Designing RNA Sequences by Self-play

Stephen Obonyo[1][a], Nicolas Jouandeau[1][b] and Dickson Owuor[2][c]
[1]*LIASD, Paris 8 University, Paris, France*
[2]*School of Computing and Enfineering Sciences, Strathmore University, Nairobi, Kenya*

Keywords: Self-playing, RNA Inverse Folding, Reinforcement Learning, Computational Biology.

Abstract: Self-play (SP) is a method in Reinforcement Learning (RL) where an agent learns from the environment by playing against itself until the policy and value functions converge. The SP-based methods have recorded state-of-the-art results in playing different computer games such as Chess, Go and Othello. In this paper, we show how the RNA sequence design problem where a sequence is designed to match a given target structure can be modelled through the SP while performing the state-value evaluation using a deep value network. Our model dubbed RNASP recorded the best and very competitive results on the benchmark RNA design datasets. This work also motivates the application of the self-play to other Computational Biology problems.

1 INTRODUCTION

Ribonucleic Acid (RNA) plays a critical role in biological processes such as protein synthesis and gene expression. The design of RNA sequences which folds to match a given target structure is often referred to as RNA Inverse Folding. The RNA Inverse Folding is key to both *in vitro* and *in vivo* research. It also has the potential to positively impact disciplines such as synthetic biology, design of functional RNA molecules, biotechnology, drug design and medicine in general.

An RNA molecule is composed of four unique bases; Guanine (G), Cytosine (C), Adenine (A) and Uracil (U). The RNA primary structure is constituted by a sequence of these bases. The molecular forces exhibited by bases enforce the base pairing between the A-U and G-C (the Watson–Crick base pairs). While the latter base pairings are the most common, G-U also tends to form base pairs in certain RNA locations such as junctions and pseudoknots.

The RNA sequence design is contingent on a predefined target structure e.g. the designed sequence must fold according to the given structural target. Figure 1 (a) shows an example of the target structure. It is denoted using the Dot-bracket notation where an opening and closing parenthesis represents the base pairing locations which are assigned A-U, G-C, G-U or their reverse while the dots represent single base locations which are assigned A, U, C or G.

Following the assignment step, the designed sequence (Figure 1 (b)) is then folded to obtain the predicted structure which is then compared with the target structure for a match. See Figure 1 for the distinction between target structure, designed sequence and predicted structure.

The quality of the designed RNA sequence can be determined through different methods; (i) Hamming loss (ii) Minimum Free Energy (MFE) and (iii) the GC content. The Hamming loss is calculated between the predicted structure and target structure. The Hamming loss between predicted structure and target sequence can be calculated using the formulae $1 - accuracy\_loss$. In the example shown in Figure 1, the Hamming loss is 0 since the target structure - (((((●●●●●●))))) - is the same as the predicted structure - (((((●●●●●●))))).

The Minimum Free Energy (MFE) determines the stability of the sequence which affects its functional effectiveness. Molecules with lower MFEs tend to be more stable (Trotta, 2014). The designed sequence must have the right proportion of G's and C's as this also affects the functional effectiveness of the molecule (Risso et al., 2011). The GC content is calculated by Equation 1 as a proportion of G and C values to that of the total bases contained in the sequence. A value within the range of 50 and 60 is generally preferred. The designed sequence (CU-

[a] https://orcid.org/0000-0002-6878-7802
[b] https://orcid.org/0000-0001-6902-4324
[c] https://orcid.org/0000-0002-0968-5742

Obonyo, S., Jouandeau, N. and Owuor, D.
Designing RNA Sequences by Self-play.
DOI: 10.5220/0011550300003332
In *Proceedings of the 14th International Joint Conference on Computational Intelligence (IJCCI 2022)*, pages 305-312
ISBN: 978-989-758-611-8; ISSN: 2184-2825

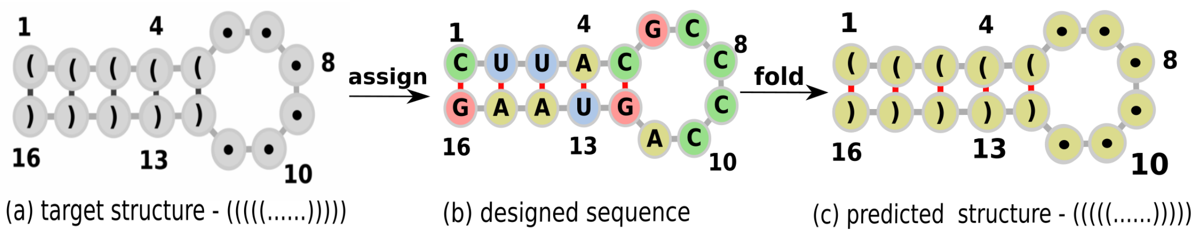
1
4
8
16
13
10
assign
C U U A C G C C C C A C G U A A G
fold
(a) target structure - (((((......)))))
(b) designed sequence
(c) predicted structure - (((((......)))))

Figure 1: Target structure (a). The locations are marked by numbers 1-16 for convenience. Locations (1, 16), (2, 15), (3, 14),(4, 13) and (5, 12) are paired - opening and closing parentheses - thus can be simultaneously assigned any value from {GC, CG, AU, UA, GU, UG}. Similarly, locations 6 - 11 are single sites - dots - and can be assigned any value from {A, U, C, G}. In the designed sequence (b) location (1, 16) is paired hence assigned C and G simultaneously, similarly, location (3, 14) is assigned U and A respectively. The same concept applies to positions (2, 15), (3, 14), (4, 13) and (5, 12). In contrast, locations 6 - 11 are not paired and thus have been assigned value from {A, U, C, G}. The designed sequence is folded by Zuker and Stiegler (1981) resulting in a predicted structure (c). The Hamming loss between the predicted structure and target structure, in this case, is 0 thus the designed sequence fold to match the target structure.

UACGCCCCAGUAAG) in Figure 1 for example has a GC content of 56.23%.

$$GC = SUM(G,C)/SUM(G,C,A,U) \qquad (1)$$

2 RELATED WORK

Several RNA sequence design methods have been proposed by different researchers. These models are heavily influenced by methods spanning different disciplines such as computer game playing, brute force, genetic and evolutionary search algorithms, dynamic programming, constraint satisfaction modelling, optimization, tree search, general AI search and thermodynamics.

Yang et al. (2017) proposed a Monte Carlo Tree Search (MCTS) which optimized both the GC content and the Hamming loss. The model also allowed the user to specify the target GC content value and using unique substitution rules, it designed RNA sequences which fold according to the target structure and the specified GC content value.

Cazenave and Fournier (2020) also proposed an MCTS-based model which optimized the Hamming loss and the GC content with a Generalized Nested Policy Adaptation (GNPA) algorithm.

Apart from the Tree search-based models, other researchers have also proposed dynamic programming-based models including the INFO-RNA (Busch and Backofen, 2006) which was reinforced by the local search to improve the candidate RNA sequences to optimize the Hamming loss between the target and predicted structure.

The method adopted by INFO-RNA is an extension of RNAinverse (Hofacker et al., 1994). This is one of the earliest RNA Inverse Folding models which has inspired several RNA Inverse Folding models.

RNA sequence design model proposed by (Andronescu et al., 2004) dubbed RNA-SSD also focused on minimizing the Hamming loss while reinforcing the candidate RNA solutions by a recursive stochastic local search. The method employed by the RNA-SSD has stark similarities with Levin et al. (2012) where the candidate RNA solutions were reinforced through the global search.

Genetic and Evolutionary algorithms have also been applied to the RNA sequence design problem. The AntaRNA model proposed by Kleinkauf et al. (2015b) is precisely inspired by the Evolutionary algorithm named Ant Colony. In this work, both the Hamming loss and the GC content were jointly minimized as well. The design of the AntaRNA was extended by Kleinkauf et al. (2015a) on a more complicated RNA design problem; the pseudoknotted structures.

The evolutionary-based model by Esmaili-Taheri and Ganjtabesh (2015) varies from the previous ones as it allowed the user to specify the energy constraints (the MFE) of the resulting sequence from the RNA Inverse Folding.

The Constraint satisfaction model, RNAiFOLD, by Garcia-Martin et al. (2013) employed a unique candidate solution method allowing for the specification of the GC content and certain RNA intermolecular properties. The ability to specify user-defined constraints is a property which has been explored by Modena (Taneda, 2012); a genetic and multi-objective model which also jointly optimized both the Hamming loss and the GC content.

Runge et al. (2018) introduced a Proximal Policy Model (PPO) based model to design RNA sequences by directly modelling the best single bases or paired bases to select at any given time. In contrast to learning the policy or value functions which then inform the base assignment, a PPO (Schulman et al., 2017) model can directly learn the best action to select at any given time. The model was further augmented by meta-learning and local search methods recording competitive results on the benchmark datasets.

3 CONTRIBUTION

We propose a self-play RNA sequence design agent (RNASP) which learns the state-value function through SP to improve its policy over time. The state-value evaluation function is represented by a Deep Neural Network which allows for wider state generalization as the agent explores the environment. According to the best of our knowledge and extensive related work review, this is the first research which investigates how the SP concept can be leveraged in RNA sequence design. The application of the SP motivates the application of the concept to other Computational Biology problems which are majorly characterized by the scarce labelled datasets.

4 METHODS

4.1 Self-play

Self-play (SP) can be defined as an *"algorithm that learns by playing against itself without requiring any direct supervision* (Bai and Jin, 2020). SP has been widely applied in Reinforcement Learning (RL) for game playing and general game intelligence to improve agent's state-action or state-value evaluation functions.

General self-play logic in computer games is as illustrated by Algorithm 1. Given a state, the next best move is selected and then applied to the environment. When the terminal state is reached, the value of the terminal state is then assigned to all the observed states during the episode.

The SP has been successfully applied in computer games e.g. Alpha Go (Silver et al., 2016) and Alpha Zero (Silver et al., 2017). The former and latter recorded state-of-the-art results in playing computer games such as Go and Chess. Alpha Go and Alpha Zero agents both learned through SP, however, the Alpha Go learning was reinforced by the expert knowledge moves which guided the move selection process.

Alpha Zero on the other hand learned entirely through SP without any expert knowledge reinforcement. Its design incorporated a value network for evaluating the game state constrained within the range of $\{1,-1\}$ and a policy network which returned the optimal move at a given game state. Furthermore, in Alpha Zero, Monte Carlo Tree Search (MCTS) was used to improve the policy and bias the exploration weight value.

Algorithm 1: Self-play Logic in games. If it is win at the end of the game then a target of +1 is assigned to all the observed states otherwise -1. Given the observed states and targets, the agent can improve its policy or value function through supervised learning.

```
1  function selfPlay($state$) :
2      $X \leftarrow \emptyset$ ;
3      $y \leftarrow \emptyset$ ;
4      while not $state.terminal()$ do
5          $action \leftarrow state.$select_move( ) ;
6          $state \leftarrow state.$make_move($action$) ;
7          $X \leftarrow X \cup state.state\_value$ ;
8      for each $x \in X$ do
9          if $state.$win( ) then $score \leftarrow 1$ ;
10         else $score \leftarrow -1$ ;
11         $y \leftarrow y \cup score$ ;
12     return $X, y$ ;
```

Following the success of Alpha Go and Alpha Zero, the SP concept has since been extended to general game playing and game intelligence (Berner et al., 2019; Cazenave et al., 2020).

4.2 RNASP Design

Similar to Alpha Go, the RNASP does not require any expert knowledge regarding the RNA sequence design except the pairing rules, however, while Alpha Zero learned both the policy and state evaluation functions, RNASP only learns the state-value evaluation function using a Deep Neural Network (also referred to as the value network).

We model the SPRNA as an RL problem. Generally any RL problem can be formally described using five parameters $\{\mathcal{S},\mathcal{A},\mathcal{R},\mathcal{P},R,\gamma\}$; where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ set of possible actions, $\mathcal{P}$ the state transition model, R the reward and γ the discount factor (Sutton and Barto, 2018).

If $\mathcal{P}$ and R are known then solving the MDP is tractable through iterative or dynamic programming methods. Modelling $\mathcal{P}$ and R, however, for most RL problems can be resource-intensive and intractable leading to the sampling of the environment through methods such as Monte Carlo (MC) and Temporal Difference (TD) Learning.

When the number of states is large e.g number of states corresponding to the number of pixels in an image or video stream, a Deep Learning model can be used to approximate $\mathcal{P}$ and R. This concept has been illustrated by Mnih et al. (2013) and Schaul et al.

(2015).The RNASP RL constrains are defined as follows;

- **Action Space ($\mathcal{A}$)**: Set of valid actions is defined by Equation 2. If the current move location is paired the valid actions are also paired or unpaired otherwise.

$$\mathcal{A} = \begin{cases} \{GC, CG, AU, UA, GU, UG\} & paired(move) \\ \{A, U, C, G\} & otherwise \end{cases} \tag{2}$$

- **State Space ($\mathcal{S}$)**: Every symbol in state value is coded using the binary code sequences shown in Table 1 and the resulting values are concatenated. The state space of sequence $\mathcal{L}$ can therefore be define as $\bigcup_{i=1}^{|\mathcal{L}|} \mathcal{C}_i$ where $\mathcal{C}_i$ any binary code of a given state value symbol and i is the the base/nucleotide assignment location and $|\mathcal{L}|$ the length of the RNA sequence to be designed.
- **Reward ($\mathcal{R}$)**: The immediate reward in every non-terminal state is 0. In the terminal state (all paired and unpaired locations assigned) the reward is 1 if the Hamming loss between the predicted and target structure is 0 and -1 otherwise. This is defined by Equation 3.

$$\mathcal{R} = \begin{cases} +1 & hamming(target,\ predicted) = 0 \\ -1 & otherwise \end{cases} \tag{3}$$

- **Transition Function ($\mathcal{P}$)**: The value of each valid action at time step t is predicted by the value network (f_θ) shown in Figure 2. It is stochastic (decayed ε-greedy; to encourage exploration) during training time and deterministic during evaluation time.

The state value is a sequence of the same length as the target structure. Initially, the state values are assigned characters similar to the target structure. At any given time step, a one-step look ahead is performed by the value network over all the possible actions in the set {GC, CG, AU, UA, GU, UG} for paired locations and {A, U, C, G} otherwise.

Given •(••)• as the target structure, the move locations is defined by the set {1, (2, 5), 3, 4, 6}. At the beginning, the state value is •(••)•. Assuming that in the next time step the move location is (2, 5) and the value network (Figure 2) returns GC as the best action then the state value becomes $\bullet G \bullet \bullet C \bullet$. Similarly, if the move location of the following time step is 6 and the value network returns A as the best action then the state value becomes $\bullet G \bullet \bullet CA$. The process is repeated until all the possible move locations have been assigned.

At each time step, the state value can be coded using the binary encodings shown in Table 1 and the resulting values concatenated to form the input feature with a single channel. In addition, the observed coded state values in each step of the episode are saved for training the value network.

Table 1: Coding Values.

state value	character	code
known unpaired	A	0000
	G	0001
	U	0001
	C	0011
known paired	GC	0100
	CG	0111
	AU	0101
	UA	1000
	UG	1001
	GU	0110
unknowns	unknown paired [(and)]	1010
	unknown unpaired [.]	1011

When an episode terminates, the designed sequence is folded using Zuker and Stiegler (1981) which was built as part of the ViennaRNA package (Lorenz et al., 2011) to obtain the predicted structure. If the Hamming loss between the predicted structure and the target structure is 0 then the observed coded state values are assigned target $+1$ and -1 otherwise.

The value network shown in Figure 2 is trained by minimizing the Mean Squared Error (MSE) loss between the predicted state value and the targets. This is capture by Equation 4 which shows the computation over the batches of size N.

The network design is inspired by AlexNet (Krizhevsky et al., 2012), however, it has important to note that it has additional layers; BatchNorm (Ioffe and Szegedy, 2015), Adaptive Pool, Drouput (Srivastava et al., 2014) and Softmax. The value network was trained at the end of one full pass over the target RNA batch sequences for 30 epochs with a batch size of 64 with Adam optimizer (α=0.0001). A summary of all the the differences has been captures in Table 2.

After multiple iterations, SPRNA eventually starts to learn the patterns over the state spaces which leads to correct sequence design. The playout and the state-evaluation algorithm are shown in Algorithm 2 and Algorithm 3 respectively.

$$\mathcal{L} = \sqrt{\left(\frac{1}{N}\right)\sum_{i=1}^{N}(f_\theta(s_t) - y)^2} \tag{4}$$

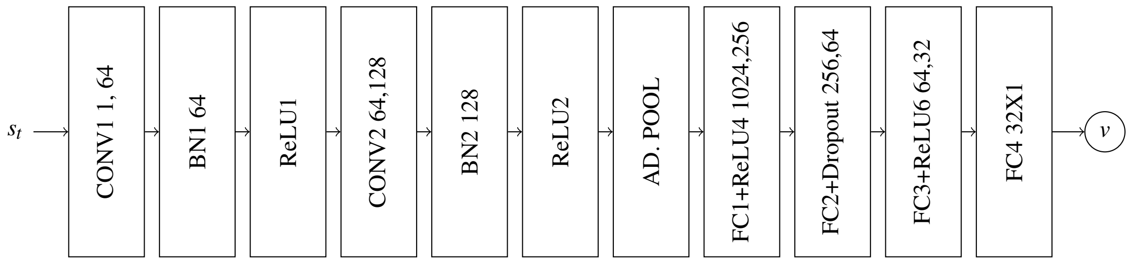

Figure 2: SPRNA value network. It accepts the coded state value (s_t) as the input and computes the value of that state. The network is composed of Convolutional Layers (CONV) (LeCun et al., 1998), Rectified Linear Units (ReLU), Batch Normalization (BN) (Ioffe and Szegedy, 2015), Drouput (with $p = 0.5$) (Srivastava et al., 2014), Adaptive Pooling (AD. Pool) and Fully Connected layers (FC). The *tanh* activation was used in the output layer. $CONV1$ is the the first Convolution while $BN2$ is the second Batch Normalization. Each layer(s) input/output value(s) have also been shown e.g. in $CONV1$ layer the numbers 1,64 corresponds to the input channels and output channels respectively while in layer $BN2$ the number 128 corresponds to the size of input volume. In FC layers, the numbers corresponds to the size of input and output neurons. Adaptive Pool allowed playing variable sized state episodes.

Table 2: A comparison between the value network (f_θ) and AlexNet. f_θ has additional layers; Batch Normalization and Adaptive Pool. The kernel size over the layers varies between the two architectures; 2, 4, 4 in the first, second and third convolutional layers respectively in f_θ and 11, 5, 3, 3, 3 in AlexNet.

Property	f_θ	AlexNet
Batch Norm.	3	0
Adaptive Pool	1	0
Output layer size	1	1000
Input channel	1	3
Convolution layers	3	5
Kernel size	2,4,4	11,5,3,3,3
Fully Connected layers	4	3
Dropout layers	1	2
Learnable parameter layers	10	8
Padding	0	1-2
Output activation	tanh	sotfmax

4.3 Dataset

Table 3 shows the information related to the dataset used in this research. The RNASP training environment was created by Dataset E while the evaluation environment was created using datasets A, B, C and D. The training and testing environments were created using Gym (Brockman et al., 2016) to ensure the extensibility of the RNA sequence problem to the already existing RL algorithms on the platform.

5 EXPERIMENTS

All the experiments we run on the NVIDIA Quadro T2000 GPU card running for a period of three days.

Algorithm 2: RNASP playout algorithm. The algorithm accepts list of RNA target structures ($\mathcal{D}$), paired bases ($\mathcal{P}$ = {GC, CG, AU, UA, GU, UG}) and unpaired bases ($\mathcal{F}$ = {A, U, C, G}). The call to to the value function invokes the value network to perform the current state evaluation. The state values are designed following the Gym (Brockman et al., 2016) Interface.

```
1  function RNA_playouts($\mathcal{D}, \mathcal{P}, \mathcal{F}$) :
2      $\mathcal{R} \leftarrow \emptyset$ ;
3      for $seq \in \mathcal{D}$ do
4          $state \leftarrow$ RNA_GymEnv($seq$) ;
5          $sites \leftarrow state$.available_sites( ) ;
6          $sites \leftarrow state$.shuffle_sites( ) ;
7          for each $s \in sites$ do
8              if pair($s$) then
                   $best \leftarrow$ value($state, \mathcal{P}$) ;
9              else $best \leftarrow$ value($state, \mathcal{F}$) ;
10             $state \leftarrow state$.apply_move($best$) ;
11         $\mathcal{R} \leftarrow \mathcal{R} \cup \{state\}$ ;
12     return $\mathcal{R}$ ;
```

Algorithm 3: The state evaluation function. The network accepts coded state s_t and returns the ε-greedy best action $a \in \mathcal{P}$ or $a \in \mathcal{F}$.

```
1  function value($state, \mathcal{A}$) :
2      $v \leftarrow \emptyset$ ;
3      for each $a \in \mathcal{A}$ do
4          $state \leftarrow state$.apply_move($a$) ;
5          $input \leftarrow state$.code( ) ;
6          $v(state, a) \leftarrow f_\theta(input)$ ;
7          $state \leftarrow state$.undo_move($a$) ;
8      return $\varepsilon greedy(v)$ ;
```

Table 3: Dataset Summary. Dataset E was used in training the model while A-D used in validation. The Length entry is average value calculated over all the sequences.

Dataset	Size	Source	Length
A	63	Kleinkauf et al. (2015b)	100.86
B	83	Kleinkauf et al. (2015b)	105.63
C	29	Taneda (2012)	191.87
D	100	Runge et al. (2018)	247.87
E	65K	Runge et al. (2018)	243.70

Table 4: Comparative Analysis on validation datasets.

Data	Model	MFE	GC	Correct
A	RNAInverse	-27.47	48.20	40/63
	incaRNAfbinv	-35.05	62.50	2/63
	Modena	-49.35	50.52	27/63
	RNAiFold	-53.70	49.76	48/63
	RNASP (Ours)	**-60.35**	54.62	**50/63**
B	RNAInverse	-29.02	48.23	47/83
	incaRNAfbinv	-34.47	46.67	3/83
	Modena	-49.90	51.00	37/83
	RNAiFold	-53.30	49.15	72/83
	RNASP (Ours)	**-62.19**	54.55	67/83
C	RNAInverse	-26.08	48.00	**20/29**
	incaRNAfbinv	0.00	0.00	0/29
	Modena	-59.64	52.09	11/29
	RNAiFold	-57.47	45.89	18/29
	RNASP(Ours)	-57.47	51.82	**20/29**
D	RNAInverse	-28.56	51.31	79/100
	incaRNAfbinv	-19.70	57.50	60/100
	Modena	-47.48	48.40	75/100
	RNAiFold	-52.10	53.76	80/100
	RNASP (Ours)	**-58.90**	**57.24**	**86/100**

6 RESULTS

The results are shown in Table 4. The GC and MFE entries are mean values while the Correct entry corresponds to instances where the Hamming loss between the predicted and target structure was 0. A comparative analysis of the mean running time (in seconds) was also carried out. The results are shown in Table 5, Table 6, Table 7 and Table 8.

In the time comparison Tables, given a target structure, each model was assigned a maximum of 60 seconds to design the RNA sequence. If the sequence could not be solved within this time constraint, its running time is marked by $-$.

Table 5: Time on A.

Model	Time
RNAInverse	0.17
incaRNAfbinv	-
Modena	9.01
RNAiFold	0.06
RNASP	1.01

Table 6: Time on B.

Model	Time
RNAInverse	0.16
incaRNAfbinv	-
Modena	9.52
RNAiFold	0.07
RNASP	1.00

Table 7: Time on C.

Model	Time
RNAInverse	2.12
incaRNAfbinv	-
Modena	14.99
RNAiFold	3.23
RNASP	1.42

Table 8: Time on D.

Model	Time
RNAInverse	3.5
incaRNAfbinv	-
Modena	41.78
RNAiFold	3.47
RNASP	1.73

7 DISCUSSION

While the existing RNA sequence models are heavily reinforced by user-defined constraints and expert knowledge such as the distribution of the GC content and other base pairings constraints, in this work we have presented a self-play RL agent which designs RNA sequences which can fold to match a given target structure.

A Deep Neural Network has been used to model the state-value function. By performing a one-step look-ahead over the value network given a valid set of actions {GC, CG, AU, UA, GU, UG} or {A, U, C, G} the agent can learn to design stable RNA sequence with desirable GC content value.

As shown in Table 4, RNASP has recorded competitive yet promising results. It recorded the best Hamming score on Datasets A and D while tying the score with RNAInverse on dataset C. While RNAiFold recorded the best result on dataset B (72/83), RNASP recorded the second-best result of 67/83. In addition, the RNASP model recorded the best MFE score on all the datasets (except dataset C) and registered a mean GC content within the generally recommended range between 50 and 60.

In relation to the mean time, all the models were assigned a maximum of 60 seconds to design a sequence conforming to the given target structure. The RNAInverse and RNAiFold models recorded the best times (in seconds) on datasets A and B. However it is important to note that RNASP recorded the best time on datasets C and D.

The average sequence length in datasets A and B is 100.86 and 105.63 respectively and 191.87 and

243.70 respectively in datasets C and D. We argue that the RNASP has a desirable mean runtime even when the sequence size increases. RNASP is ideal for designing longer sequences within a shorter period.

Modena recorded the longest amount of mean time while incaRNAfbinv could not solve any sequence within the 60 seconds constraint.

8 CONCLUSIONS AND FUTURE WORK

In this research, we have shown that Self-play can be used to model an agent which learns to design RNA sequences which fold to match a given target structure. By performing state evaluation using a Deep Neural Network, we have shown that RNASP can learn to design RNA sequences with desirable energy and GC content values.

The RNASP recorded the best score on two benchmark datasets and the best run time on longer sequences. As future research, it would be interesting to investigate if other encoding scheme or different value network architecture or novel learning methods would yield better results.

In addition, extension of Self-play to other real world problems such as drug design, genetics, protein folding and protein-protein interaction also remains an interesting future research endeavor.

9 SUPPLEMENTARY MATERIAL

The code and data used in the research is available at https://github.com/kobonyo/sprna

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of the French Embassy in Kenya and Strathmore University in Nairobi. The two entities facilitated a working environment which enabled the success of this research work.

REFERENCES

Andronescu, M., Fejes, A. P., Hutter, F., Hoos, H. H., and Condon, A. (2004). A new algorithm for rna secondary structure design. *Journal of molecular biology*, 336(3):607–624.

Bai, Y. and Jin, C. (2020). Provable self-play algorithms for competitive reinforcement learning. In *International conference on machine learning*, pages 551–560. PMLR.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint 1912.06680*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

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

Cazenave, T., Chen, Y.-C., Chen, G.-W., Chen, S.-Y., Chiu, X.-D., Dehos, J., Elsa, M., Gong, Q., Hu, H., Khalidov, V., et al. (2020). Polygames: Improved zero learning. *ICGA Journal*, pages 1–13.

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

Esmaili-Taheri, A. and Ganjtabesh, M. (2015). Erd: a fast and reliable tool for rna design including constraints. *BMC bioinformatics*, 16(1):1–11.

Garcia-Martin, J. A., Clote, P., and Dotu, I. (2013). Rnaifold: a constraint programming algorithm for rna inverse folding and molecular design. *Journal of bioinformatics and computational biology*, 11(02):1350001.

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

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

Kleinkauf, R., Houwaart, T., Backofen, R., and Mann, M. (2015a). antarna–multi-objective inverse folding of pseudoknot rna using ant-colony optimization. *BMC bioinformatics*, 16(1):1–7.

Kleinkauf, R., Mann, M., and Backofen, R. (2015b). antarna: ant colony-based rna sequence design. *Bioinformatics*, 31(19):3114–3121.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *IEEE*, 86(11):2278–2324.

Levin, A., Lis, M., Ponty, Y., O'donnell, C. W., Devadas, S., Berger, B., and Waldispühl, J. (2012). A global sampling approach to designing and reengineering rna secondary structures. *Nucleic acids research*, 40(20):10041–10052.

Lorenz, R., Bernhart, S. H., Höner zu Siederdissen, C., Tafer, H., Flamm, C., Stadler, P. F., and Hofacker, I. L. (2011). Viennarna package 2.0. *Algorithms for molecular biology*, 6(1):1–14.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint 1312.5602*.

Risso, D., Schwartz, K., Sherlock, G., and Dudoit, S. (2011). Gc-content normalization for rna-seq data. *BMC bioinformatics*, 12(1):1–17.

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

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint 1511.05952*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

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.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Taneda, A. (2012). Multi-objective genetic algorithm for pseudoknotted rna sequence design. *Frontiers in genetics*, 3:36.

Trotta, E. (2014). On the normalization of the minimum free energy of rna's by sequence length. *PloS one*, 9(11):e113380.

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

Zuker, M. and Stiegler, P. (1981). Optimal computer folding of large rna sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148.