Leiden University

Master Computer Science: Bioinformatics

master's thesis

AlphaGo-like Inverse RNA Folding algorithms

Author
Oscar van der Meer
(s1913638)

Supervisors
Alexander Goultiaev
Andrius Bernatavicius

15th June 2021

Abstract

In RNA inverse folding the goal is to find a nucleotide sequence given a target secondary or tertiary structure. The RNA inverse folding problem is an NP-complete problem and chaotic in nature, which makes finding a near-perfect algorithm for solving the problem challenging. The RNA inverse folding problem has recently been converted into a game called Eterna where human players are able to solve together problems that the best algorithms today cannot solve. AlphaGo-like algorithms are great in solving games where human strategies are needed. Therefore in this project, we explore the possibility of implementing an AlphaGo-like algorithm for the RNA inverse folding problem. In this process we give some promising solutions for the pre-requisites of such an algorithm. First, this includes a representation of the secondary structure of the RNA molecule in the form of a matrix. Secondly, we propose a state representation in the form of a tensor, which can readily be used by a neural network as input. Third, we based the neural network architecture at the core of the Monte Carlo Tree Search on U-Net. Finally, we combine these proposed pre-requisites together, culminating in a proof of concept.

Acknowledgements and personal note

First and foremost, I would like to thank both my supervisors, dr. Alexander Goultiaev and Andrius Bernatavicius, for their everlasting support, insights and enthusiasm throughout the project, especially during these difficult times.

This master thesis was executed and written during the Corona pandemic, in which personal contact with my supervisors/others and working on-site at the university was not possible. I strongly believe that this project (and many others) suffered from this; though luckily most things could be done from home (due to the nature of the project), I feel like more could have come out of this project. This experience has shown me the importance of personal guidance, day-to-day social contact and on-site working for a more effective, pleasant and productive workday.

Contents

1 Introduction

RNA not only is a necessary step in the translation of DNA into proteins, but it also plays a vital role in the regulatory processes of the cell. These functions for example include controlling the mRNA translation or regulating the replication in single-stranded RNA viruses. The function of an RNA molecule is determined by its secondary or tertiary structure, which it gets by folding back on itself. The process of folding is determined by the sequence of nucleotides of the RNA molecule. Determining which structure an RNA molecule with a certain sequence folds into is, therefore, an important problem to solve within the life sciences. (Zuker et al., 1999)

Arguably, an even bigger challenge is the so-called inverse RNA folding problem: what sequence of nucleotides would fold into a given secondary structure? This knowledge can be used to engineer RNA molecules with specific functions, such as regulating gene expression, sensing small molecules or self-cleaving. The engineering of these RNA molecules with specific functions has a promising outlook in the field of synthetic biology and can ultimately lead to new ways of combating diseases. The inverse RNA folding problem is found to be NP-complete and is chaotic in nature (Cazenave & Fournier, 2020). A small change in the nucleotide sequence can have a large impact on the resulting secondary structure. Brute forcing an inverse RNA folding problem with a sequence of 100 nucleotides would yield 4^{100} comparisons.

Thus to efficiently use the limited room for in vitro experiments, a preselection of good candidates by an in silico model is warranted. Many algorithms with different approaches have been proposed, which include:

- The Vienna RNA package contains an RNA inverse fold algorithm based on dynamic programming introduced in Hofacker et al. (1994).
- INFO-RNA is introduced in Busch and Backofen (2006). This algorithm also uses a dynamic programming approach for the initial sequence but introduces a stochastic local search method to optimize this initial sequence to a final solution.
- Taneda (2011) introduced MODENA, a Genetic Algorithm approach.
- RLIF introduced by Bernatavicius (2019) is an algorithm based on a reinforcement learning method with a convolutional neural network at its core.
- A number of other algorithms, of which some will be discussed further.

One very interesting approach is used in Lee et al. (2014) where the RNA inverse folding is converted into an online game. Each RNA inverse folding problem is treated as a puzzle that can be solved by changing the nucleotides of the sequence. Using the gathered

data via crowdsourcing they were successful in the conversion of human strategies of solving RNA inverse folding puzzles into new RNA design rules. An algorithm was designed according to these new design rules, which significantly outperformed the existing RNA inverse folding algorithms at that point in time.

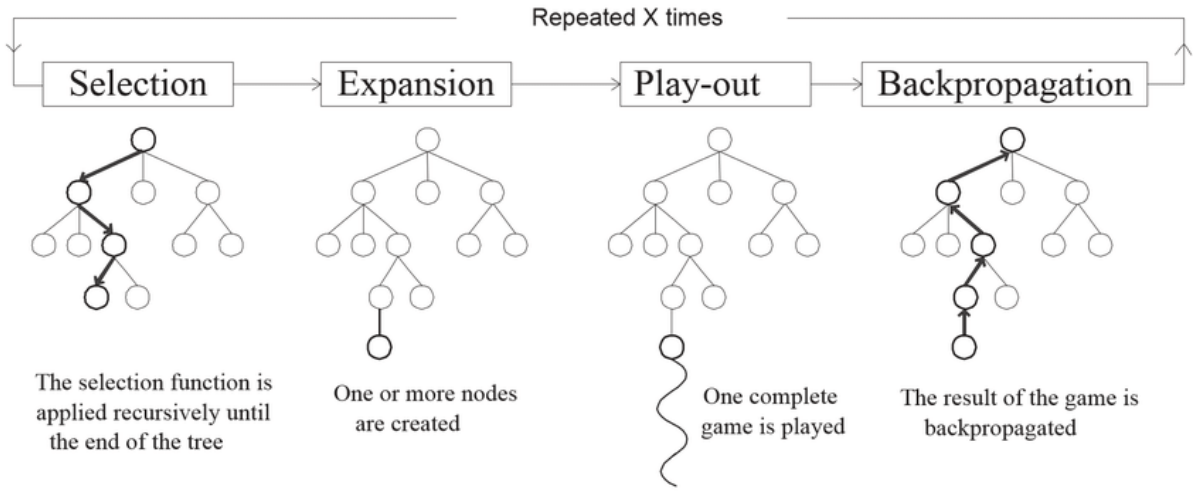


Figure 1: Schematic overview of the basic four steps of MCTS (from Chaslot et al., 2008).

The state space of a particular RNA inverse folding puzzle with length L consists of all possible sequences of length L. The RNA inverse folding problem is thus a problem with a large state space. The challenge with employing algorithms to solve problems with large state spaces is how to efficiently use the computational resources available. A method of guiding the exploration of the state space and use the gathered information about the state space (called exploitation) is Monte Carlo Tree Search (MCTS) (Chaslot et al., 2008). This method is a loop of four distinct steps: selection, expansion, simulation, backpropagation (depicted in Figure 1). At first, a node to expand is selected based on selection strategy which defines the balance between exploration and exploitation of the state space. One such strategy is calculating the upper confidence bound $U(s, a)$ of every node based on Equation 1. Here, $Q(s, a)$ is the expected reward for taking action a from state s, $N(s, a)$ is the number of times action a is taken from state s and $P(s, a)$ is the initial estimate of taking action a from the state s determined by the policy. This equation also contains a hyperparameter c_{puct} which controls the balance between exploration and exploitation. After calculating the U-values for every possible action at a node, the action with the highest upper confidence bound $U(s, a)$ is selected.

$$U(s, a) = Q(s, a) + c_{puct} \cdot P(s, a) \cdot \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \tag{1}$$

Once arrived at a leaf node, this node is expanded by adding nodes that represent all possible actions at that node. Then one random game is simulated until a terminal node is reached and the game has ended. The end score of the random game simulation

is calculated and backpropagated up the selection path. Multiple variants and strategies have been formulated for MCTS, such as simulating multiple random games at once, parallelization of the MCTS search process, using a non-random policy for the simulations and different selection strategies. MCTS algorithms have successfully been employed for solving RNA inverse folding problems, such as in Yang et al. (2017), Portela (2018) and Cazenave and Fournier (2020).

AlphaGo Zero is a recent reinforcement learning approach that uses MCTS at its core (Silver et al., 2017). Instead of the classic way of dealing with a leaf node in MCTS, AlphaGo Zero uses a deep learning model to predict the policy and the value of that node. Secondly, it uses the concept of self-play in which it plays against another instance of itself. This results in data about which moves constituted a win, a draw, or a loss. By simulating N games new training data is gathered for training the deep learning model. This in turn improves the policy and value prediction, which then improves the quality of the newly gathered training data. This reinforcement learning approach makes it possible to train an algorithm without any prior human knowledge about the game, only basing its learning process on the end result of the games it plays. In board games such as Go this algorithm demonstrates that it can employ human-like strategies and can achieve superhuman performance (Holcomb et al., 2018).

The goal of this project is to explore and find a way to implement an AlphaGo-like algorithm for solving RNA inverse folding problems.

2 Implementation

To be able to implement an AlphaGo like algorithm we need to explore questions such as:

- how do we represent an RNA secondary structure?
- how do we represent a state and which architecture do we employ for the policy?
- in the future even value prediction?

2.1 Structure representation

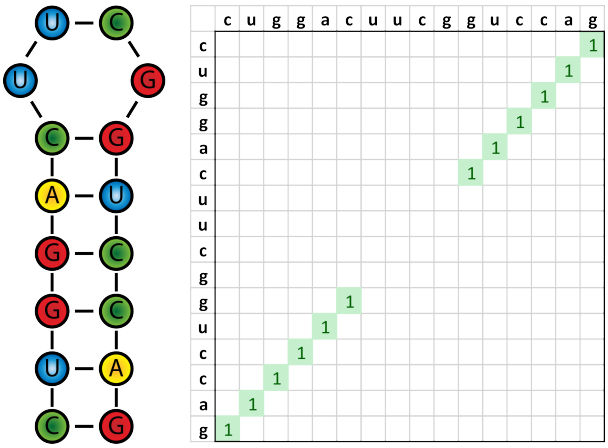


Figure 2: A schematic overview of the secondary structure of an RNA molecule (on the left) represented in a structure matrix (on the right). Each cell represents a combination of nucleotides and gets a 1 if these nucleotides have a pairing.

Secondary structures of RNA molecules are usually denoted with the dot-bracket notation. In this notation a pairing of nucleotides is denoted as opening and closing brackets and non-pairing nucleotides as a dot. For example, the structure in Figure 2 is denoted as ((((((....)))))). This dot-bracket notation has the benefit of being a compact notation, humanly readable and widely adopted. However, this notation has its limitations as well. First, the notation does not consider pseudoknots, base-pairing of loops with the regions outside the stems flanking them. Second, because the notation is sequential, you will need to go through the sequence to find the corresponding paired nucleotide. Especially the last limitation is a problem for computational models to extract features of the secondary structure. Therefore, we propose in this experiment a different secondary structure notation by representing the structure in a matrix. Both axes of this structure matrix represent the nucleotides in the RNA sequence, which means that each cell represents a pair of nucleotides. A value of 1 is given for all pairings and a value of o

is given otherwise. In Figure 2 an example is shown of a secondary structure of an RNA molecule represented by the structure matrix representation.

2.2 State representation

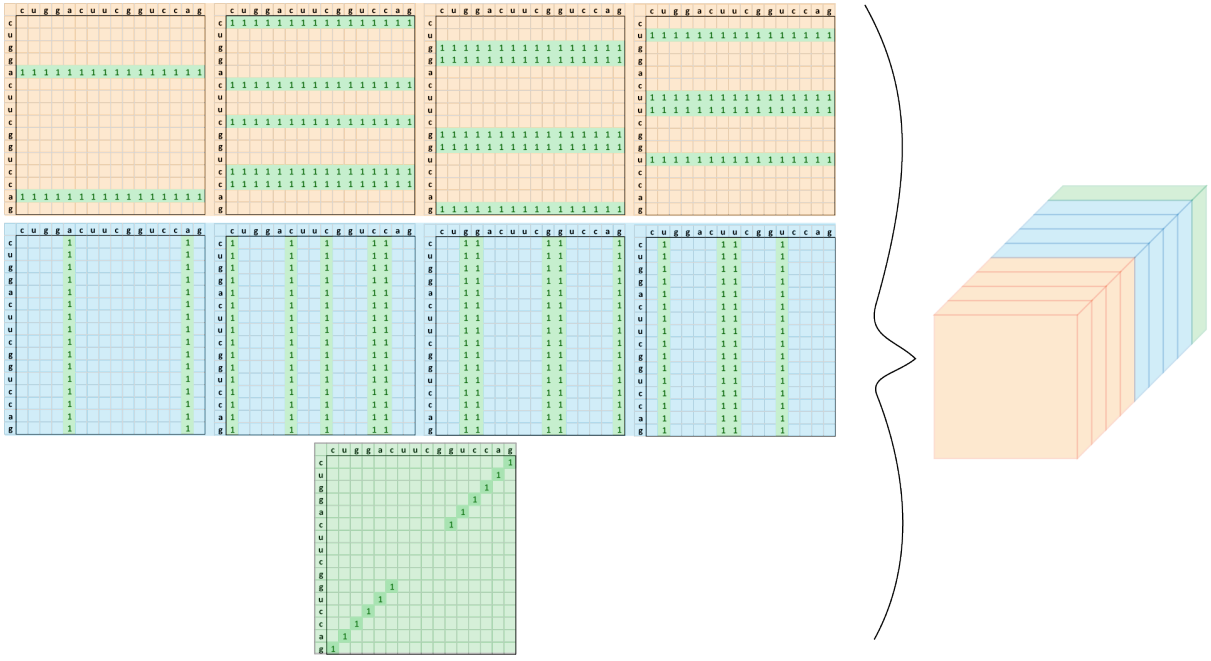


Figure 3: A schematic overview of the sequence state representation based on the example in Figure 2. Each cell represents a pair of nucleotides. The orange matrices represent the first nucleotides. The blue matrices represent the second nucleotides. The green matrix is the structure matrix of the target as in Figure 2.

We define the state space of the inverse folding problem as all the possible sequences with the length of the target secondary structure. A state thus represents one particular sequence of nucleotides. The denotation of a sequence of nucleotides Adenine, Cytosine, Guanine and Uracil with the letters A, C, G and U has the same benefit as the dot-bracket notation for the structure: a compact notation, humanly readable and widely adopted. The important limitation of this notation is that the structure presentation deals with pairings of nucleotides, thus direct comparison of the two representations is not possible. The solution is denoting the state in the same way as the structure: in a matrix where both axes of this structure matrix represent the nucleotides in the RNA sequence. Eight of these matrices would be needed to represent a state, two for each nucleotide. Each cell in the matrix represents a pair of nucleotides, thus each nucleotide is seen twice, once as the first of a pair and once as the second of a pair. In Figure 3 an example is shown of a state represented by the state matrix representation. For further reference, we will call this the **sequence state**.

An alternate strategy is to use the secondary structure of the state as input for a model (see Figure 4). This would be in the same representation as the structure

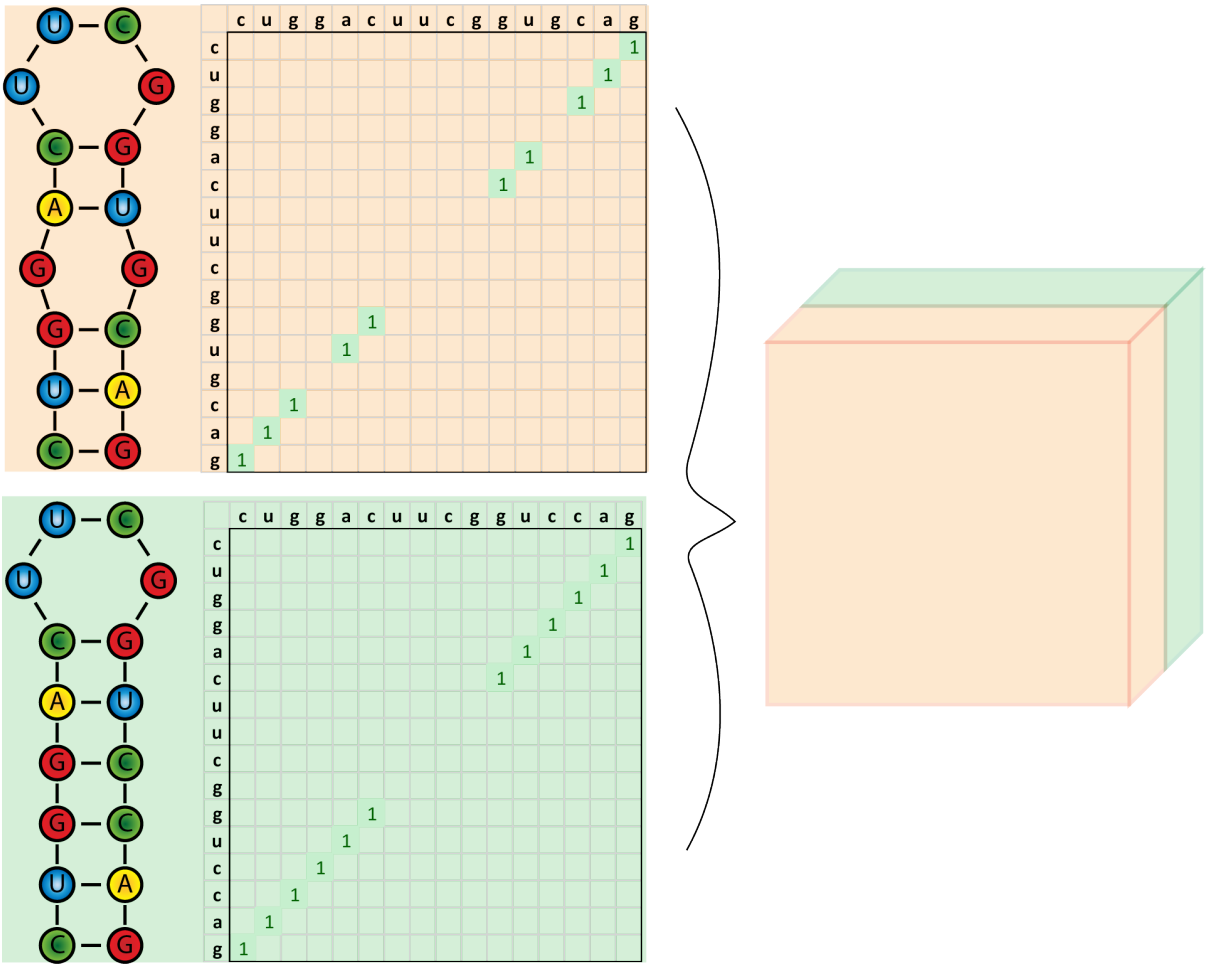


Figure 4: A schematic overview of the structure state representation. Each cell represents a pair of nucleotides. The orange matrix represent the current state. The green matrix represents the target.

presentation mentioned earlier. The benefit of using the structure of the state instead of a representation of the state itself is that only 2 channels of input information are needed (two structures: current state and target) instead of 9 channels (8 for current state plus one for target structure). Secondly, by using the structure of the current state instead of the state itself, features between the current structure and the target can be extracted by a model, which might give a better result if the first strategy fails. For further reference, we will call this the **structure state**.

2.3 Neural Network Architecture

At the core of an AlphaGo-like algorithm is a neural network that translates a state to a policy, a value or both, which then can be used by the MCTS. Our neural network needs to be able to deal with variable input sizes, variable output sizes and spatial relations. The neural network architecture of Silver et al. (2017) deals with spatial relations, but uses dense layers as well for translating extracted features to the desired output. Though

dense layers are by their very nature well equipped for this translation task, they are fixed in dimensional size. Therefore this architecture cannot deal with variable input and output sizes without heavy modifications.

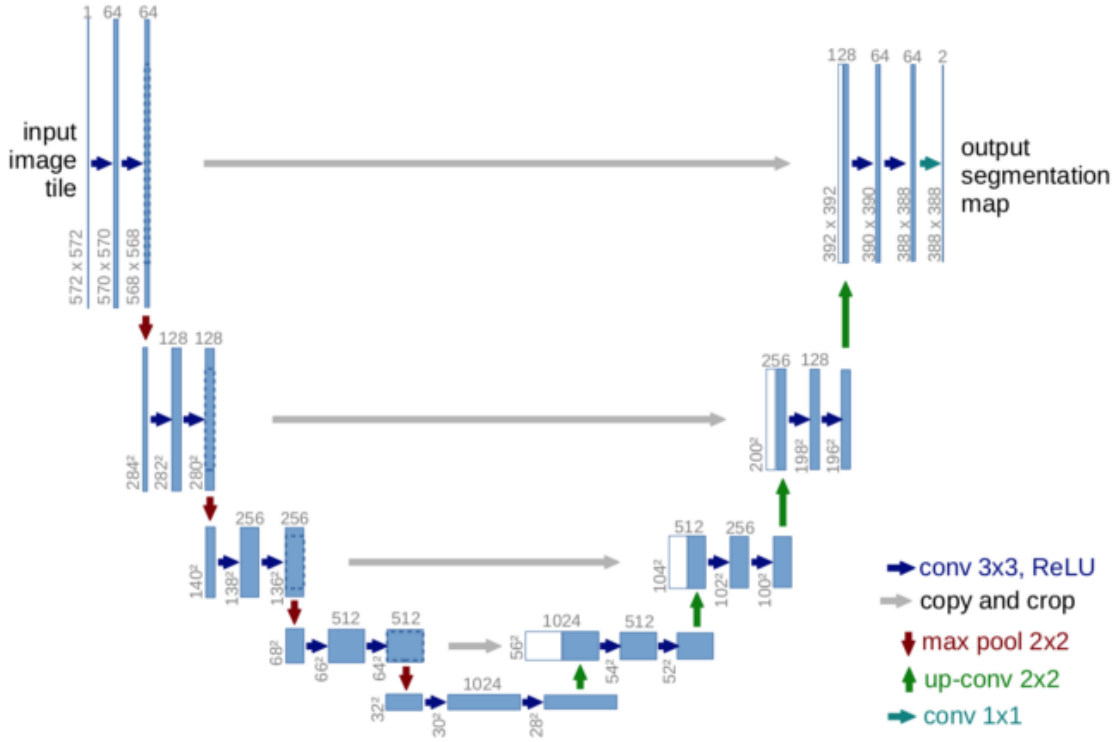


Figure 5: A schematic overview of the original U-Net architecture (from Ronneberger et al. (2015)). This neural network extracts features from the input image with which it can build the output image.

Fully convolutional neural networks are in theory able to deal with variable input sizes, variable output sizes and spatial relations. The fully convolutional neural network U-Net developed by Ronneberger et al. (2015) translates an input image to an output image (see Figure 5). It uses convolutional and pooling layers to extract features from the input image represented in feature maps called the encoder part. These feature maps are then upscaled in the decoder part with convolutional and pooling layers to the desired output image. What makes the U-Net architecture exceptional is that it extracts features on different levels that are carried over to the corresponding decoder part. Because of this U-Net can recognize and use local and global features of the input image to build the output. The combined use of local and global features is essential to counteract the chaotic nature of RNA inverse folding.

One of the possible outputs of such a neural network is a probability tensor. A schematic example of the probability tensor is provided in figure 6. The probability tensor provides the probabilities of all nucleotide combinations at each pairing location. All possible pairing locations are represented in a matrix of $lengthXlength$ and each

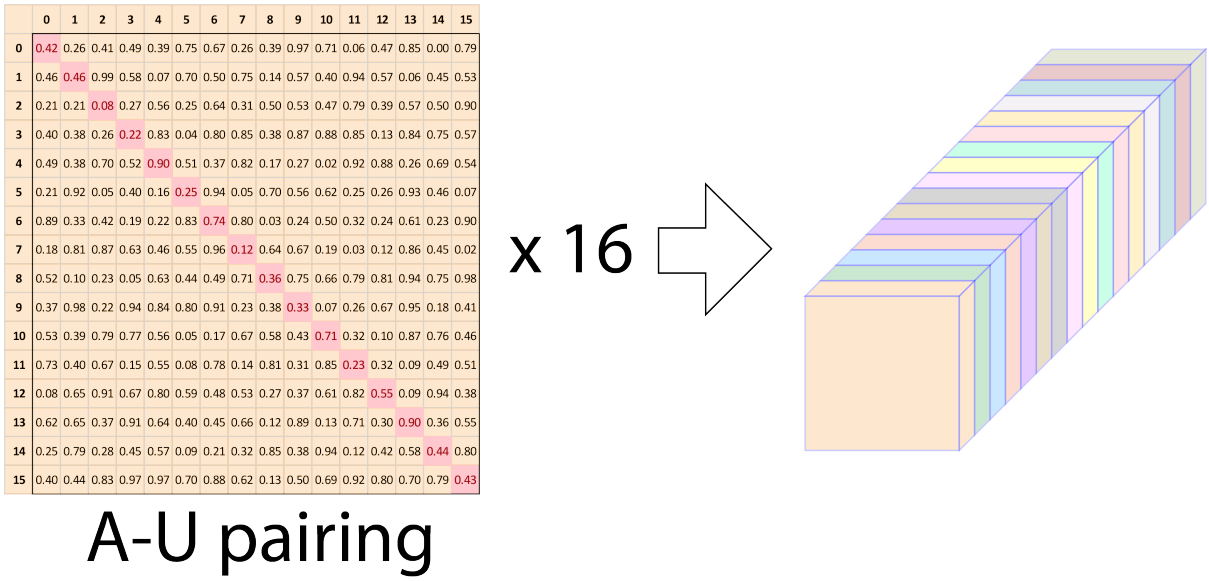


Figure 6: A schematic overview of the probability tensor. Each cell in the orange matrix represents a location of nucleotide pairs. 16 of these matrices are needed for all combinations of nucleotide pairs. The red diagonal will be discarded, as a nucleotide can't pair with itself.

matrix represents a combination of nucleotides. The order of nucleotides withing these combinations is important, thus the AU matrix and the UA matrix are distinct. The probability tensor covers the full action space: at each possible pairing location there are 16 possible nucleotide combinations. Based on this probability tensor a policy can be formulated for the MCTS.

2.4 Solving RNA inverse folding problems

To start solving an RNA inverse folding problem we need a starting nucleotide sequence. This can be a sequence of adenine or a randomly generated nucleotide sequence. However if our U-Net architecture is able to learn features and provide a decent probability tensor, we can use this to generate a best guess nucleotide sequence. This benefits us in at least three ways. First, it gives the algorithm a better starting position, which on average is closer to the solution. This is likely to increase the chance of reaching a solution and cost less computational power to get to it. Second, the probability tensor it provides might be used for determining follow-up moves. Third, by providing a better starting position the second part of the algorithm can be more fine-tuned in closing the resulting gap instead of having to get to a solution from scratch. For further reference, we will call the model which provides the probability tensor for this starting point the **init model**.

From the starting state an MCTS tree is initialized. To determine the next move N amount of MCTS simulations are done. When a leaf node is encountered, the policy and the value of this node are determined. The policy will be constituted based on either the

sequence state or structure state strategy, according to which one of these strategies seems to work best. Including all possible moves in the policy is computationally expensive. For every possible move, the upper confidence bound $U(s, a)$ must be calculated for each step in the simulation. Because our algorithm makes moves in nucleotide pairs the total amount of moves at any given state would be $lengthXlengthX16$ which is the size of the probability tensor. Given that a typical RNA sequence length in question is from 50 up to about 400 nucleotides, this quickly amounts to huge numbers. Only the best X amount of moves will thus be included in the policy to limit the computational resources needed for the MCTS simulations. For further reference, we will call the model to determine the policy based on the state (whichever strategy is used), the **state model**.

The value of the node for these experiments will be based on the Hamming distance between state and target. The use of the Hamming distance for formulating a score has been suggested by Runge et al. (2018). Ideally, the value of the node will be determined by a value model so the algorithm can learn how to guide itself to a solution, however this is out of the scope of this project.

3 Materials and methods

All experiments are written in Python 3.7 in the form of a Jupyter Notebook and executed through Google Colab using their GPU runtime. For the Neural Network architecture and handling, Pytorch version 1.8.1 is used for CUDA version 10.1. All folding of RNA sequences are done with the ViennaRNA folding algorithm using version 2.4.17 and using the parameters of Zuker et al. (1999). The code for all experiments is available at request through the Leiden Institute of Advanced Computer Science.

3.1 Neural Network Architecture

The model is based on the U-Net architecture from Ronneberger et al. (2015), with the following settings or differences. First, the input is zero-padded to the first multiple of 64, to ensure proper dimensions during reductions and upscaling in the model. Starting with 64 feature maps, in the encoder path, these are doubled on every level until 1024 feature maps are reached. The decoder path then halves the number of features every level back to 64. To mitigate the impact of possible vanishing gradients, leaky rectified linear unit layers were used throughout the model as well as batch normalization layers. Instead of max-pooling in the encoder path and up-sampling in the decoder path, convolutional layers are used to reduce or upscale the feature map size. To compensate for possible dimensional differences in respective encoder and decoder levels through reducing and upscaling zero-padding is added. As the skeleton for the code, a basic U-Net architecture in Pytorch was used (Witwitchayakarn, 2018). Two different heads can be attached to process the intermediate output: a policy head and a value head. In all the experiments only the policy head is used. The last convolutional layer of the U-Net architecture takes the 64 feature maps to 16 channels and cuts off the extra padding, thus the input dimensions except for the number of channels are the same as the output dimensions.

3.2 Random puzzle generation

For testing the different models, random puzzles are generated. Generating a random puzzle starts with generating a random nucleotide sequence. This sequence is then folded by the ViennaRNA folding algorithm. If the structure has at least one pairing of nucleotides it is added as a puzzle. All tests were done on puzzles with a random sequence between 50 and 400 nucleotides.

3.3 Scoring

The score of a nucleotide sequence as a solution to a target structure is calculated as follows. First, the Hamming distance is calculated by summing up the number of different

characters between the dot-bracket notations of the proposed solution and the target. The Hamming distance is then normalized by dividing it with the length of the sequence and inversed to get a score between 0 and 1, with 1 being a perfect score.

3.4 Statistical tests

To determine the significance of all result comparisons, first, a Shapiro test is done to determine the normality of the data points. After this, all the results are put in a Levene test to determine homoscedasticity. Since we have the luxury of generating as many data points as we want for the results and the normality and homoscedasticity are in question, more stringent statistical tests are chosen to determine the significance. If multiple results are being compared, first a Kruskal-Wallis test is done to determine whether any significant difference ($p < 0.05$) is found within all the results. After this, all combinations of results are compared with a Mann–Whitney U test (if the results are based on random puzzles) or a Wilcoxon Signed Rank test (in the case of results based on the Eterna benchmark set), corrected for multiple testing with the Bonferroni correction method.

3.5 Experiments

Multiple experiments were done to test different parts of the proposed algorithm.

- Experiment 1 tested the init models
- Experiment 2 tested the state models
- Experiment 3 tested the MCTS
- Experiment 4 tested the combination of the best init model, the best state model and the MCTS

3.6 Experiment 1: Init models

Experiment 1 is to test whether the U-Net architecture is able to learn features of the target structure and can provide a workable probability tensor for the nucleotide pairings. Four models were trained and tested: a control model with no training, a model trained on sequences of length 63, a model trained on sequences of length 127 and a model trained on sequences of variable lengths between and including 63 and 127. The lengths of 63 and 127 are chosen as they are very close to a multiple of 64. This mitigates the need for most of the zero padding. The training was done on randomly generated nucleotide sequences that are predicted to have a structure other than having no pairings. The models were trained for 70 epochs of 10 batches with a batch size of 64. Due to the

random nature of the training set generation, overfitting is not possible. Therefore the reduction in training loss is a good indicator for the fit of the model. After training the models are tested on 100 randomly generated puzzles. The models are given the structure as input and predict a probability tensor as output. This probability tensor is used to determine the most likely nucleotide sequence according to the model. Then, the structure is predicted by ViennaRNA based on this sequence. The Hamming distance between the predicted structure and target structure is then calculated and normalized to a range of 0 and 1 as a measure for performance. (Experiment 1A).

As a second test for each model, after a starting nucleotide sequence is predicted, the hundred best runner-ups for nucleotide pairings are calculated based on the probabilities tensor. This constitutes a hundred possible good moves to try out. First, all moves are shuffled and sequentially tried out. If a move is found that results in the same score, this move is accepted. If a move is found that exceeds the current score, the move is accepted and all moves are shuffled and tried out. The final score of 100 random puzzles is gathered for all four models. (Experiment 1C).

The same two tests are also done on the Eterna benchmark set. (Experiment 1B and 1D).

3.7 Experiment 2: state models

The aim of experiment 2 is to see whether one of the two chosen strategies for determining the policy is promising. The input of the sequence state model (experiment 2A and 2B) is constituted by 9 channels. Figure 3 gives an example of the 9 channels and the resulting input tensor. The first 8 channels represent the current sequence state and the last channel represents the target structure as in the init models. The first eight channels represent the four different nucleotides as the first and second nucleotide in a possible pairing, thus the first channel represents Adenine (A) at the first position and the fifth channel represents Cytosine (C) in the second position and so on. The training set is generated by generating random puzzles which define the targets. The training set input is generated by mutating the target nucleotide sequence at 2 to 10 places, thus altering it slightly compared to the targets. The training set output is then generated by first, folding the mutated sequence to its structure, second, translating the structure to a prediction ("state prediction") and finally, subtracting the target prediction from the state prediction. Training is done by training on 10 generated training sets of batch size 64, in which each epoch a new training set is generated to replace the oldest one to overcome overfitting without sacrificing too much computational power. For the training of this model, a validation set is also generated and monitored during training. The best model is saved based on the loss on this validation set.

Testing the sequence state model on 100 random puzzles is done by using the best init

model of experiment 1 for the initial prediction on which the initial sequence is generated, followed up by 10 sequential moves based on the prediction of the sequence state model. The result is calculated by subtracting the initial score (based on the initial sequence) from the final score. The control for this experiment is an untrained sequence state model.

Experiment 2C and 2D is to test the structure state model. The input of this model is constituted by the structure of the current state in channel 1 and the structure of the target in channel 2. Figure 4 gives an example of the 2 channels and the resulting input tensor. The output of this model is a matrix that represents all possible nucleotide pairing locations (one channel). The input of the training set is generated by mutating the target sequence between 4 to 20 times and using ViennaRNA folding to provide the structure of this mutated sequence. The output of the training set is generated by placing a 1 at every cell in the matrix where the two sequences are different. The training is done by training on 10 generated training sets of batch size 64, in which each epoch five new training sets are generated to replace the five oldest ones to overcome overfitting. For the training of this model, a validation set is also generated and monitored during training. The best model is saved based on the loss on this validation set.

Testing the structure state model on 100 random puzzles is done by using the best init model of experiment 1 for the initial prediction on which the initial sequence is generated, followed up by 20 sequential moves based on the prediction of the structure state model multiplied by the initial prediction by the init model. The result is calculated by subtracting the initial score (based on the initial sequence) from the final score. The control for this experiment is to use the initial prediction for the 20 sequential moves without multiplying with a prediction of the structure state model.

3.8 Experiment 3: MCTS

The MCTS implementation is based on the code provided by Thakoor et al. (2017). The value of a node is the score of the current structure. To test the MCTS algorithm 25 possible moves are selected based on the init model prediction. Then 250 MCTS simulations are done with the 25 moves set as the policy at each node. The result is calculated by the difference between the initial score and the best score. As a control, the 25 possible moves are sequentially applied and the result is calculated in the same way.

3.9 Experiment 4: Combination

The MCTS implementation for experiment 4 differs slightly from experiment 3. The value of a MCTS node is 1 if the sequence folds into the target structure. If the sequence does not fold into the target structure, the value assigned to a node is the score divided by 10. By dividing the score by 10 the initial value of a node lies within the range of

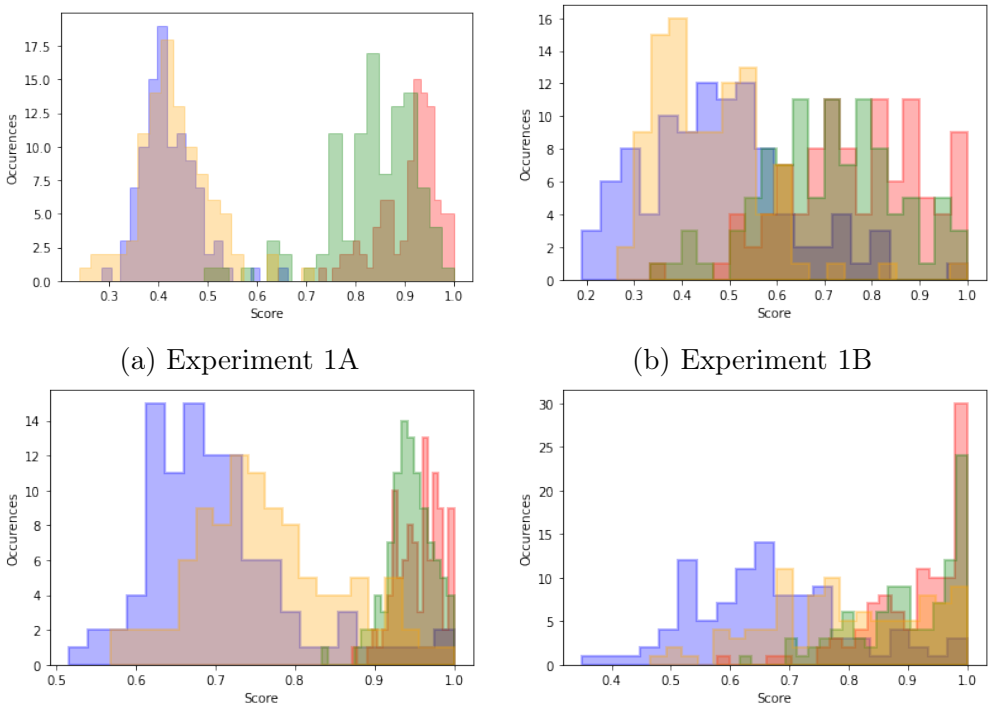
0-0.1. Though a score of 0.9 indicates that a structure seems to be close to the target structure, the solution might be far off and therefore the score should have a lesser weight in the value of the node. The policy of each node is provided by the multiplication of the init model prediction and the structure state model prediction and the best 20 moves are extracted. To solve a puzzle this algorithm will do 40 MCTS simulations and the most visited child node of the current node dictates the next move. After each move, ViennaRNA inverse folding is called to try to solve the problem with the new nucleotide sequence. If no solution is found, the next move is determined by the MCTS simulations and so on.

4 Results

4.1 Experiment 1: Init models

Experiment 1 is to test whether the U-Net architecture is able to learn features of the target structure and can provide a workable probability tensor for the nucleotide pairings. Four models were trained and tested: a control model with no training (control), a model trained on sequences of length 63 (model 63), a model trained on sequences of length 127 (model 127) and a model trained on sequences of variable lengths between and including 63 and 127 (model var). The models were tested and scored in four different settings:

- experiment 1A: initial output of the model on 100 random puzzles;
- experiment 1B: initial output of the model on the Eterna benchmark;
- experiment 1C: a simple solving algorithm based on the output of the model on 100 random puzzles;
- experiment 1D: a simple solving algorithm based on the output of the model on the Eterna benchmark.



(a) Experiment 1A

(b) Experiment 1B

(c) Experiment 1C

(d) Experiment 1D

Figure 7: Histograms of the scores of each model in the different experiments. Control, model 63, model 127, model var are shown in blue, red, green, yellow respectively. Control is an untrained model.

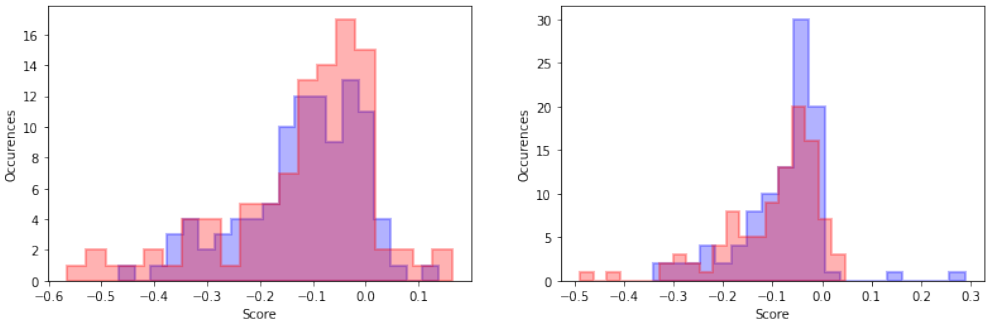
In Table 1 the final scores of the different models are shown for all tests and to visually aid in the results, the histograms are shown in Figure 7. Model 63 is the best performing model in all tests and is statistically significant in all tests for all models except for test 1d in which $p = 0.0307$ while the Bonferroni corrected significance threshold is $p < 0.00833$. Model var showed no statistically significant difference to the control in test 1c ($p = 0.0530$) and 1d ($p = 0.238$), but performed marginally better in tests 1c and 1d than control. In experiment 1D the number of solved puzzles for control, model 67, model 127 and model var is 3, 25, 23, 9 respectively.

Table 1: Final scores of control, model 63, model 127 and model var for each experiment 1. Model 63 shows the highest scores overall, while the scores of model var are very close to the control.

	control	**model 63**	**model 127**	**model var**
exp1a	0.420 (±0.055)	0.908 (±0.059)	0.833 (±0.093)	0.433 (±0.075)
exp1b	0.475 (±0.154)	0.769 (±0.140)	0.710 (±0.139)	0.459 (±0.116)
exp1c	0.702 (±0.092)	0.956 (±0.028)	0.945 (±0.029)	0.767 (±0.091)
exp1d	0.667 (±0.132)	0.916 (±0.083)	0.901 (±0.090)	0.794 (±0.133)

4.2 Experiment 2: state models

The aim of experiment 2 is to see whether one of the two chosen strategies for determining the policy shows promise. First, the results of the strategy based on the sequence state are shown (experiment 2A and 2B) followed by the results of the strategy based on the structure state (experiment 2C and 2D). In experiment 2B and 2D the ViennaRNA RNA inverse fold algorithm is used to help finding solutions.



(a) Experiment 2A (b) Experiment 2B

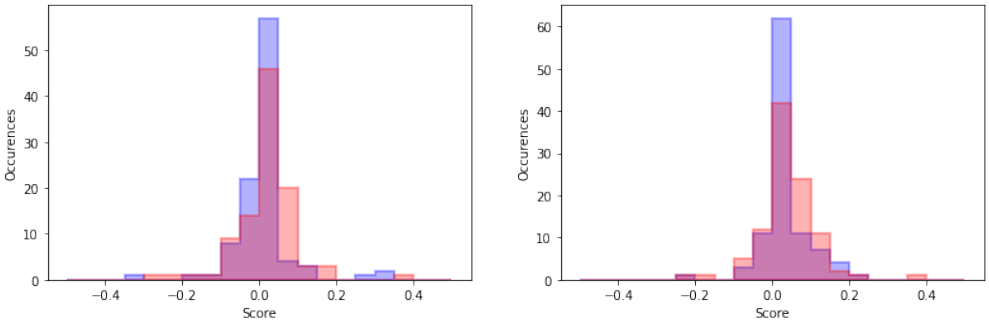
Figure 8: Histograms of the score difference of the **sequence state model** in the different experiments. Control and model are shown in blue and red respectively. In experiment 2B the ViennaRNA RNA inverse fold algorithm is used to help finding solutions. Control is an untrained model.

The results of the strategy based on the sequence state are shown as histograms of scores in Figure 8. In the experiment without the help of ViennaRNA RNA inverse fold

algorithm, both control and the sequence state model were not able to solve any of the 100 random puzzles. With the help of this algorithm control and the sequence state model were able to solve 4 and 5 puzzles respectively. Table 2 shows the mean and standard deviation of these final scores. Both means are negative, which means that on average, scores were worsened by the models. No significant difference was found between the control and the sequence state model with a p-value of 0.236 and 0.147 respectively for experiments 2A and 2B.

Table 2: Final score differences of control and the **sequence state model**. Both the model and the control show similar results. In experiment 2B the ViennaRNA RNA inverse fold algorithm is used to help finding solutions.

	control	**model**
exp2a	-0.120 (±0.115)	-0.117 (±0.142)
exp2b	-0.078 (±0.090)	-0.098 (±0.097)



(a) Experiment 2C (b) Experiment 2D

Figure 9: Histograms of the score difference of the **structure state model** in the different experiments. Control and model are shown in blue and red respectively. In experiment 2D the ViennaRNA RNA inverse fold algorithm is used to help finding solutions. Control is the use of moves based on the initial prediction without the help of the structure state model.

Table 3: Final score differences of control and the **structure state model**. The model shows a slightly better result than the control. In experiment 2D the ViennaRNA RNA inverse fold algorithm is used to help finding solutions.

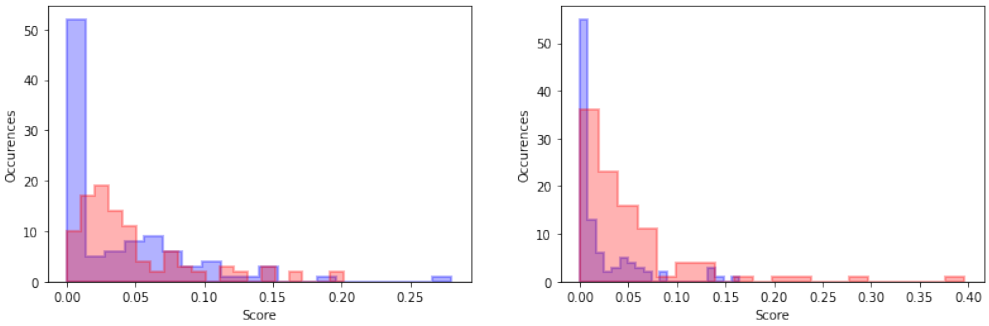
	control	**model**
exp2c	0.003 (±0.076)	0.020 (±0.077)
exp2d	0.026 (±0.058)	0.044 (±0.097)

The results of the structure state model versus control are shown in Figure 9 and Table 3. Both with and without the use of the Vienna RNA inverse folding algorithm, the structure state model shows a mean improvement of the scores. In experiment 2C the structure model solved 16 puzzles versus 3 by control and 18 puzzles versus 4 in experiment 2D. The difference in scores versus control is also significant for experiments

2C ($p = 1.49 * 10^{-5}$) and 2D ($p = 1.30 * 10^{-3}$). A direct comparison of the score differences between experiments is not possible due to the differences of scoring.

4.3 Experiment 3: MCTS

The goal of experiment 3 is to see whether the implementation of our MCTS algorithm together with our init model shows an improvement on its own. Experiment 3A is done with 250 moves and 1000 MCTS searches, while experiment 3B is done with 25 moves and 250 MCTS searches.



(a) Experiment 3A (b) Experiment 3B

Figure 10: Histograms of the score differences of the MCTS algorithm in the different experiments. Control and MCTS are shown in blue and red respectively. Control applies the moves sequentially.

Figure 10 shows the histograms of the differential scores between the initial scores and the best scores. The mean differential scores are shown in Table 4. In both experiments the MCTS performed better than the control. The difference between both results in the experiments is significant for experiments 3A ($p = 7.14 * 10^{-4}$) and 3B ($p = 7.25 * 10^{-5}$).

Table 4: Final score differences of control and the MCTS algorithm. The MCTS algorithm shows a slightly better result than the control.

	control	**model**
exp3a	0.037 (±0.050)	0.048 (±0.044)
exp3b	0.022 (±0.036)	0.049 (±0.060)

4.4 Experiment 4: combination

Experiment 4 combines the previous experiments to test the efficacy of the approach. It solved 13 out of the 100 puzzles with a mean score of 0.798 (±0.134). The initial score after the first predicted solution based on the initial model was 0.717 (±0.132). The algorithm thus improved 0.081 on the score.

5 Discussion

The goal of experiment 1 was to test whether the U-Net architecture is able to learn features of the target structure and can provide a workable probability tensor for the nucleotide pairings. Except for the init model trained on variable sequence sizes, all init models showed a clear improvement versus control. The U-Net architecture thus is able to predict a probability tensor for nucleotide pairings. More interesting is the question of why the init model trained on variable sequence sizes did not show a clear improvement. Our U-Net model extends the width of the nucleotide sequence to multiples of 64 by zero padding. Lengths of 63 and 127 are strategically chosen to be very close to these multiples so less zero-padding is needed for the puzzles in the training set. Secondly, by choosing 63 or 127 as length the puzzles remain in the same multiple of 64, while the variable model has puzzles within lengths of 0-64 and 65-128. The problem with the variable model might come from training on puzzles with variable lengths in multiple ranges of 64 (for example 0-64 and 65-128). This could be tested by having a variable model trained on sequence lengths that are completely within the range of a multiple of 64 (for example, either 0-64 or 65-128). Lastly, because of the variable sizes of the puzzles in the variable model, variable zero-padding is needed, which might have functioned as added noise to such a degree that during training the model could not converge to a local optimum.

To answer the question whether we can extend the init model to also deal with input of the current state in conjunction with the target structure, experiment 2 was done. The first approach where the current state is encoded in 8 different channels did not show any improvement versus control, with control, in this case, being an untrained state model. It is possible that the model was not able to learn anything due to the amount of input, which can actually introduce too much complexity which cannot be resolved. Secondly, the chosen representation of the sequence state using 8 channels might make feature recognition impossible within the chosen architecture. An interesting follow-up experiment would be if the nucleotide state represented in the same way as the output probability tensor would be able to show an improvement versus control. Though the question is whether such a model would outperform the naive solution of just subtracting the input tensor from the init model prediction.

The second type of state model is based on the input of the structure that the current sequence state folds into in conjunction with the target structure. This model would provide an output matrix with information on which nucleotide pairings need to change. The results show that the state model provides an improvement versus control, in which the control is simply the init model prediction without multiplying it with the output of the state model. Such a state model is useful for changing the weights of the initial probability tensor predicted by the init model based on the current state and is, therefore,

a candidate to be used within the frame of an MCTS algorithm.

The MCTS experiment results show that the MCTS algorithm we propose can use the information provided by the probability tensor predicted by the init model and provide an improved solution versus control. This result is expected because the MCTS can rearrange all the possible moves in a combination of moves that provide a better overall result than trying each move and accepting the move only if it provides a better score.

Experiment 4 combines the init model for the prediction of the initial sequence, the state model for providing the policy at every leaf node and the MCTS for exploring and exploiting the state space. The combination model was able to solve 13 out of 100 Eterna benchmark puzzles with a mean score of 0.798. This means that the combination model is able to navigate the state-space and provide solutions to some complex puzzles. However, the naive solution of trying out the best 100 moves in random combinations in experiment 1D showed better results on the Eterna benchmark set, with 25 solved puzzles by model 67 and a mean score of 0.916. This can be due to the combination model using just 20 moves per leaf node and having just 40 simulations per move. These constraints severely limit the MCTS in exploring the state space. Secondly, the combination model has a loop problem in which the model gets stuck in the same loop of trying out moves. Though the model catches these loops and tries three times to get out of the loop by mutating the sequence of the current state, it usually ends up in another loop. This could be caused by the state model giving very similar policies to the leaf nodes, effectively limiting the model to just 20+ moves. Lastly, the model calculates the value of all the nodes based on the current score and the policy. However, this could be problematic, because the value based on this might drive the model to a local optimum with a limited possibility to escape. In the future, a value model trained with the use of reinforcement learning might drive the model towards moves that have more potential to reach a solution rather than a higher score.

The maximum potential of our approach is closely related to the strength of the structure prediction algorithm, because all training data is derived from this algorithm. Currently the ViennaRNA algorithm is used, but in the future we would like to use more novel approaches. Such a break-through deep learning approach has been demonstrated in protein structure prediction by AlphaFold 1 and recently its successor AlphaFold 2 (Callaway, 2020). While AlphaFold doesn't use the MCTS and reinforcement learning techniques as in AlphaGo, it shows remarkable promise for this project if an Alphafold-like algorithm can be constructed for RNA structure prediction.

6 Conclusion

The goal was to define and explore the pre-requisites to make an AlphaGo-like algorithm work for the RNA inverse folding problem. In multiple experiments, we explored different neural network architectures, representations of the state space and combined them all together for a proof of concept. The U-Net architecture was able to recognize patterns based on the chosen structure input and was able to translate this input to a probability tensor for all possible nucleotide pairings. We showed that the same U-Net architecture was also able to provide a probability matrix of where to change nucleotide pairings based on the current structure in conjunction with the target structure. These two outputs together form a specific policy vector for each node in the MCTS. Lastly, we showed that the MCTS based on this policy generating mechanism and using values based on the Hamming distance can solve RNA inverse folding problems. The next step would be to get to the full realization of an AlphaGo like algorithm. The values of the MCTS nodes would need to be based on a deep learning model which is trained using reinforcement learning. This reinforcement learning can at the same time improve the models that provide the outputs for the policy vector. Looking at what AlphaGo did for the world of Go, this provides a hopeful outlook to the future of RNA inverse folding.

7 References

Bernatavicius, A. (2019). *Rlif: Reinforcement learning-based rna design tool* (Master's thesis). Leiden University.

Busch, A. & Backofen, R. (2006). Info-rna—a fast approach to inverse rna folding. *Bioinformatics*, *22*(15), 1823–1831.

Callaway, E. (2020). 'it will change everything': Deepmind's ai makes gigantic leap in solving protein structures. *Nature*.

Cazenave, T. & Fournier, T. (2020). Monte carlo inverse folding. *arXiv preprint arXiv:2005.09961*.

Chaslot, G., Bakkes, S., Szita, I. & Spronck, P. (2008). Monte-carlo tree search: A new framework for game ai. *AIIDE*.

Hofacker, I. L., Fontana, W., Stadler, P. F., Bonhoeffer, L. S., Tacker, M. & Schuster, P. (1994). Fast folding and comparison of rna secondary structures. *Monatshefte für Chemie/Chemical Monthly*, *125*(2), 167–188.

Holcomb, S. D., Porter, W. K., Ault, S. V., Mao, G. & Wang, J. (2018). Overview on deepmind and its alphago zero ai. *Proceedings of the 2018 international conference on big data and education*, 67–71.

Lee, J., Kladwang, W., Lee, M., Cantu, D., Azizyan, M., Kim, H., Limpaecher, A., Gaikwad, S., Yoon, S., Treuille, A. et al. (2014). Rna design rules from a massive open laboratory. *Proceedings of the National Academy of Sciences*, *111*(6), 2122–2127.

Portela, F. (2018). An unexpectedly effective monte carlo technique for the rna inverse folding problem. *BioRxiv*, 345587.

Ronneberger, O., Fischer, P. & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *International Conference on Medical image computing and computer-assisted intervention*, 234–241.

Runge, F., Stoll, D., Falkner, S. & Hutter, F. (2018). Learning to design rna. *arXiv preprint arXiv:1812.11951*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al. (2017). Mastering the game of go without human knowledge. *nature*, *550*(7676), 354–359.

Taneda, A. (2011). Modena: A multi-objective rna inverse folding. *Advances and applications in bioinformatics and chemistry: AABC*, *4*, 1.

Thakoor, S., Jhunjhunwala, M., Kumar, S., Tyurin, E., MBoss, Habjan, J. & Lawson, A. (2017). Alpha Zero General. https://github.com/suragnair/alpha-zero-general

Witwitchayakarn. (2018). U-net with pytorch. https://www.kaggle.com/witwitchayakarn/u-net-with-pytorch

Yang, X., Yoshizoe, K., Taneda, A. & Tsuda, K. (2017). Rna inverse folding using monte carlo tree search. *BMC bioinformatics*, *18*(1), 1–12.

Zuker, M., Mathews, D. H. & Turner, D. H. (1999). Algorithms and thermodynamics for rna secondary structure prediction: A practical guide. *Rna biochemistry and biotechnology* (pp. 11–43). Springer.