SN Computer Science (2024) 5:403
https://doi.org/10.1007/s42979-024-02659-x

ORIGINAL RESEARCH

Self-Playing RNA Inverse Folding

Stephen Obonyo[1] · **Nicolas Jouandeau**[1] · **Dickson Owuor**[2]

Received: 5 April 2023 / Accepted: 26 January 2024

Abstract
Ribonucleic acid (RNA) sequence controls many biological functions in the body of living organisms including protein synthesis and gene regulations. The sequence folds according to the low Minimum Free Energy (MFE) which enables it to achieve various roles. Given a target RNA structure, an RNA sequence can be designed to fold accordingly in a process called RNA Inverse Folding. In this paper, we present an RNA Inverse Folding model that performs a single-step look-ahead to select optimal RNA base and base pairs to design RNA sequences. Our proposed model (SPRNA) learns to accurately design RNA sequences based on self-improving policy function. The model recorded state-of-the-art results on two test datasets and very competitive results on others.

Keywords RNA sequence design · RNA Inverse Folding · Reinforcement learning · Self-play · Machine learning · Deep learning · Self-improving algorithms

Introduction

An RNA molecule is fundamental in the body of living organisms. It controls key life-controlling biological processes such as the synthesis of proteins [3]. In the the *central dogma* of molecular biology [19], the RNA is the intermediate between the deoxyribonucleic Acid (DNA) and proteins. First, the DNA is transcribed to RNA which is then (ii) translated to proteins. Proteins are the fundamental building blocks of cells and tissues in the body of living organisms [19]. Proteins are also pivotal in catalyzing many biochemical processes in the body of living organisms enabling biological processes such as cell generation and metabolism [81]. Besides protein synthesis, RNA also plays a key role in the regulation and expression of genes that dictate the phenotypic and genotypic characteristics of living organisms [3].

RNA sequence is composed of four key Nitrogenous bases: (i) Adenine, (ii) Guanine (G), (iii) Cytosine and (iv) Uracil (U). A study by Schaffner [67] showed that there is a pairing between bases A and U, and G and C. These are often referred to as the *Watson-Crick pairs*. Some RNA also exhibits base pairing between A and G. In this paper, we will refer to AU, UA, GC, CG, GU and UG as base pairs and A, G, C and U as bases. The general base and base pair distributions are A:93%, G:5% C:1% and U:1% and GC/CG:60%, AU/UA:33% and GU/UG:7% respectively. The base pair distribution can vary in complicated RNA sequences e.g in left-most junctions can have GC/CG:82%, AU/UA:11%: and GU/UG:7% [57].

In Fig. 1, A is the target structure of the RNA sequence while B is a sample RNA sequence that conforms to the given target structure. RNA target structure defines the base and base pair positions in the RNA sequence. Given a target RNA structure, a sequence of bases and base pairs can be designed that fold to match it. This process is often referred to as RNA sequence design or RNA Inverse Folding. RNA sequence that does not fold according to the target structure can be dysfunctional and inactive [87].

This article is part of the topical collection "Advances on Computational Intelligence 2022" guest edited by Joaquim Filipe, Kevin Warwick, Janusz Kacprzyk, Thomas Bäck, Bas van Stein, Christian Wagner, Jonathan Garibaldi, H. K. Lam, Marie Cottrell and Faiyaz Docto.

✉ Stephen Obonyo
sobonyo@up8.edu

Nicolas Jouandeau
n@up8.edu

Dickson Owuor
dowuor@strathmore.edu

[1] LIASD, University of Paris 8, Saint-Denis, France

[2] SCES, Strathmore University, Nairobi, Kenya

Published online: 06 April 2024

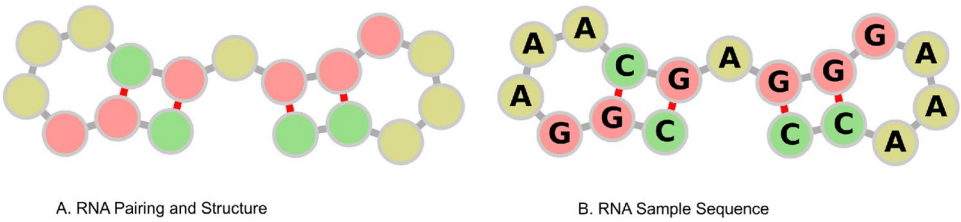


Fig. 1 **A** Target—((....)).((....)).—, **B** sample RNA—CGGAAACGAGGGAAACC

Before going deeper into the RNA Inverse Folding problem, we present some key concepts related to the problem.

(i) *Base* A base is a single nucleotide in the RNA sequence. It can be A, G, C or U. A sequence of bases composes an RNA sequence e.g. CGGAAA CGAGGGAAACC

(ii) *Base pair* In RNA sequence AU, UA, GC, CG, GU and UG bonds are referred to as base pairs. For instance in Fig. 1B, there are four base pairs: CG, GC, GC and CG in the sample sequence CGGAAA CGAGGGAAACC.

(iii) *RNA target structure* This structure defines the base and base pair positions in any RNA sequence. It is often represented in Dot Bracket notation e.g. in Fig. 1A—((....)).((....))—where the dots represent base positions while the matching brackets represent base pair positions. Defining RNA target structure using dots and matching brackets is a standard notation generally referred to as Dot Bracket notation. In this paper, all the target structures are represented in Dot Bracket notation.

(iv) *Fold structure* Given a sample RNA sequence such as CGGAAACGAGGGAAACC, it can be folded to generate a fold structure. There are no guarantees that the fold structure will match the target structure since replacing a single base or base pair can result in a different fold structure. Many algorithms that can be used to achieve this already exist e.g. Hofacker et al. [36], Zuker et al. [88].

(v) *Hamming distance* Hamming distance [32] is a mathematical formulae used to measure the distance between two strings of equal length. Given string s_1 and s_2 of equal length, the hamming distance is the number of positions where the two strings differ. In this context, we evaluate the success of the RNA Inverse Folding task by computing the hamming distance between the target and fold structure. If the hamming distance is 0 then the RNA Inverse Folding task is considered solved. Mathematically it is represented as follows:

$$\mathcal{H}(s_1, s_2) = \sum_{i=1}^{n} \phi(s_{1i}, s_{2i}) \tag{1}$$

where s_{1i} and s_{2i} are the ith positions in s_1 and s_2 respectively. ϕ is the function defined in Eq. (2).

$$\phi(s_{1i}, s_{2i}) = \begin{cases} 1 & \text{if } s_{1i} \neq s_{2i} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

(vi) *Folding accuracy* While the hamming distance computes the error value between the target and fold structure (the differences), the folding accuracy corresponds to the percentage of the matching positions between the two structures. It can be derived from the hamming distance as shown in Eq. (3).

$$\text{Accuracy } (s_1, s_2) = 1 - \frac{\mathcal{H}(s_1, s_2)}{n} \tag{3}$$

where n is the length of the two strings. Throughout this paper, we use a report the folding accuracy as a measure of the success of the RNA Inverse Folding task denoted as accuracy.

(vii) *Correct folding accuracy or expected folding accuracy* This is the folding accuracy which is equivalent to 1.0 e.g. the target and fold structure are identical.

(viii) *The GC content* The GC content is the percentage of G and C bases in the RNA sequence. For instance, in the sequence CGGAAACGAGGGAAACC, the GC content is $\frac{10}{17}$ or 58.82%. Both in vitro and in vivo processes are affected by the GC content as the quantity determines the stability of the RNA sequence which subsequently affects the functional effectiveness [17, 21, 78].

(ix) In vitro This is Latin for *in glass*. In this context, we use the term to refer to the experiments carried out in a computer or laboratory environment.

(x) In vivo This is Latin for *in life*. In this context, we use the term to refer to the experiments carried out in a living organism.

(xi) *The Minimum Free Energy (MFE)* This is the lowest conformational energy state of an RNA sequence. Methods for computing MFE already exist e.g Turn-

er's energy model [51], MultiRNAFold [31], Zucker's energy model [88] and Nussinov's energy model [55]. Similar to the GC content, the MFE also determines the sequence stability which is a key factor in the functional effectiveness of an RNA sequence. In this work, we compute the MFE using Zucker's MFE model which is accessible through the Vienna RNA software [48].

(xii) *Dynamic programming (DP)* This is a classical approach for solving optimization problems where a solution is obtained by breaking the problem into smaller sub-problems then combining the solutions to the sub-problems to obtain the solution to the main problem. DP methods have been used to solve RNA Inverse Folding [36] while its formulation also underlies the Bellman equations [9]. Reinforcement Learning (RL) solves sequential decision-making problems by learning the optimal state transition and value functions through trial and error. The Bellman equations are fundamental in deriving such functions [75]. When RNA Inverse Folding is formulated as a sequential decision-making problem, RL can be used to solve it [24, 65].

(xiii) *Design position* This is a position in the RNA sequence that is yet to be assigned a base or base pair in the intermediate RNA sequence s. If the position accepts a base pair according to the target structure, then it is a base pair position (accept GC, CG, AU, UA, GU or UG) otherwise, it is a base position (accept A, G, C or U). Base pair positions are assigned simultaneously. This is summarised as a function in (4).

$$\mathcal{A}(t,i) = \begin{cases} \{A,G,C,U\} & \text{if } t_i = . \\ \{GC,CG,AU,UA,GU,UG\} & \text{if } t_i = (\end{cases} \tag{4}$$

where t is the target structure, i is the design position and $\mathcal{A}$ is the set of possible actions.

(xiv) *Deep Neural Network (DNN)* Deep Neural Networks (DNN) are Artificial Intelligence (AI) methods can learn to approximate any function [37]. They are commonly used to find a mapping function from input space X to output space y (supervised learning) or approximate the distribution and patterns of the input space X (unsupervised learning). In design, DNNs are composed of artificial neurons where each neuron is a mathematical function that computes the weighted sum of the input data and then applies an activation function—this simulates the activation of biological neurons in the brain. In this paper, we used a DNN to evaluate RNA Inverse Folding states and select the optimal action (base or base pair) leading to the best cumulative hamming distance (reward). The network is then trained by automatically labeling the states with the reward at the end of a design task in a self-improving fashion (self-play).

(xv) *Value and policy functions* The value function is a function that evaluates the quality of a state according to a scalar variable. For instance, in RNA Inverse Folding, a value of 1.0 corresponds to the correct folding accuracy while a value of 0.0 otherwise. A policy (π) function on the other hand is a function that selects the optimal action (base or base pair) in a given state. The policy can be generated from the value function by selecting the action with the highest value in any given state e.g. $\pi(s) = \arg\max_a Q_\theta(s,a)$ where π is the policy, s is the state, a is the action and Q is the value function parameterized by DNN. For simpler problems, the Q function can be represented as a table where each row corresponds to a state and each column corresponds to an action.

While the primary goal in RNA Inverse Folding is to design an RNA sequence that folds according to the given target structure, it is also imperative to design a sequence containing desired GC content and Minimum Free Energy (MFE) values. The latter and former are key in RNA applications involving in vitro and in vivo experiments. Accordingly, as previously mentioned, they control the stability of the RNA sequence which translates to the effectiveness of the RNA.

We show the RNA Inverse Folding pipeline in Fig. 2, A. is the target structure (in Dot Bracket notation) B is

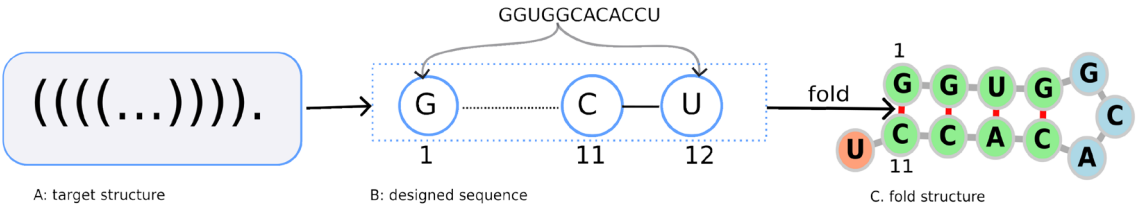


Fig. 2 RNA Inverse Folding pipeline. **A** The target structure, **B** the designed sequence and **C** the fold structure. The fold structure must conform to the target structure in any successful RNA inverse folding task. The paired and unpaired design positions in the target and fold match then the RNA Inverse Folding task is considered solved

the designed RNA sequence following base pair and base assignments while C is the fold structure obtained from folding the designed RNA sequence. According to the target structure, the position (1, 11) expects a base pair while 12 is a base. These have been assigned accordingly in the designed RNA sequence e.g. GC and U respectively. This procedure is repeated to obtain the designed RNA sequence. To validate if the fold structure matches the target structure, the paired and unpaired positions in the target structure are compared to those in the fold structure. If the hamming distance between the two is 0 then the RNA design task is considered solved. Given an RNA target structure, many possible RNA sequences can be designed to fold accordingly.

The success of RNA Inverse Folding can impact biological research and application including drug design, nanotechnology, precision medicine, genetic engineering, synthetic biology, bioengineering and material science [12, 16, 27]. These technologies are key in handling some global challenges such as the rapid development of vaccines and drugs, the development of technologies to address climate change and the growing population.

RNA Inverse Folding is one of the challenging tasks in structural biology. The design of simple RNA sequences can be solved using dynamic programming methods in $\mathcal{O}(n^3)$ time [55, 88]. In addition, some methods have reported deterministic runtime [11] while others polynomial [66]. The upper bound is NP-hard however [2, 38, 49]. NP-hard problems are those that cannot be solved in polynomial time complexity.

Several RNA Inverse Folding methods have been proposed in the past. These include dynamic programming methods [36], constraint programming [28, 29, 52], sampling methods [46, 59], evolutionary algorithms [26], genetic algorithms [77], Monte Carlo Tree Search [86], nested Monte Carlo Tree Search [57], ant-colony optimization [43, 44] and RL [24, 65].

While these methods have recorded very promising results, some of them are limited by their design choice. Dynamic programming methods are computationally expensive and can only be used to solve simple RNA Inverse Folding tasks since the whole search space is explored. Sampling-based methods can generalize well, however, they are limited by the exploration of the search space. While the evolutionary and genetic algorithms are suited to solving many optimization problems, they are limited by some design choices such as the fitness function and search space exploration leading to local optima convergence. MCTS-based methods are strong solvers. They can balance the exploration and exploitation of upper confidence bound (UCB) [45] is used as a tree policy. These methods, however, are computationally expensive and other variations such as nested MCTS [57] include expert heuristics which can be difficult to scale. RL-based methods are also strong solvers with the ability to balance exploitation and exploration. While they are prone to convergence issues, some novel design choices can be used to mitigate such issues.

In this paper, we present an RL-based RNA Inverse Folding model called Self-Playing RNA Inverse Folding (SPRNA). The model learns to design RNA sequences that fold according to the given target structure by performing a one-step look-ahead using DNN as a value function. The value function evaluates the base and base pairs at any given state and selects the optimal action (base or base pair) leading to the best cumulative reward (folding accuracy). The network is trained in a self-play fashion where the states that lead to the expected folding accuracy are automatically labeled with a positive reward and a negative otherwise. Deriving the policy from the value function is achieved by selecting the action with the highest value in the current state. Accordingly, SPRNA learns to design RNA sequences that fold according to the given target structure without any human feedback or hand-engineered folding rules.

We present our contributions as follows:

- *RNA Inverse Folding algorithm* We present an RNA Inverse Folding model that learns to improve by itself to design RNA sequences that accurately fold according to the given target. The model learns by self-play whereby the observations that lead to correct folding are automatically labeled with a positive reward and a negative otherwise. The algorithm is composed of two phases: (i) the sampling phase where the state observations are collected and (ii) the learning phase where the observations are used to train the value function to improve the policy.
- *Feature coding scheme* This is the method used to encode states during the RNA Inverse Folding process. In this work, we present a new feature coding scheme for the RNA Inverse Folding states. It is composed of a sequence of binary codes representing different states controlled by a feature parameter.
- *Local search improvement* We proposed a local search improvement method for the candidate solutions that do not have expected fold accuracy. Accordingly, the base pairs are broken or formed in the designed RNA sequence according to the target structure constraints. This idea is inspired by the existing methods, however, in our case it is only performed if the hamming distance between the target and fold structure is less than some threshold value δ.

While this paper builds on the work of Obonyo et al. [56] there are several key differences as outlined below:

(i) *The value network architecture* RNASP the value network was composed of two layers of CONV → BN → ReLU followed by an Adaptive Pooling layer,

an FC layer, a Dropout layer and two FC layers. In this paper, the network design follows CONV→NORM→RELU design with skip connection [33]. This design pipeline is repeated four times followed by one Adaptive Pooling and FC layer, one Dropout layer [73], and two FC layers. While RNASP was normalized using BN [39] the network normalization in this paper was achieved using Layer Normalization (NORM) [8]. In addition, the network in this paper is deeper than RNASP.

(ii) *Feature selection* The feature selection in RNASP was composed of binary codes that represent the known and unknown states. In this paper, a similar coding scheme was used, however, a feature control parameter is introduced to allow robust state feature representation.

(iii) *Test data set* RNASP was trained on 65K instances prepared according to Runge et al. [65]. It was then tested on datasets used by Runge et al. [65], Taneda [77] and Kleinkauf et al. [44] models. In this paper, the proposed model was trained and tested on similar datasets, however, during testing datasets A and B (test and train set from Kleinkauf et al. [44]) were combined into one test set. In addition, the proposed model is tested on a new test set to validate its performance. The dataset was prepared according to Anderson-Lee et al. [5].

(iv) *Comparative models* incaRNAfbinv [60] is one of the comparative models presented in Obonyo et al. [56]. The model did not record competitive results, thus, its results were not included in this paper. Three new comparative models, however, have been introduced in this paper including NEMO [57], MCTS-RNA [86] and LEARNA [65]. The first model is a nested Monte Carlo Tree (NMCTS) [13] search algorithm, the second is based on classical Monte Carlo Tree Search (MCTS) with Upper Confidence Boundary (UCB) [45] while the last is based on the Proximal Policy gradient algorithm [70].

The rest of this paper is organized as follows "Reinforcement Learning" covers Reinforcement Learning and related concepts key to this work, "Related Work" presents the related work, "Methods" introduce the methods used in this work including the algorithmic design, model testing and evaluation, "Experiments" includes the experiments carried out in this work as well as the results, "Discussion" discusses the obtained results while "Future Work" composes the future research perspectives and conclusion in "Conclusion".

Reinforcement Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that involves designing algorithms that learn from data and improve over time. There are two main types of ML: (i) supervised and (ii) unsupervised ML. In supervised ML, the algorithms learn from labeled data where the objective is to learn a mapping function f from the input to the output e.g. $y = f(X)$ where X is the input and y is the output. Importantly, each sample $s_i \in X$ is assumed to be independent and identically distributed (i.i.d) e.g. a given sample s_i is independent of the other samples $s_j \in X$ where $j \neq i$.

Several supervised ML algorithms exist such as Support Vector Machines (SVM) [18], Decision Trees [58], Random Forests [10], Neural Networks Rosenblatt [62] and Naive Bayes [61]. The algorithms can be applied to solve both the regression—where the output is a continuous value—and classification—where the output is a discrete value—problems. In unsupervised ML there are no labels or targets associated with each sample and the goal is to find underlying patterns in the data e.g. clusters which can be used to group similar samples. Unsupervised ML algorithms include K-means [50] and Principal Component Analysis (PCA) [85]. By extension, unsupervised ML algorithms can be used as a preprocessing step in supervised ML problems e.g. to reduce the dimensionality of or to identify and remove repetitive samples in the data.

Reinforcement Learning (RL) is a variation of supervised ML that involves learning by trial and error. In RL the labels are not explicitly provided, instead, the agent learns by interacting with the environment and collecting positive and negative rewards. The actions that lead to positive rewards are *reinforced* while those that lead to undesirable rewards are discouraged. RL agent can be formally represented as a Markov Decision Process composed of a tuple (S, A, P, R, γ). S is the state space, A is the set of actions to be taken at any given step, $P(s, a, s')$ is the transition function that encodes the probability of moving to the next state $s' \in S$ given the current state $s \in S$ and action $a \in A$, $R(s, a)$ is the reward function that defines the reward obtained by the agent upon taking an action $a \in A$ in a given state $s \in S$. γ is the control parameter referred to as the discount factor which determines the relative importance of future rewards. A policy π (s, a) defines how in any given state $s \in S$ the agent selects an action $a \in A$. The goal of an RL agent is to learn the optimal policy π^* that maximizes the expected cumulative reward. This can be formally expressed as a value function $V^{\pi}(s)$ under policy π as shown in Eq. (5).

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r(s_t, a_t) \mid s_0 = s_t\right] \tag{5}$$

where $\mathbb{E}_{\pi}$ is the expectation of the reward obtained by the agent over episodes (trajectories) following policy π and

$r(s_t, a_t)$ is the reward obtained by the agent at time t when it is in state s_t and takes action a_t.

The value function can also be represented as a state-action function $Q^{\pi}(s, a)$ where π is the policy, s is the state and a is the action as shown in Eq. (6).

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r(s_t, a_t) \mid s_0 = s_t, a_0 = a_t\right] \tag{6}$$

The optimal policy π^* is the policy that maximizes the value function $V^{\pi}(s)$ or the state-action function $Q^{\pi}(s, a)$ as shown in Eqs. (7) and (8) respectively.

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \tag{7}$$

$$\pi^* = \arg\max_{\pi} Q^{\pi}(s, a) \tag{8}$$

In RL the discount factor determines how much the current reward is weighted toward future rewards. The inclusion of the discount factor in the RL target underscores the difference between RL and classical supervised ML. In addition to this, samples in RL are not i.i.d. e.g. the current state at time t is dependent on the previous state at time $t - 1$.

There are four variations of RL algorithms (i) value-based, (ii) policy gradient, (iii) actor-critic and (iv) model-based. In value-based RL, the agent learns the state value or action-value function and derives the policy from the value function e.g. Q-learning [82] and SARSA [64]. In policy gradient RL, the agent learns the policy directly (without calculating the state or action value) according to the RL objective functions in Eqs. (5) or (6) e.g. REINFORCE [84]. Policy gradient methods are prone to high variance which makes learning and generalization hard. To address this problem, actor-critic (AC) method proposed a learning objective that includes both the policy and value (or advantage) functions. In AC, the actor selects the action while the critic evaluates the quality of the action relative to others—the advantage function. The advantage function encourages the selection of actions that are better than the average action in the current state while also controlling the variance of the reward estimates. Common actor-critic methods include A2C and A3C [54], Proximal Policy Optimization (PPO) [70], Deep Deterministic Policy Gradient (DDPG) [47] and Trust Region Policy Optimization (TRPO) [69]. In model-based the RL agent learns the model of the environment which then informs the policy [15, 20, 74].

While in simpler problems policy functions can be represented as a table, in more complex problems where there is a large state space e.g. in games such as Go and Chess and computational biology problems such as RNA Inverse Folding and assembly of DNA sequences, the value functions are parameterized as supervised ML models such as neural networks [6, 53]. Using DNN as a policy is generally referred to as Deep Reinforcement Learning (DRL) in the RL literature. DRL has been successful in solving complex problems such as Go [71], Chess [72], StarCraft II [79], Atari games [53] and computational biology problems such as RNA Inverse Folding [24, 65]. Owing to the ability of DNN to approximate represent any function [37] they are widely used in DRL.

While RL algorithms alleviate the need for labeling data—a constraint that can be expensive and time-consuming—they are prone to convergence issues. They require long hours of training which can be computationally expensive. In addition, they are sensitive to the choice of hyperparameters e.g. learning rate, discount factor, optimization algorithm, and network architecture.

Related Work

Several RNA Inverse Folding models have been proposed in the past decades. The models include concepts and ideas borrowed from dynamic programming (DP), constraint programming (CP), and Reinforcement Learning (RL): value-based and policy gradient methods, graphical models, probabilistic models, stochastic optimization, physics-based and energy optimization models, genetic and evolutionary algorithms, Monte Carlo Tree Search (MCTS), and a combination of MCTS and machine learning (ML).

Eastman et al. [24] proposed an RL-based model to solve the RNA Inverse Folding problem. The authors used DNN with residual connections as value functions.

Their method recorded competitive results with limited comparison to other methods and datasets. In, Runge et al. [65] the authors proposed a proximal policy gradient algorithm that learns the policy based on neural network gradient optimization. The output of the network is a probability distribution over the action base or base pairs (4) with states encoded using one-hot encoding. The model was trained and tested on the 65K instances obtained from RFAM [30]. On average it recorded over 0.9 accuracy on the test set, however, the same was not recorded on a different test set such as Anderson-Lee et al. [5] and Kleinkauf et al. [43, 44].

MCTS is *any-time* optimization algorithm that has been applied to solve complex problems in games and planning problems [76]. MCTS has also been applied to RNA Inverse Folding problem recording very competitive and strong results compared to several baseline models. The MCTS algorithm is composed of four phases: (i) selection, (ii)

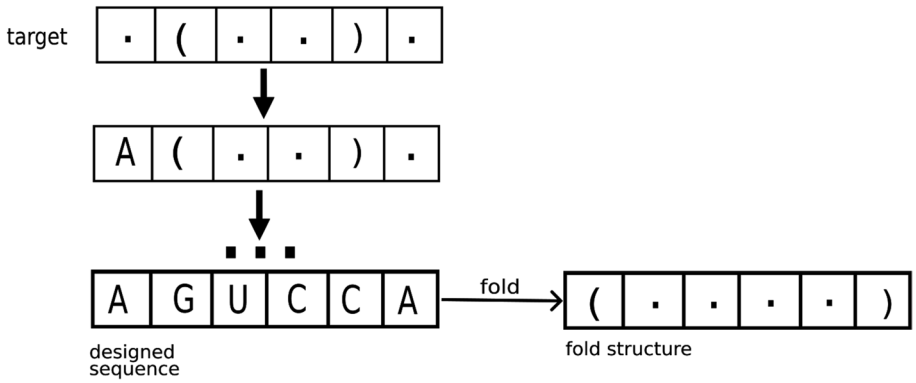


Fig. 3 SPRNA episode over a sample target structure • (• •)•, each state value can be known unpaired, known paired, unknown paired or unknown unpaired. The designed sequence is folded to generate the fold structure which is then compared to the target

expansion, (iii) simulation and (iv) backpropagation. In the selection phase, the node in the tree with the highest UCB value is selected. In the expansion phase, the selected node is expanded by adding a new node to the tree. In the simulation phase, valid actions are selected randomly and applied until the terminal state is reached. In the backpropagation phase, the reward obtained in the simulation phase is backpropagated to update the Q values of the nodes in the tree. In RNA Inverse Folding, the reward is the folding accuracy or hamming distance while every node represents a valid base or base pair assigned to a given design position.

Recently, MCTS-based algorithms have become more attractive due to three key reasons: (i) the ability to effectively sample large search spaces, and (ii) MCTS is an *anytime* algorithm—MCTS can be terminated at any time while collecting the existing results, (iii) convenient combination with ML—since Silver et al. [72] the simulation phase of the MCTS increasingly being replaced with DNN prediction.

Several MCTS algorithms have been proposed for solving RNA Inverse Folding. The MCTS-based model proposed by Yang et al. [86] selects the best action according to the UCB [45] formulae while balancing the exploration and exploitation using a set exploration-exploitation parameter. If the fold structure does not match the target structure then a local search is performed to break incorrect base pairs and form the missing ones according to the target structure constraints. Portela [57] also proposed a nested MCTS solver which is guided based on strong hand-engineered rules in the loops, left and right junctions. Generally, in nested MCTS the solutions in the lower recursive level inform upper level(s) [63]. Nested MCTS methods have been more effective in solving some hard optimization problems in games and computational biology [13, 63] compared to classical MCTS at a computational cost. In another work, Cazenave and Fournier [14] presented an MCTS-based solver composed of DNN policy evaluation and general rollout adaptation e.g. combination of MCTS with DNN. MCTS solvers are strong, however, are computationally expensive and can be sensitive to the initial parameter and hyperparameter values of the policy networks.

Garcia-Martin et al. [28] proposed constraint programming model (CP). The constraints were defined according to the expected MFE values, GC content and base pair distributions. A model with a similar algorithmic design was also proposed by Garcia-Martin et al. [29] which could solve complex RNA Inverse folding tasks. CP-based algorithms have strong theoretical foundations and can be used to solve complex problems, however, in the context of RNA Inverse Folding, they are limited by the set of constraints that must be specified prior to the design process. Such specifications can be difficult in a large-scale context or complex to adapt to new problems. In another study, Minuesa et al. [52] presented a CP model that included stronger heuristics and restart strategies. These restart strategies could guide the search to the global optimum. According to the authors, the model could design RNA sequences that are usable in in vitro as well as in vivo. Despite this argument, this method also inherits the limitations of CP-based methods previously mentioned.

Besides the above-mentioned RNA Inverse Folding methods, other researchers have also proposed different design approaches including weighted sampling approaches [59], genetic algorithms [4, 22, 41, 77], adaptive sampling with optional search improvement [46, 59], dynamic programming [36], evolutionary algorithms with heuristics [25, 26], ant colony optimization [43, 44] and graphical and shape-aware solvers [7, 23, 41, 83].

Overall, the existing RNA Inverse Folding methods have recorded very promising results. However, in the design of a new method, it is imperative to design algorithms that are not only accurate and generate RNA sequences with expected MFE and GC content but also loosely dependent on human supervision (e.g. can automatically self-improve).

The latter is important in the design of RNA Inverse Folding methods as it underlies the scale and cost of the RNA Inverse Folding.

Methods

RNA Inverse Folding as a Markov Decision Process

RNA Inverse Folding can be formulated as a Markov Decision Process (MDP) according to $\{\mathcal{A}, \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma\}$. $\mathcal{A}$ is the set of actions that can be taken in any state—$\{A, C, U, G, \}$ for base design positions and $\{AU, UA, GC, CG, GU, UG\}$ for base pairs, $\mathcal{S}$ is the state composed of a binary sequence of length N where the known and unknown states are coded according to Table 1 in Obonyo et al. [56]. In this paper, a state is known if the design position is assigned a base or base pair and unknown otherwise. $\mathcal{P}$ is a deterministic transition function that models state change from s_t to s_{t+1} with a probability of 1. $\mathcal{R}$ is the reward which is 0 in all states except the terminal state where it is $+1$ if the hamming distance between the target and fold structure is 0 and -1 otherwise. The discount factor γ is set to 1 so that the agent only the value of the immediate reward.

The SPRNA Algorithmic Design

The SPRNA requires no expert knowledge to learn except the rules of RNA Inverse Folding. Given a set of valid actions and the coded state s_t, a one-step look-ahead state evaluation is performed using the value network, v_θ, to select the optimal action from a set of valid actions $\mathcal{A}_t = \{A, C, U, G, \}$ for base design positions and $\{AU, UA, GC, CG, GU, UG\}$ for base pairs. The value network v_θ returns the the value associated with each valid action. The optimal action is selected according to the epsilon-greedy (ϵgreedy) policy to allow for the balancing of exploration and exploitation e.g. with probability ϵ the action is selected randomly and with probability $1-\epsilon$ the action with the the highest value is selected. The pseudocode of the SPRNA value algorithm is presented in Algorithm 1.

Algorithm 1 Value algorithm

Input: s_t, $\mathcal{A}_t$, ϵ, v_θ, δ
Output: a_t
$V \leftarrow v_\theta(s_t, \mathcal{A}_t)$ $\{\mathcal{A}_t$ is bases or base pairs$\}$
$a_i \leftarrow \epsilon\text{greedy}(\epsilon, V)$
$a_t \leftarrow \mathcal{A}_t[a_i]$
return a_t

Algorithm 2 ϵgreedy function

Input: ϵ, V $\{V$ is the value of each action$\}$
Output: a_t
random $\leftarrow$ random$(0, 1)$
if random $< \epsilon$ **then**
  $a_t \leftarrow$ random$(0, \text{length}(V))$ {Exploration: select a random action}
else
  $a_t \leftarrow \arg\max_{a \in \mathcal{A}_t} V$ {Exploitation: select the best action}
end if
return a_t

Algorithm 3 The self-play algorithm. Execute a single episode

Input: v_θ, $\mathcal{B}$, target, ϵ, δ
Output: $\mathcal{B}$
seq $\leftarrow$ target {Initialize the fold as the target}
for all $i \in \{1, \ldots, length(\text{target})\}$ **do**
  $A_t \leftarrow getActions(\text{target[i]})$
  $a_t \leftarrow value(s_t, A_t, \epsilon, v_\theta)$
  seq[i] $\leftarrow applyAction(a_t)$
  $s_t \leftarrow encodeState(\text{seq})$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{s_t\}$ {Save the state in the replay buffer}
end for
seq $\leftarrow LS(\text{seq}, \text{target}, \delta)$ {Local search. Details in Section 4.4}
foldstruct $\leftarrow Fold(\text{seq})$ {Generate the fold structure}
$r \leftarrow \mathcal{H}(\text{target}, \text{foldstruct})$ {$\mathcal{H}$ is the hamming distance function (Equation 3)}
if $r = 0$ **then**
  $r \leftarrow 1$
else
  $r \leftarrow -1$
end if
for all $i \leftarrow 1$ *to* $length(\mathcal{B})$ **do**
  $\mathcal{B}[i] \leftarrow \mathcal{B}[i] \cup \{r\}$ {Label each state with the reward}
end for
return $\mathcal{B}$

Initially, ϵ is set to 0.1 and then decays exponentially as a function of SPRNA training epochs. The self-play algorithm accepts the target and value network as input and runs a single episode over the paired and unpaired design positions. Every design position is encoded, and then optimal action is obtained by calling the value Algorithm 1. The action is then applied to the current state to obtain the next state which is saved in the replay buffer $\mathcal{B}$. At the end of the episode, each state in the replay buffer is labeled with a + 1 reward if the hamming distance between the target and fold structure is 0 and − 1 otherwise.

As an example, a sample SPRNA episode is shown in Fig. 3. Initially, the fold sequence is initialized as the target sequence. In every step of the episode, the state is encoded and passed to the value network to obtain the optimal action (base or base pair) which is then applied to the current state to obtain the next state.

Feature Selection

The self-play Algorithm 3 returns a replay buffer $\mathcal{B}$ composed of states and their corresponding labels. The states are the features while the labels are the targets. These are used to train the value network v_θ by minimizing the mean squared error (MSE) (shown in Eq.) loss function.

$$\mathcal{L}(r, s) = \frac{1}{N} \sum_{i=1}^{N} (r_i - v_\theta(s_i))^2 \tag{9}$$

The replay buffer size is limited to 20K as such the oldest samples are removed from the buffer. The generation and update of the training samples and training of the value network are automated e.g. there is no need for human supervision. AI for games literature refers to this as self-play [72] where the agent learns to improve by itself by collecting observations and improving the policy function. Self-play

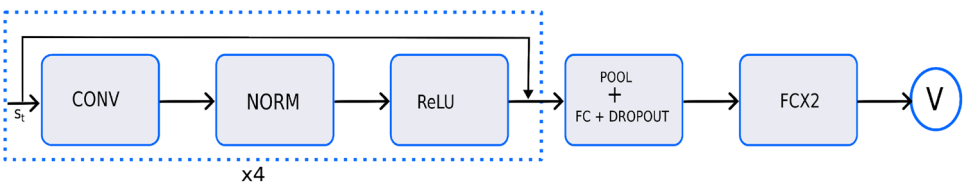


Fig. 4 SPRNA value network. *CONV* convolutional layer, *NORM* layer normalization, *ReLU* rectified linear units, *FC* fully connected layer, *POOL* adaptive pooling layer. s_t is the is the encoded state and the network outputs a scalar reward value

has been successful in solving complex problems such as Chess [72] and GO [71].

State representation (line 8 in Algorithm 3) is a key component of any RL algorithm. In this paper, we adopt the state representation proposed in Obonyo et al. [56] which is composed of encoding states as a sequence of binary codes for the known and unknown states. Bases {A, G, U, C} are encoded using 0000, 0001, 0010, 0100, and 1000 binary codes respectively while base pairs {GC, CG, AU, UA, UG, GU} are encoded using 0010, 0111, 0101,1000,1001, 0110 binary codes respectively. We refer to the former as known unpaired and the latter as known paired. The matching parenthesis of Dot Bracket (unknown paired) is encoded using 1010 and the dots (unknown unpaired) using 1011. We experimented with one-hot encoding, however, it performed worse than our proposed encoding scheme.

As an example, given a target structure •(• •)•, the move positions are defined by {1, (2, 5), 3, 4, 6}. Given that the move position is 1 and the value function returns A as the optimal action, state s_{t+1} becomes A(• •)•. Similarly, after a one-time step, the state s_{t+2} becomes AG• •C• with (2, 5) as the move position and GC as the optimal action. Importantly, the base pairs are assigned simultaneously and the move positions are indexed in the intermediate sequence (seq) and then encoded according to the procedure discussed in the preceding paragraph before being passed to the value function. This process is repeated until all the base and base pair positions are assigned.

In this paper, we also introduce a feature control parameter $\mathcal{W}$ to allow for richer state feature representation by determining the overlap between the move positions. For example, based on the •(• •)• example, if move locations are defined by the set {(1), (2, 5), (4), (3), (6)} then $\mathcal{W}$ is 1. If $\mathcal{W} = 2$, then the set defining move locations becomes { (1, (2, 5)), ((2, 5), 4), (4, 3), (3, 6)} and {(1, (2, 5), 4), ((2, 5), 4, 3), (4, 3, 6)} if $\mathcal{W} = 3$. Thus in the latter and former move position instances, there is an overlap between the design positions. This permeates robust state feature representation as our experimentation with one-hot encoding performed worse. In the experiments, $\mathcal{W}$ was varied between 1 to 6. In Obonyo et al. [56], the value was set to 1 throughout the experiments. its value was set to 1 throughout the experiments.

Local Search (LS) Improvement

If the hamming distance between the target and fold structure is greater than 0, and the distance is below or equal to the threshold δ, then a local search procedure is performed to improve the solution. This procedure is key in instances where the target structure can be obtained by performing a few mutations on the designed RNA sequence. In the experiments, δ was set to 0.1. The local search procedure is shown in Algorithm 4. The procedure is similar to the one proposed in Yang et al. [86], however, in this paper, it is constrained by the hamming distance threshold δ.

Algorithm 4 LS function

1: **Input:** seq, target, δ
2: **Output:** seq
3: foldstruct $\leftarrow Fold$(seq)
4: hamming_distance $\leftarrow \mathcal{H}$(target, foldstruct)
5: **if** hamming_distance $\leq \delta$ and hamming_distance > 0 **then**
6: seq $\leftarrow random\_mutation$(seq)
7: **end if**
8: return seq

Table 1 Test dataset statistics

Name	Total sequences	Source	Train	Average length
A & B	146	Kleinkauf et al. [44]	X	103.24
C	29	Taneda [77]	X	191.87
D	100	Runge et al. [65]	X	247.87
E	65K	Runge et al. [65]	✓	243.70
F	100	Anderson-Lee et al. [5]	X	159.01

Table 2 SPRNA results on A & B

$\mathcal{W}$	Correct/146	MFE	GC
1	100	− 61.20	49.21
2	105	− 62.89	52.16
3	110	− 62.76	53.02
4	**123**	− 62.19	56.55
5	114	− 64.63	52.91
6	113	− 61.10	51.85

The bold shows the best score entry

Table 3 SPRNA results on C

$\mathcal{W}$	Correct/29	MFE	GC
1	18	− 58.22	54.16
2	17	− 62.18	51.62
3	19	− 60.57	53.14
4	**24**	− 63.35	55.62
5	20	− 49.83	46.45
6	21	− 60.09	53.36

The bold shows the best score entry

Table 4 SPRNA results on D

$\mathcal{W}$	Correct/100	MFE	GC
1	80	− 40.21	58.08
2	85	− 47.38	56.74
3	90	− 58.90	57.26
4	**92**	− 59.92	51.64
5	88	− 60.28	58.23
6	89	− 52.45	57.68

The bold shows the best score entry

Table 5 SPRNA results on F

$\mathcal{W}$	Correct/100	MFE	GC
1	50	− 58.96	49.15
2	52	− 61.76	49.15
3	56	− 61.88	47.64
4	**69**	− 60.47	51.82
5	59	− 55.62	50.33
6	58	− 54.59	59.09

The bold shows the best score entry

The Value Network

The value network design is shown in Fig. 4. The coded state (s_t) is fed into the network to generate a scalar reward signal. The network design is composed of Convolutional, Normalization, Rectified Linear Units, Fully connected and Adaptive Pooling layers. The network architecture also includes skip (residual) connections which allow for training deeper network models with no performance degradation [33] as training DNN with many layers generally leads to worse performance due to the vanishing gradient problem [34]. The Layer Norm (LN) [8] enables the normalization of activations across the input features dimension enabling faster training and reducing the model overfitting. This is as opposed to Batch Norm (BN) [39] used in Obonyo et al. [56] which normalizes the activations across the batch dimension. There are three key differences between this value network and Obonyo et al. [56]. The network is training based on the sample output from the self-play Algorithm 3. This is achieved by running the self-play algorithm over several training target structures, updating the training samples then training the value network. This procedure is concretely shown in Algorithm 5. The full network training such as details such as the parameter and hyperparameter configurations is shown in Appendix 1.

Table 6 Folding accuracy comparative analysis. SPRNA entry is based on $\mathcal{W} = 4$

Data	Inv.	NEMO	MCTS	iFold	MODENA	LEARNA	SPRNA
A & B [/146]	87	100	102	100	85	105	**110**
C [/29]	20	**24**	23	18	11	21	**24**
D [/100]	79	85	91	80	75	88	**92**
F [/100]	55	**95**	70	71	52	68	69

The bold shows the best score entry

Table 7 GC content comparative analysis

Data	Inv.	NEMO	MCTS	iFold	MODENA	LEARNA	SPRNA
A & B [/146]	48.21	50.23	51.22	49.45	50.76	49.82	53.58
C [/29]	48.00	52.12	50.33	45.89	52.09	50.01	52.83
D [/100]	51.31	52.85	51.03	53.76	48.40	51.23	56.34
F [/100]	52.05	53.01	54.50	49.94	50.12	53.20	54.26
Average	49.89	52.05	51.77	49.76	50.34	51.10	54.25

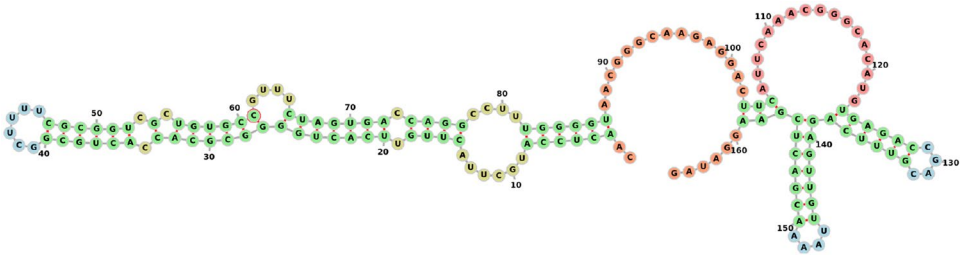
Table 8 MFE comparative analysis

Data	Inv.	NEMO	MCTS	iFold	MODENA	LEARNA	SPRNA
A & B [/146]	− 28.23	− 30.35	− 35.45	− 53.50	− 49.62	− 31.05	− 50.01
C [/29]	− 26.08	− 29.98	− 40.23	− 57.47	− 59.64	− 37.20	− 56.44
D [/100]	− 28.15	− 31.01	− 45.23	− 51.10	− 47.48	− 47.34	− 57.34
F [/100]	− 25.33	− 33.22	− 42.55	− 50.03	− 54.33	− 51.12	− 48.33
Average	− 26.95	− 31.14	− 40.87	− 53.09	− 52.77	− 41.68	− 53.03

Table 9 Average time taken

Data	Inv.	NEMO	MCTS	iFold	MODENA	LEARNA	SPRNA
A & B [/146]	1.17	2.17	5.17	1.07	9.25	8.23	3.09
C [/29]	2.12	2.12	6.98	2.50	14.99	17.70	3.30
D [/100]	1.16	3.50	4.02	3.53	41.78	10.01	2.58
F [/100]	10.22	5.09	8.98	10.01	50.23	25.00	4.10
Average	3.67	3.22	6.29	4.28	29.06	15.24	3.26

Fig. 5 Sample correct fold from A & B



Algorithm 5 SPRNA training algorithm

initialize network v_θ, expoloration parameter ϵ, batch size b, epochs, δ
targets $\leftarrow$ getTargets()
train_samples $\leftarrow \emptyset$
for epoch $\leftarrow$ 1 to epochs **do**
  for all $i \in \{1, \ldots, length(\text{targets})\}$ **do**
    $\mathcal{B} \leftarrow$ self-play(v_θ, targets[i], ϵ, δ)
    train_samples $\leftarrow$ train_samples $\cup\, \mathcal{B}$
    if $i \mod \text{b} = 0$ **then**
      samples $\leftarrow$ sample(targets, b)
      $v_\theta \leftarrow$ train(v_θ, samples)
    end if
  end for
end for

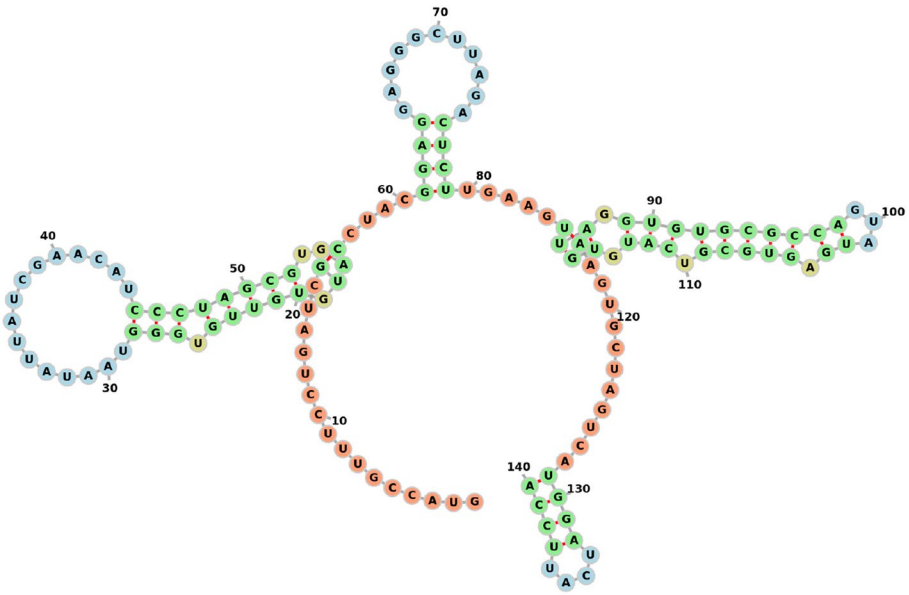


Fig. 6 Sample correct fold from C

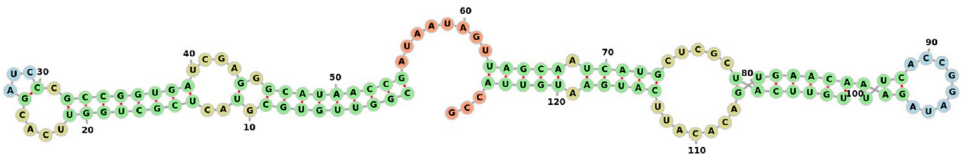


Fig. 7 Sample correct fold from D

Some key differences between the value network in SPRNA and Obonyo et al. [56] are (i) the SPRNA uses Layer Norm (LN) [8] while Obonyo et al. [56] uses Batch Norm (BN) [39], (ii) the SPRNA uses skip connections [33] while Obonyo et al. [56] does not, and (iii) the SPRNA has 16 layers while Obonyo et al. [56] has 11 layers.

Datasets

SPRNA was trained on a dataset containing 65K RNA target sequences obtained from RFAM [30] and compiled by Runge et al. [65]. Following the training, it was tested on four benchmark datasets. Contrary to Obonyo et al. [56], in this paper, the training and testing samples from Kleinkauf et al. [44] are combined leading to a total of 146 sample samples. The RFAM dataset is generally used to benchmark RNA Inverse Folding methods and it is one of the largest datasets and commonly used by several methods [65, 77]. Also, the SPRNA was tested on a new benchmark test dataset prepared according to Anderson-Lee et al. [5]. It contains a total of 100 sequences including simple as well as complex samples. This dataset was prepared to allow testing RNA Inverse Folding methods on new complex target structures. A summary of all the training and testing datasets is presented in Table 1. The data name entry corresponds to the name of the authors.

Experiments

After training, the SPRNA model was tested and its performance was compared to some existing baselines. The baselines were selected based on two key criteria: (i) the methodology used to solve the RNA Inverse Folding problem: e.g. tree search, RL, etc., and (ii) their performance on the benchmark datasets: the existing strong baselines were selected. The baselines included (i) MODENA [77]: evolutionary algorithm, (ii) NEMO [57]: nested MCTS, (iii) RNAiFold [28]: constraint programming, (iv) RNAInverse [35]: dynamic programming, (v) LEARNA [65]: policy gradient and (vi) MCTS-RNA [86]: MCTS. The abbreviations are as follows: RNAiFold is abbreviated as iFold, RNAInverse: Inv., and MCTS-RNA: MCTS. The MFE values were computed using the Vienna software [48]. The SPRNA results with different $\mathcal{W}$ values are presented in Tables 2, 3, 4 and 5. The results of the best SPRNA model according to $\mathcal{W}$ compared some existing baselines are presented in Tables 6, 7, 8 and 9.

Discussion

In this section, we discuss the SPRNA results according to the control parameter $\mathcal{W}$, folding accuracy, MFE and GC and the time taken during the design process. We also discuss some complexity associated with the test datasets and the SPRNA model.

On the control parameter $\mathcal{W}$. The $\mathcal{W}$ the parameter controls the feature representation of the state (discussed in methods "Feature Selection"). The value was varied between 1 to 6 with $\mathcal{W} = 4$ recording folding accuracy of 123/146 (84.24%), 24/29 (82.75%), 92/100 (92%) and 69/100 (69%) on test datasets A &B, C, D and F, respectively. We compared these results in a standard encoding scheme such as one-hot encoding and obtained folding accuracy of 90/146 (61.64%), 19/29 (65.51%), 85/100 (85%) and 59/100 (59%) on test datasets A &B, C, D and F respectively. This corresponds to a 22.6, 17.24, 7 and 10% improvement on test datasets A &B, C, D and F, respectively. This also shows that our feature representation was richer than the one-hot encoding.

On folding accuracy We benchmarked SPRNA ($\mathcal{W} = 4$) with existing baselines on test datasets A &B, C, D and F. SPRNA recorded the best result on two test datasets; A &B and D, while recording a similar score to NEMO on test dataset C. NEMO also recorded the best result on the test dataset C. NEMO [57] is a nested MCTS algorithm; MCTS the algorithm where lower-level results are used to bootstrap the upper-level results [13]. NEMO is a strong algorithm, however, it includes complex hand-engineered heuristics such as different distributions of base and base pairs in junctions and loops. While such constraints are key in RNA Inverse Folding, they do not scale well in large-scale RNA engineering. In contrast, SPRNA does not have such constraints: it automatically learns the correct base and base pair distributions without any supervision or rules. This is achieved by training the policy function through self-play (self-improvement). While LEARNA can also be considered a self-improving algorithm (policy gradient), it is outperformed by our model on all the test datasets. Moreover, its design objective, learning and state representation are different from SPRNA. The LEARNA recorded a folding accuracy of 105/146 (71.92%), 21/29 (72.41%), 88/100 (88%) and 68/100 (68%) on test datasets A &B, C, D and F respectively. Accordingly, this corresponds to a margin of 12.32, 9.83, 4 and 1% when compared to SPRNA($\mathcal{W} = 4$) on test datasets A &B, C, D and F, respectively. Furthermore, SPRNA also outperformed MODENA, RNAiFold, RNAInverse and MCTS-RNA on test datasets A &B, C, D and F. MCTS-RNA (abbreviated as MCTS in the results table) is an MCTS-based algorithm that selects an optimal action by performing several simulations and then backpropagating the reward signal at the end of the simulation to reinforce the action selection process. This is achieved by incrementally building the tree, and collecting statistics such as the number of node visits and average Q-values. While MCTS-RNA was run for 500 simulations, we were able to get better results with 1 simulation using SPRNA e.g. in every step of the episode a one-step look-ahead is performed until the terminal state when all the design positions are filled (single simulation). Besides the performance gains, SPRNA was able to design RNA sequences that other baselines failed to design e.g. Figure 5 (from test dataset A & B), Fig. 6 (test dataset C), and Fig. 7 (test dataset D), are such examples. SPRNA learned to correctly balance the base and base pair distribution in left and right junctions and the loops.

On MFE and GC content In RNA Inverse Folding is important for designing a model which not only accurate but also designs sequences with desired MFE and GC content values. The latter and the former properties determine the functional effectiveness of the RNA sequences in both in vitro and in vivo [40, 80]. While there is no conclusive argument on the optimal MFE and GC content values, sequences with higher GC content and lower MFE values are generally more stable [17, 78]. According to Table 7 SPRNA ($\mathcal{W} = 4$) recorded the highest mean GC content value of 54.25% compared to other baselines; RNAInverse: 49.89%, NEMO: 52.05%, MCTS: 51.77%, iFold: 49.76%, MODENA: 50.34% and LEARNA: 51.10%. This corresponds to a margin of 4.36, 2.2, 2.48, 4.49, 3.91 and 3.15% respectively. SPRNA and NEMO recorded the lowest mean MFE values of − 53.03 and − 52.05 respectively which can be translated to higher stability.

Table 10 Parameter/hyperparameter for the value network

Parameter/hyperparameter/setting	Value
Dropout	0.4
Learning rate	1e−05
Optimizer	Adam
First CONV features	64
Second CONV features	64
Kernel size	4
Adaptive pooling size	2
Batch size	32
Learning rate decay	exponential
Epochs	100

Table 11 SPRNA with one-hot encoding

Data	Solved	MFE	GC content
A & B [/146]	90	− 69.20	51.34
C [/29]	19	− 40.40	55.10
D [/100]	85	− 43.31	48.22
F [/100]	59	− 38.12	53.45

On the inference time Time complexity is a key factor in any computational problem. We benchmarked how long the SPRNA ($\mathcal{W} = 4$) takes to design RNA sequences on average. The average time is reported in seconds and only includes the inference time. The average SPRNA inference time across the test dataset was 3.26 s while the baselines 3.67, 3.22, 6.29, 4.28, 29.06 and 15.24 s for RNAInverse, NEMO, MCTS, iFold, MODENA and LEARNA respectively. This corresponds to a margin of 0.41, 0.04, 3.03, 1.02, 25.8 and 11.98 s respectively. SPRNA recorded the second-lowest average inference time.

On the test datasets The test datasets A &B, C, D and F are different in terms of the complexity of the target structures. Test datasets A &B, C and D are relatively simple compared to test dataset F. Complexity of dataset F is attributed to the dataset generation process which included complex junctions and stem loops for the Eterna 100 challenge. We refer the readers to Anderson-Lee et al. [5] for more details. Such complexities, however, we are not part of the SPRNA training dataset yet SPRNA still recorded a competitive performance on test dataset F-69/100 (69%).

This shows that SPRNA learns novel RNA folding patterns that can be generalized to new unseen target structures. We argue that augmenting the SPRNA training set with complex RNA sequences can improve its generalizability subsequently leading to better performance on more complex RNA Inverse Folding problems tasks. This is a key future research direction.

Future Work

SPRNA has recorded very competitive yet promising results. Despite this, however, it did not perform well on dataset F containing complex loops and junctions (for Eterna challenging puzzles). Part of our future work will involve augmenting the training dataset with complex sequences to allow for better generalization and performance. In addition, we are also keen on designing molecular-aware value and policy networks that accept bases and base pairs as atomic input of Nitrogenous bases. This can be achieved by using graph neural network architectures such as GNN [42] or RGCN [68]. This design choice can lead to better performance due to better molecular representation.

Conclusion

In this paper, we presented a new RNA Inverse Folding model called SPRNA. By performing a one-step look-ahead using a deep value network, it selects the optimal action given base or the base pair set. The model recorded the best score on two test datasets while recording similar accuracy performance on one dataset. An RNA Inverse Folding model should design sequences that fold according to the given target as well as have the desired MFE and GC content. SPRNA designed GC-richer thus more stable sequences. Augmenting the training dataset with complex samples can help a one-step look-ahead model like SPRNA design complex RNA. Similarly, designing molecular-aware policy or value networks can also lead to better performance. The latter and the former are two key interesting future research fronts.

Appendix 1 Value Network Training

The value network was trained according to the following parameters and hyperparameter constraints. These were selected using Optuna [1]. framework (Table 10).

Appendix 2 One-Hot Feature Encoding Results

The experiments according to one-hot encoding did not yield better scores compared to the coding scheme of the results presented in the experiment section of this paper (Table 11).

Appendix 3 Software and Hardware Settings

The SPRNA was trained with PyTorch 1.13 and Python 3.7. All the experiments were run on NVIDIA V100.

Acknowledgements This work was granted access to the HPC/AI resources of IDRIS under the allocation 2023-AD010614071 made by GENCI.

Data availability All the datasets used in this work are publicly available. All the references for training and testing datasets are provided in Table 1.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Akiba T, Sano S, Yanase T, et al. Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, p. 2623–2631.

2. Akutsu T. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. Discrete Appl Math. 2000;104(1–3):45–62.
3. Alberts B, Johnson A, Lewis J, et al. Molecular motors. Molecular biology of the cell. 4th ed. New York: Garland Science; 2002.
4. Anderson JW, Sizikova E, Badugu A, et al. FRNAkenstein: multiple target inverse RNA folding. BMC Bioinformatics. 2012;13:1–12.
5. Anderson-Lee J, Fisker E, Kosaraju V, et al. Principles for predicting RNA secondary structure design difficulty. J Mol Biol. 2016;428(5):748–57.
6. Anthony T, Tian Z, Barber D. Thinking fast and slow with deep learning and tree search. Advances in Neural Information Processing Systems 2017;30:5366–5376.
7. Avihoo A, Churkin A, Barash D. RNAexinv: an extended inverse RNA folding from shape and physical attributes to sequences. BMC Bioinformatics. 2011;12:1–8.
8. Ba JL, Kiros JR, Hinton GE. Layer normalization. 2016. arXiv preprint arXiv:1607.06450
9. Bellman R. A Markovian decision process. Journal of Mathematics and Mechanics. 1957;6:679–84.
10. Breiman L. Random forests. Mach Learn. 2001;45:5–32.
11. Bringmann K, Grandoni F, Saha B, et al. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. SIAM J Comput. 2019;48(2):481–512.
12. Busch A, Backofen R. Info-RNA—a fast approach to inverse RNA folding. Bioinformatics. 2006;22(15):1823–31.
13. Cazenave T. Nested Monte-Carlo search. In: Twenty-First International Joint Conference on Artificial Intelligence, 2009.
14. Cazenave T, Fournier T. Monte Carlo inverse folding. In: Monte Carlo Search: First Workshop, MCS 2020, Held in Conjunction with IJCAI 2020, Virtual Event, January 7, 2021, Proceedings 1. Springer; 2021. p. 84–99.
15. Chua K, Calandra R, McAllister R, et al. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. Advances in Neural Information Processing Systems 2018;31:4759–4770.
16. Churkin A, Retwitzer MD, Reinharz V, et al. Design of RNAs: comparing programs for inverse RNA folding. Brief Bioinformatics 2018;19(2):350–8.
17. Cleaves HJJ, et al. Watson–Crick pairing. Encyclopedia of astrobiology. 2015. p. 2650.
18. Cortes C, Vapnik V. Support-vector networks. Mach Learn. 1995;20:273–97.
19. Crick F. Central dogma of molecular biology. Nature. 1970;227(5258):561–3.
20. Deisenroth M, Rasmussen CE. Pilco: a model-based and data-efficient approach to policy search. In: Proceedings of the 28th International Conference on machine learning (ICML-11), 2011, p. 465–72.
21. Doherty EA, Batey RT, Masquida B, et al. A universal mode of helix packing in RNA. Nat Struct Biol. 2001;8(4):339–43.
22. Dromi N, Avihoo A, Barash D. Reconstruction of natural RNA sequences from RNA shape, thermodynamic stability, mutational robustness, and linguistic complexity by evolutionary computation. J Biomol Struct Dyn. 2008;26(1):147–61.
23. Drory Retwitzer M, Reinharz V, Ponty Y, et al. incaRNAfbinv: a web server for the fragment-based design of RNA sequences. Nucleic Acids Res. 2016;44(W1):W308–14.
24. Eastman P, Shi J, Ramsundar B, et al. Solving the RNA design problem with reinforcement learning. PLoS Comput Biol. 2018;14(6): e1006176.
25. Esmaili-Taheri A, Ganjtabesh M. ERD: a fast and reliable tool for RNA design including constraints. BMC Bioinform. 2015;16(1):1–11.
26. Esmaili-Taheri A, Ganjtabesh M, Mohammad-Noori M. Evolutionary solution for the RNA design problem. Bioinformatics. 2014;30(9):1250–8.
27. Gao JZ, Li LY, Reidys CM. Inverse folding of RNA pseudoknot structures. Algorithms Mol Biol. 2010;5:1–19.
28. Garcia-Martin JA, Clote P, Dotu I. RNAifold: a constraint programming algorithm for RNA inverse folding and molecular design. J Bioinform Comput Biol. 2013;11(02):1350001.
29. Garcia-Martin JA, Dotu I, Clote P. RNAifold 2.0: a web server and software to design custom and RFAM-based RNA molecules. Nucleic Acids Res. 2015;43(W1):W513–21.
30. Griffiths-Jones S, Bateman A, Marshall M, et al. Rfam: an RNA family database. Nucleic Acids Res. 2003;31(1):439–41.
31. Hamada M, Kiryu H, Sato K, et al. Prediction of RNA secondary structure using generalized centroid estimators. Bioinformatics. 2009;25(4):465–73.
32. Hamming RW. Error detecting and error correcting codes. Bell Syst Tech J. 1950;29(2):147–60.
33. He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, p. 770–8.
34. Hochreiter S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. Int J Uncertain Fuzziness Knowl-Based Syst. 1998;6(02):107–16.
35. Hofacker IL. Vienna RNA secondary structure server. Nucleic Acids Res. 2003;31(13):3429–31.
36. Hofacker IL, Fontana W, Stadler PF, et al. Fast folding and comparison of RNA secondary structures. Monatshefte für Chemie/Chem Mon. 1994;125(2):167–88.
37. Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Netw. 1989;2(5):359–66.
38. Ieong S, Kao MY, Lam TW, et al. Predicting RNA secondary structures with arbitrary pseudoknots by maximizing the number of stacking pairs. J Comput Biol. 2003;10(6):981–95.
39. Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning. PMLR; 2015. p. 448–56.
40. Isaacs FJ, Dwyer DJ, Ding C, et al. Engineered riboregulators enable post-transcriptional control of gene expression. Nat Biotechnol. 2004;22(7):841–7.
41. Jain S, Tao Y, Schlick T. Inverse folding with RNA-as-graphs produces a large pool of candidate sequences with target topologies. J Struct Biol. 2020;209(3): 107438.
42. Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. 2016. arXiv preprint arXiv:1609.02907.
43. Kleinkauf R, Houwaart T, Backofen R, et al. antaRNA-multi-objective inverse folding of pseudoknot RNA using ant-colony optimization. BMC Bioinform. 2015;16(1):1–7.
44. Kleinkauf R, Mann M, Backofen R. antaRNA: ant colony-based RNA sequence design. Bioinformatics. 2015;31(19):3114–21.
45. Kocsis L, Szepesvári C. Bandit based Monte-Carlo planning. In: European Conference on Machine Learning. Springer; 2006. p. 282–93.
46. Levin A, Lis M, Ponty Y, et al. A global sampling approach to designing and reengineering RNA secondary structures. Nucleic Acids Res. 2012;40(20):10041–52.
47. Lillicrap TP, Hunt JJ, Pritzel A, et al. Continuous control with deep reinforcement learning. 2015. arXiv preprint arXiv:1509.02971.
48. Lorenz R, Bernhart SH, Hönerzu Siederdissen C, et al. ViennaRNA package 2.0. Algorithms Mol Biol. 2011;6:1–14.
49. Lyngsø RB, Pedersen CN. RNA pseudoknot prediction in energy-based models. J Comput Biol. 2000;7(3–4):409–27.
50. MacQueen J, et al. Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley

Symposium on Mathematical Statistics and Probability, Oakland, CA, USA, 1967, p. 281–97.
51. Mathews DH, Disney MD, Childs JL, et al. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. Proc Natl Acad Sci. 2004;101(19):7287–92.
52. Minuesa G, Alsina C, Garcia-Martin JA, et al. MoiRNAifold: a novel tool for complex in silico RNA design. Nucleic Acids Res. 2021;49(9):4934–43.
53. Mnih V, Kavukcuoglu K, Silver D, et al. Playing Atari with deep reinforcement learning. 2013. arXiv preprint arXiv:1312:5602
54. Mnih V, Badia AP, Mirza M, et al. Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning. PMLR; 2016. p. 1928–37.
55. Nussinov R, Jacobson AB. Fast algorithm for predicting the secondary structure of single-stranded RNA. Proc Natl Acad Sci. 1980;77(11):6309–13.
56. Obonyo S, Nicolas J, Owuor D. Designing RNA sequences through self-play. In: IJCCI; 2022. p. 305–12.
57. Portela F. An unexpectedly effective Monte Carlo technique for the RNA inverse folding problem. BioRxiv. 2018:345587.
58. Quinlan JR. Induction of decision trees. Mach Learn. 1986;1:81–106.
59. Reinharz V, Ponty Y, Waldispühl J. A weighted sampling algorithm for the design of RNA sequences with targeted secondary structure and nucleotide distribution. Bioinformatics. 2013;29(13):i308–15.
60. Retwitzer MD, Reinharz V, Churkin A, et al. incaRNAfbinv 2.0: a webserver and software with motif control for fragment-based design of RNAs. Bioinformatics. 2020;36(9):2920–2.
61. Rish I, et al. An empirical study of the naive bayes classifier. In: IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, 2001, p. 41–6.
62. Rosenblatt F. The perceptron, a perceiving and recognizing automaton project para. Cornell Aeronautical Laboratory; 1957.
63. Rosin CD. Nested rollout policy adaptation for Monte Carlo tree search. In: IJCAI; 2011. p. 649–4.
64. Rummery GA, Niranjan M. On-line Q-learning using connectionist systems, vol. 37. Cambridge: University of Cambridge, Department of Engineering; 1994.
65. Runge F, Stoll D, Falkner S, et al. Learning to design RNA. 2018. arXiv preprint arXiv:1812.11951.
66. Saha B. Fast & space-efficient approximations of language edit distance and RNA folding: an amnesic dynamic programming approach. In: 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). IEEE; 2017. p. 295–306.
67. Schaffner KF. The Watson–Crick model and reductionism. Br J Philos Sci. 1969;20(4):325–48.
68. Schlichtkrull M, Kipf TN, Bloem P, et al. Modeling relational data with graph convolutional networks. In: The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15. Springer; 2018. p. 593–607.
69. Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization. In: International Conference on Machine Learning. PMLR; 2015. p. 1889–97.
70. Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. 2017. arXiv preprint arXiv:1707.06347.
71. Silver D, Huang A, Maddison CJ, et al. Mastering the game of go with deep neural networks and tree search. Nature. 2016;529(7587):484–9.
72. Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of go without human knowledge. Nature. 2017;550(7676):354–9.
73. Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15(1):1929–58.
74. Sutton RS. Dyna, an integrated architecture for learning, planning, and reacting. ACM Sigart Bull. 1991;2(4):160–3.
75. Sutton RS, Barto AG. Reinforcement learning: an introduction. Cambridge: MIT Press; 2018.
76. Świechowski M, Godlewski K, Sawicki B, et al. Monte Carlo tree search: a review of recent modifications and applications. Artif Intell Rev. 2023;56(3):2497–562.
77. Taneda A. Multi-objective optimization for RNA design with multiple target secondary structures. BMC Bioinform. 2015;16(1):1–20.
78. Trotta E. On the normalization of the minimum free energy of RNAs by sequence length. PLoS ONE. 2014;9(11): e113380.
79. Vinyals O, Babuschkin I, Czarnecki WM, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature. 2019;575(7782):350–4.
80. Wang T, Wei JJ, Sabatini DM, et al. Genetic screens in human cells using the CRISPR-Cas9 system. Science. 2014;343(6166):80–4.
81. Watford M, Wu G. Protein. Adv Nutr. 2018;9(5):651–3.
82. Watkins CJ, Dayan P. Q-learning. Mach Learn. 1992;8:279–92.
83. Weinbrand L, Avihoo A, Barash D. RNAfbinv: an interactive java application for fragment-based design of RNA sequences. Bioinformatics. 2013;29(22):2938–40.
84. Williams RJ. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn. 1992;8:229–56.
85. Wold S, Esbensen K, Geladi P. Principal component analysis. Chemom Intell Lab Syst. 1987;2(1–3):37–52.
86. Yang X, Yoshizoe K, Taneda A, et al. RNA inverse folding using Monte Carlo tree search. BMC Bioinform. 2017;18(1):1–12.
87. Zemora G, Waldsich C. RNA folding in living cells. RNA Biol. 2010;7(6):634–41.
88. Zuker M, Mathews DH, Turner DH. Algorithms and thermodynamics for RNA secondary structure prediction: a practical guide. In: RNA biochemistry and biotechnology. Berlin: Springer; 1999. p. 11–43.