.74	0.94
		TC	0.8	0.19	0.36	0.0	0.03	0.71	0.82	0.9	0.85	0.27	0.21	0.2	0.83
		C	0.01	0.01	0.03	0.0	0.01	0.11	0.16	0.1	0.08	0.04	0.04	0.0	0.08
		SP	-189	-1820	-1245	-628	-406	295	520	730	-47	5364	-1734	-13999	5552
Q-learning	C-score	Q	0.24	0.02	0.02	0.11	0.03	0.6	0.73	0.46	0.53	0.09	0.01	0.02	0.05
		TC	0.0	0.0	0.0	0.08	0.0	0.41	0.54	0.25	0.29	0.02	0.0	0.0	0.01
		C	0.04	0.0	0.02	0.01	0.03	0.13	0.16	0.04	0.05	0.0	0.0	0.0	0.0
		SP	-116	-1562	-389	-391	-203	543	817	-340	79	1799	-6187	-12748	-7742
	SP-score	Q	0.46	0.01	0.03	0.11	0.05	0.62	0.73	0.36	0.49	0.17	0.0	0.03	0.1
		TC	0.11	0.0	0.0	0.01	0.0	0.43	0.54	0.07	0.23	0.0	0.0	0.0	0.14
		C	0.03	0.0	0.02	0.0	0.0	0.12	0.16	0.02	0.05	0.0	0.0	0.0	0.0
		SP	-45	-1543	-203	-277	-53	551	817	-503	156	3996	-6372	-12635	-6657

Table A.16: Comparison of the four scores we use applied to alignments from state-of-the-art-tools and Q-learning algorithms using the λ-return. The scores are computed on sequence data from from 4 different reference sets of the BAliBASE [21]. In green the interesting bests are marked and in red are the surprisingly bad alignments.

	Name	1	2	3	4
1	1aab_	1	0.25	0.24	0.21
2	1j46_A	0.25	1	0.24	0.28
3	1k99_A	0.24	0.24	1	0.24
4	2lef_A	0.21	0.28	0.24	1

Table A.17: Identity matrix for the BB11001 dataset from the RV11 reference set of BAliBASE.

	Name	1	2	3	4	5	6	7	8
1	1abo_A	1	0.17	0.22	0.22	0.23	0.19	0.18	0.23
2	1ycs_B	0.17	1	0.2	0.13	0.18	0.19	0.22	0.16
3	1pht_	0.22	0.2	1	0.21	0.25	0.19	0.22	0.2
4	1ihv_A	0.22	0.13	0.21	1	0.18	0.19	0.18	0.17
5	1bb9_	0.23	0.18	0.25	0.18	1	0.23	0.21	0.2
6	1hjd_A	0.19	0.19	0.19	0.19	0.23	1	0.22	0.19
7	1ov3_A	0.18	0.22	0.22	0.18	0.21	0.22	1	0.23
8	1ue9_A	0.23	0.16	0.2	0.17	0.2	0.19	0.23	1

Table A.18: Identity matrix for the BB11002 dataset from the RV11 reference set of BAliBASE.

	Name	1	2	3	4
1	1ad3_A	1	0.25	0.24	0.23
2	1uzb_A	0.25	1	0.24	0.25
3	1eyy_A	0.24	0.24	1	0.23
4	1o20_A	0.23	0.25	0.23	1

Table A.19: Identity matrix for the BB11003 dataset from the RV11 reference set of BAliBASE.

	Name	1	2	3	4	5
1	1idy_	1	0.21	0.18	0.22	0.18
2	1hst_A	0.21	1	0.21	0.22	0.2
3	1tc3_C	0.18	0.21	1	0.24	0.21
4	1aoy_	0.22	0.22	0.24	1	0.2
5	1jhg_A	0.18	0.2	0.21	0.2	1

Table A.20: Identity matrix for the BB11013 dataset from the RV11 reference set of BAliBASE.

	Name	1	2	3	4
1	1tvx_A	1	0.2	0.19	0.22
2	1prt_F	0.2	1	0.19	0.21
3	1sap_	0.19	0.19	1	0.18
4	1lt5_D	0.22	0.21	0.18	1

Table A.21: Identity matrix for the BB11025 dataset from the RV11 reference set of BAliBASE.

	Name	1	2	3	4
1	1vvc_	1	0.3	0.22	0.27
2	1ghq_B	0.3	1	0.23	0.28
3	1ckl_A	0.22	0.23	1	0.28
4	1nwv_A	0.27	0.28	0.28	1

Table A.22: Identity matrix for the BB12020 dataset from the RV12 reference set of BAliBASE.

	Name	1	2	3	4
1	1led_	1	0.31	0.29	0.31
2	1lte_	0.31	1	0.32	0.31
3	1avb_A	0.29	0.32	1	0.27
4	LECN_PEA	0.31	0.31	0.27	1

Table A.23: Identity matrix for the BB12024 dataset from the RV12 reference set of BAliBASE.

	Name	1	2	3	4	5	6	7
1	1zip_	1	0.32	0.29	0.3	0.26	0.27	0.26
2	2ak2_	0.32	1	0.33	0.3	0.27	0.28	0.29
3	2ak3_A	0.29	0.33	1	0.28	0.26	0.25	0.26
4	1zak_A	0.3	0.3	0.28	1	0.26	0.31	0.29
5	1qf9_A	0.26	0.27	0.26	0.26	1	0.35	0.41
6	3adk_	0.27	0.28	0.25	0.31	0.35	1	0.37
7	1uky_	0.26	0.29	0.26	0.29	0.41	0.37	1

Table A.24: Identity matrix for the BB12036 dataset from the RV12 reference set of BAliBASE.

	Name	1	2	3	4	5
1	1hrb_	1	0.3	0.27	0.26	0.23
2	HEML_NAEFO	0.3	1	0.25	0.27	0.24
3	HEMT_METJA	0.27	0.25	1	0.29	0.23
4	HEMT_CLOAB	0.26	0.27	0.29	1	0.24
5	HEMT_PSEAE	0.23	0.24	0.23	0.24	1

Table A.25: Identity matrix for the BB12040 dataset from the RV12 reference set of BAliBASE.

	Name	1	2	3	4	5	6	7	8	9
1	1csp	1	0.47	0.44	0.18	0.37	0.18	0.38	0.3	0.2
2	1mef	0.47	1	0.46	0.19	0.43	0.2	0.37	0.37	0.2
3	CSPA_BORPE	0.44	0.46	1	0.17	0.31	0.18	0.3	0.27	0.2
4	GRP2_NICSY	0.18	0.19	0.17	1	0.2	0.55	0.15	0.16	0.17
5	CSPA_RHIME	0.37	0.43	0.31	0.2	1	0.21	0.43	0.29	0.19
6	GR2B_ARATH	0.18	0.2	0.18	0.55	0.21	1	0.16	0.18	0.18
7	CSPA_RICCN	0.38	0.37	0.3	0.15	0.43	0.16	1	0.28	0.19
8	CSPH_ECOLI	0.3	0.37	0.27	0.16	0.29	0.18	0.28	1	0.19
9	PIPI_HUMAN	0.2	0.2	0.2	0.17	0.19	0.18	0.19	0.19	1

Table A.26: Identity matrix for the BB40010 dataset from the RV40 reference set of BAliBASE.

	Name	1	2	3	4	5	6	7	8	9
1	1bqb_A	1	0.44	0.38	0.28	0.31	0.27	0.22	0.21	0.23
2	SEPA_STAEP	0.44	1	0.29	0.33	0.26	0.22	0.26	0.24	0.27
3	1esp_	0.38	0.29	1	0.4	0.3	0.29	0.23	0.23	0.26
4	NPRS_BACST	0.28	0.33	0.4	1	0.26	0.23	0.26	0.25	0.28
5	PRT1_ERWCA	0.31	0.26	0.3	0.26	1	0.26	0.22	0.22	0.23
6	1ezm_	0.27	0.22	0.29	0.23	0.26	1	0.29	0.32	0.21
7	PROA_LEGLO	0.22	0.26	0.23	0.26	0.22	0.29	1	0.3	0.25
8	HAPT_VIBCH	0.21	0.24	0.23	0.25	0.22	0.32	0.3	1	0.25
9	PRZN_RENSA	0.23	0.27	0.26	0.28	0.23	0.21	0.25	0.25	1

Table A.27: Identity matrix for the BB40014 dataset from the RV40 reference set of BAliBASE.

	Name	1	2	3	4	5	6	7	8	9
1	1fmb_	1	0.28	0.09	0.31	0.3	0.17	0.26	0.1	0.18
2	7upj_B	0.28	1	0.08	0.27	0.31	0.17	0.26	0.1	0.17
3	POL_HV2D2	0.09	0.08	1	0.09	0.1	0.17	0.09	0.22	0.18
4	1b11_A	0.31	0.27	0.09	1	0.27	0.17	0.25	0.1	0.17
5	1bai_A	0.3	0.31	0.1	0.27	1	0.2	0.26	0.12	0.2
6	VPRT_JSRV	0.17	0.17	0.17	0.17	0.2	1	0.19	0.21	0.37
7	1nso_A	0.26	0.26	0.09	0.25	0.26	0.19	1	0.11	0.21
8	GAG_IPMA	0.1	0.1	0.22	0.1	0.12	0.21	0.11	1	0.21
9	VPRT_SMRVH	0.18	0.17	0.18	0.17	0.2	0.37	0.21	0.21	1

Table A.28: Identity matrix for the BB40018 dataset from the RV40 reference set of BAliBASE.

	Name	1	2	3	4	5	6	7	8	9
1	1qpg_	1	0.37	0.48	0.4	0.33	0.32	0.35	0.3	0.29
2	13pk_A	0.37	1	0.37	0.39	0.39	0.6	0.34	0.28	0.29
3	1ltk_A	0.48	0.37	1	0.38	0.32	0.32	0.31	0.3	0.27
4	1php_	0.4	0.39	0.38	1	0.33	0.35	0.37	0.31	0.29
5	P25055	0.33	0.39	0.32	0.33	1	0.46	0.29	0.26	0.26
6	P08891	0.32	0.6	0.32	0.35	0.46	1	0.29	0.26	0.28
7	Q8XD03	0.35	0.34	0.31	0.37	0.29	0.29	1	0.3	0.3
8	Q9PQL2	0.3	0.28	0.3	0.31	0.26	0.26	0.3	1	0.28
9	Q97BC6	0.29	0.29	0.27	0.29	0.26	0.28	0.3	0.28	1

Table A.29: Identity matrix for the BB50004 dataset from the RV40 reference set of BAliBASE.

Benchmark	Data		References				SARSA		
	# Seqs	ØLen	RL [12]	CLUSTAL	MAFFT	MUSCLE	λ	MC	TD
Hep-C-Virus	10	212	< 2	0.05	0.08	0.02	1800.75	1860.1	1711.39
Papio Anubis	5	1092	190	0.49	0.09	0.28	2984.31	3180.77	3329.85
Oxbench 469	3	332	< 1	0.03	0.08	0.02	11.51	11.41	11.4
Oxbench 429	12	171	186	0.06	0.08	0.03	1992.5	2056.51	2050.95
LGM-Dataset	3	39	< 1	0.01	0.07	0.002	0.85	0.86	0.86
RLO-Dataset	3	93	< 1	0.01	0.07	0.003	1.67	1.67	1.65
Dataset 1	3	129	< 1	0.01	0.07	0.002	0.17	0.16	0.2

Benchmark	Q-learning			Policy			Tree based
	λ	MC	TD	REINFORCE + Bsl.	REINFORCE	A2C	UCT
Hep-C-Virus	11797.99	9275.2	15944.95	19934.94	188735.25	18700.3	6224.15
Papio Anubis	6120.25	5141.77	5679.35	4397.12	4003.9	4001.91	3967.13
Oxbench 469	11.27	13.59	11.17	16.94	12.07	11.99	13.92
Oxbench 429	18303.52	13824.49	20607.26	20910.33	24926.32	23099.49	10484.31
LGM-Dataset	0.86	9.62	10.96	5.86	1.38	1.37	1.01
RLO-Dataset	10.76	1.64	5.2	7.65	3.28	6.69	2.05
Dataset 1	0.17	0.17	0.17	9.5	1.43	0.78	0.2

Table A.30: Comparison of the runtime of C-score optimizing agents.

Benchmark	Data		References				SARSA		
	# Seqs	ØLen	RL [12]	CLUSTAL	MAFFT	MUSCLE	λ	MC	TD
Hep-C-Virus	10	212	< 2	0.05	0.08	0.02	2073.78	2189.96	2126.08
Papio Anubis	5	1092	190	0.49	0.09	0.28	3238.14	2984.15	2982.06
Oxbench 469	3	332	< 1	0.03	0.08	0.02	11.38	11.37	11.29
Oxbench 429	12	171	186	0.06	0.08	0.03	26673.9	2823.75	2791.71
LGM-Dataset	3	39	< 1	0.01	0.07	0.002	1.01	1.06	1.25
RLO-Dataset	3	93	< 1	0.01	0.07	0.003	1.62	1.62	1.77
Dataset 1	3	129	< 1	0.01	0.07	0.002	0.17	0.28	0.22

Benchmark	Q-learning			Policy			Tree based
	λ	MC	TD	REINFORCE + Bsl.	REINFORCE	A2C	UCT
Hep-C-Virus	11664.02	11943.84	11242.68	21614.37	19483.68	18696.89	6214.93
Papio Anubis	6571.49	6571.85	6777.13	5617.07	4431.8	4301.8	3513.6
Oxbench 469	11.36	11.31	11.48	18.2	11.72	12.45	1.95
Oxbench 429	19288.6	18007.15	17302.26	29675.77	25140.1	25726.59	11490.03
LGM-Dataset	1.11	0.95	1.09	17.57	2.29	1.89	0.32
RLO-Dataset	1.63	1.65	2.5	6.5	2.9	2.22	0.29
Dataset 1	3.89	0.32	0.37	7.81	0.78	1.12	0.07

Table A.31: Comparison of the runtime of SP-score optimizing agents.

		Length	2	3	4	5	6	7	8	9
		max	90	720	5040	30240	151200	604800	1814400	3628800
C-score	SARSA	λ	86	284	465	545	534	465	374	284
		MC	88	296	489	557	565	483	391	284
		TD	89	294	459	538	496	431	349	254
	Q-lrn.	λ	90	583	1396	2014	2356	2591	2764	2864
		MC	89	344	756	1180	1532	1871	2200	2429
		TD	90	552	1541	2763	3463	3851	4069	4197
	Policy	RF. + B.	90	713	2939	4500	4863	4958	4989	4994
		RF	90	711	916	4466	4881	4967	4988	4995
		A2C	90	714	2890	4433	4877	4959	4993	4997
		UCT	90	467	851	1176	1406	1567	1781	1889
SP-score	SARSA	λ	86	259	440	523	570	592	541	358
		MC	88	275	453	607	655	614	525	371
		TD	90	258	423	564	618	607	526	378
	Q-lrn.	λ	89	396	765	1089	1365	1610	1782	1896
		MC	89	438	783	1079	1300	1515	1675	1794
		TD	90	416	762	1070	1300	1497	1676	1792
	Policy	RF. + B.	90	715	2776	4361	4831	4954	4985	4988
		RF	90	713	2905	4475	4868	4965	4987	4993
		A2C	90	710	2888	4471	4881	4962	4985	4992
		UCT	90	417	853	1190	1411	1586	1799	1905

Table A.32: Number of alignments computed on the hepatitis C virus dataset from [12]. The amounts are distinguished into the number of sequences aligned.

		C-score			SP-score		
	Length	2	3	4	2	3	4
	max	20	60	120	20	60	120
SARSA	λ	20	55	81	20	58	89
	MC	20	55	88	20	55	83
	TD	20	55	81	20	58	79
Q-lrn.	λ	20	58	104	20	59	105
	MC	20	53	79	20	59	107
	TD	20	55	99	20	58	111
Policy	RF. + B.	20	60	120	20	60	119
	RF.	20	60	120	20	60	120
	A2C	20	60	120	20	60	120
UCT		20	60	119	20	60	116

Table A.33: Number of alignments computed on the papio anubis dataset from [12]. The amounts are distinguished into the number of sequences aligned.

		Length	2	3	4	5	6	7	8	9	10	11
		max	132	1320	11880	95040	665280	3991680	19958400	79833600	239500800	479001600
C-score	SARSA	λ	125	361	572	640	638	610	515	418	332	255
		MC	125	382	618	714	687	593	505	419	340	249
		TD	125	355	555	648	681	637	526	415	339	263
	Q-lrn.	λ	132	805	1511	1937	2491	3151	3740	4147	4470	4648
		MC	125	537	1054	1539	1964	2464	2821	3060	3389	3597
		TD	132	852	1840	2518	3238	3880	4470	4913	5205	5347
	Policy	RF. + B.	132	893	2611	4606	5509	5773	5835	5955	5986	5990
		RF	132	1238	4225	5648	5943	5990	5999	6000	6000	6000
		A2C	132	1232	4127	5621	5931	5991	5999	6000	6000	6000
	UCT		132	597	1051	1526	2015	2365	2586	2816	3098	3225
SP-score	SARSA	λ	125	370	567	669	727	749	727	662	593	439
		MC	126	379	617	737	782	794	783	725	633	446
		TD	124	356	552	713	67	772	780	737	681	458
	Q-lrn.	λ	126	594	1113	1537	1875	2228	2519	2744	2949	3066
		MC	130	620	1055	1412	1776	2118	2422	2647	2806	2923
		TD	129	562	1011	1383	1728	2070	2359	2539	2732	2874
	Policy	RF. + B.	132	1208	3932	5606	5937	991	5998	6000	6000	6000
		RF	132	1245	4154	647	5935	5988	5997	5999	6000	6000
		A2C	132	1259	4397	5714	5946	5989	5997	6000	6000	6000
	UCT		132	597	1135	1686	2177	2552	2899	3246	3547	3703

Table A.34: Number of alignments computed on the OxBench 429 dataset from [12]. The amounts are distinguished into the number of sequences aligned.