Monte Carlo Inverse RNA Folding

Tristan Cazenave and Hamza Touzani

LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France

Abstract. The Inverse RNA Folding problem deals with designing a sequence of nucleotides that will fold into a desired target structure. Generalized Nested Rollout Policy Adaptation (GNRPA) is a Monte Carlo search algorithm for optimizing a sequence of choices. It learns a playout policy to intensify the search of the state space near the current best sequence. The algorithm uses a prior on the possible actions so as to perform non uniform playouts when learning the instance of problem at hand. We trained a Transformer neural network on the Inverse RNA Folding problem using the Rfam database. This network is used to generate a prior for every Eterna100 puzzle. GNRPA is used with this prior to solve some of the instances of the Eterna100 dataset. The Transformer prior gives better result than handcrafted heuristics.

Keywords: RNA · Monte Carlo Search · Transformers.

1 Introduction

The Inverse RNA Folding problem is essentially the inverse of the folding problem. Instead of finding the secondary structure of a given RNA sequence, the inverse folding problem deals with designing an RNA sequence that will fold into a desired target secondary structure. RNA design has applications in RNA-based therapeutics and diagnostics.

Monte Carlo Tree Search (MCTS) [1, 2] has been successfully applied to many games and problems [3]. It originates from the computer game of Go [4] with a method initially based on simulated annealing [5]. The principles underlying MCTS then evolved into a method using statistics on random games to find the best move [6]. It was further refined to use a tree at the beginning of the playouts [1, 2]. It was a revolution in computer Go and then in computer game playing. It is used in modern approaches to game playing such as Alpha Zero [7].

Nested Monte Carlo Search (NMCS) [8] is a recursive algorithm which uses lower level playouts to bias its playouts, memorizing the best sequence at each level. At each stage of the search, the move with the highest score at the lower level is played by the current level. At each step, a lower-level search is launched for all possible moves and the move with the best score is memorized. At level 0, a random playout is performed, playing randomly until a terminal state is reached. At the end, the score for the position is returned. NMCS has given good results on many problems like puzzle solving, single player games [9], retrosynthesis [10] or the Inverse RNA Folding problem with the NEMO program [11].

Based on NMCS, the Nested Rollout Policy Adaptation (NRPA) algorithm was introduced [12]. NRPA combines nested search, memorizing the best sequence of moves found, and the online learning of a playout policy using this sequence. NRPA achieved world records in Morpion Solitaire and crossword puzzles and has been applied to many problems such as object wrapping [13], traveling salesman with time window [14, 15], vehicle routing problems [16, 17] or network traffic engineering [18].

GNRPA (Generalized Nested Rollout Policy Adaptation) [19] generalizes the way the probability is calculated using a temperature and a bias. It has been applied to some problems like Inverse RNA Folding [20] and Vehicle Routing Problem (VRP) [21].

This paper is organized as follows. The second section presents the Inverse RNA Folding problem. The third section presents Monte Carlo Search. The fourth section deals with Monte Carlo Search for Inverse RNA Folding. The fifth section describes experimental results.

2 Inverse RNA Folding

The design of molecules with specific properties is an important topic for health related research. RNA molecules are long molecules composed of four possible nucleotides. Molecules can be represented as strings composed of the four characters A, C, G, U. For RNA molecules of length N, the size of the state space of possible strings is exponential in N. It can be very large for long molecules. The sequence of nucleotides folds back on itself to form what is called its secondary structure. It is possible to find in a polynomial time the folded structure of a given sequence. However, the opposite, which is the Inverse RNA Folding problem, is hard [22].

We compare Monte Carlo Search algorithms on the Eterna100 benchmark which contains 100 RNA secondary structure puzzles of varying degrees of difficulty. A puzzle consists of a given structure under the dot-bracket notation. This notation defines a structure as a sequence of parentheses and points each representing a base. The matching parentheses symbolize the paired bases and the dots the unpaired ones. The puzzle is solved when a sequence of the four nucleotides A,U,G and C, folding according to the target structure, is found. In some puzzles, the value of certain bases is imposed. Figure 1 gives an example of an Eterna100 problem.

Where human experts have managed to solve the 100 problems of the benchmark, no program has so far achieved such a score. The best score so far is 95/100 by NEMO, NEsted MOnte Carlo RNA Puzzle Solver [11] and by GNRPA [20].

3 Monte Carlo Search

This section presents the GNRPA algorithm which is a generalization of the NRPA algorithm to the use of a prior.

The Nested Rollout Policy Adaptation (NRPA) [12] algorithm is an effective combination of NMCS and the online learning of a playout policy. NRPA holds world records for Morpion Solitaire and crosswords puzzles.

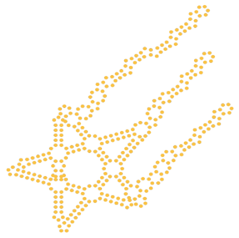

Fig. 1. Example of a RNA design puzzle from Eterna100: the Shooting Star. The associated target structure is: ((((((((((((((((...(.(..((.(((.(((.((....).))).)))).).)..))..))))))).(((((..(.(..((.(((.(((((.(((.((((.((....).)))).))))).))).)))).).)..))..)(....)..))))(((((((((...))))).(((((...)))))(.(....).).)))).)))) .((((.((((...(((((((...))))))).(((((...))))))))(((((((...))))))).(((((...))))))))((((..(....)(..(..(..(.(.(((.((((.((....).)))).)))).)..)..).)..))))).)))))).

In NRPA/GNRPA each move is associated to a weight stored in an array called the policy. The goal of these two algorithms is to learn these weights using the best sequences of moves found during the search. The weights are used in the softmax function to produce a playout policy that generates good sequences of moves.

NRPA/GNRPA use nested search. In NRPA/GNRPA, each level takes a policy as input and returns a sequence and its associated score. At any level > 0, the algorithm makes numerous recursive calls to the lower level, adapting the policy each time with the best sequence of moves to date. The changes made to the policy do not affect the policy in higher levels. At level 0, NRPA/GNRPA return the sequence obtained by the playout function as well as its associated score.

The playout function sequentially constructs a random solution biased by the weight of the moves until it reaches a terminal state. At each step, the function performs Gibbs sampling, choosing the actions with a probability given by the softmax function.

Let w_m be the weight associated to a move m in the policy. In NRPA, the probability of choosing move m is defined by:

$$p_m = \frac{e^{w_m}}{\sum_k e^{w_k}}$$

where k is an element of the set of possible moves, including m.

GNRPA [19] generalizes the way the probability is calculated using a temperature τ and a bias β_m. The temperature makes it possible to address the exploration/exploitation trade-off. The probability of choosing move m becomes:

$$p_m = \frac{e^{w_m + \beta_m}}{\sum_k e^{w_k + \beta_k}}$$

By taking $\beta_k = 0$, we find again the formula for NRPA.

In NRPA it is possible to initialize the weights according to a heuristic relevant to the problem to solve. In GNRPA, the policy initialization is replaced by the bias. It is sometimes more practical to use β_k biases than to initialize the weights as the codes for the moves can be different from the codes of the biases.

The algorithm to perform playouts in GNRPA is given in algorithm 1. The main GNRPA algorithm is given in algorithm 3. Is calls the adapt algorithm to modify the policy weights so as to reinforce the best sequence of the current level. The policy is passed by reference to the adapt algorithm which is given in algorithm 2.

The principle of the adapt function is to increase the weights of the moves of the best sequence of the level and to decrease the weights of all possible moves by an amount proportional to their probabilities of being played. $\delta_{bm} = 0$ when $b \neq m$ and $\delta_{bm} = 1$ when $b = m$.

Algorithm 1 The playout algorithm

```
 1: playout ($policy$)
 2:   $state \leftarrow root$
 3:   **while** true **do**
 4:     **if** terminal($state$) **then**
 5:       **return** (score ($state$), $sequence(state)$)
 6:     **end if**
 7:     $z \leftarrow 0$
 8:     **for** $m \in$ possible moves for $state$ **do**
 9:       $o[m] \leftarrow e^{policy[code(m)]+\beta_m}$
10:       $z \leftarrow z + o[m]$
11:     **end for**
12:     choose a $move$ with probability $\frac{o[move]}{z}$
13:     play ($state$, $move$)
14:   **end while**
```

Algorithm 2 The adapt algorithm

```
 1: adapt ($policy$, $sequence$)
 2:   $polp \leftarrow policy$
 3:   $state \leftarrow root$
 4:   **for** $b \in sequence$ **do**
 5:     $z \leftarrow 0$
 6:     **for** $m \in$ possible moves for $state$ **do**
 7:       $o[m] \leftarrow e^{policy[code(m)]+\beta_m}$
 8:       $z \leftarrow z + o[m]$
 9:     **end for**
10:     **for** $m \in$ possible moves for $state$ **do**
11:       $p_m \leftarrow \frac{o[m]}{z}$
12:       $polp[code(m)] \leftarrow polp[code(m)] - \alpha(p_m - \delta_{bm})$
13:     **end for**
14:     play ($state$, $b$)
15:   **end for**
16:   $policy \leftarrow polp$
```

Algorithm 3 The GNRPA algorithm.

```
1:  GNRPA ($level$, $policy$)
2:    if level == 0 then
3:      return playout ($policy$)
4:    else
5:      $bestScore \leftarrow -\infty$
6:      for N iterations do
7:        (score,new) $\leftarrow$ GNRPA($level - 1$, $policy$)
8:        if score $\geq$ bestScore then
9:          bestScore $\leftarrow$ score
10:         seq $\leftarrow$ new
11:       end if
12:       $policy$ $\leftarrow$ adapt ($policy$, $seq$)
13:     end for
14:     return (bestScore, seq)
15:   end if
```

Algorithm 4 The GNRPALR algorithm.

```
1:  GNRPALR ($level$, $policy$)
2:    if level == 0 then
3:      return playout ($policy$)
4:    else
5:      $bestScore \leftarrow -\infty$
6:      $repetitions \leftarrow 0$
7:      while $repetitions < R$ do
8:        (score,new) $\leftarrow$ GNRPALR($level - 1$, $policy$)
9:        if score = bestScore then
10:         $repetitions \leftarrow repetitions + 1$
11:       end if
12:       if score > bestScore then
13:         $repetitions \leftarrow 0$
14:         bestScore $\leftarrow$ score
15:         seq $\leftarrow$ new
16:       end if
17:       $policy$ $\leftarrow$ adapt ($policy$, $seq$)
18:     end while
19:     return (bestScore, seq)
20:   end if
```

The GNRPA with Limited Repetitions (GNRPALR) algorithm has been recently proposed in [23]. It stops searching at a level if the best score repeats a given number of times. the GNRPALR algorithm is described in algorithm 4.

4 Monte Carlo Search for Inverse RNA Folding

This section presents our contributions to Monte Carlo Inverse RNA Folding. It starts with defining the state space of the problem. It then recalls previous work. The training of a transformer to learn a policy is then presented, followed by the use of the most probable sequence to build a prior and of the use of the Transformer policy to sample the state space.

4.1 The State Space

The state space is the set of all sequences that are consistent with the secondary structure given as input. The secondary structure is a sequence of characters. The possible characters are '.', '(' and ')'. For each '.' in the input sequence there are four possible characters in the nucleotide sequence: 'A', 'C', 'G' and 'U'. Each '(' character is associated to the ')' character that closes the expression it has opened (e.g. when the same number of '(' and ')' are in between the two). Six pairs of characters are possible to replace the '(' and the corresponding ')': 'CG', 'GC', 'GU', 'UG', 'AU' and 'UA'. When a nucleotide sequence is complete, the ViennaRNA package [24] is used to fold the sequence and verify if it folds into the target structure.

The score of a sequence of nucleotide is computed the same way as NEMO [11] using the ViennaRNA package [24]. Refinements on the scoring function are possible such as the ones presented in [25].

4.2 Previous Work

NEMO, the NEsted MOnte Carlo RNA Puzzle Solver [11] uses NMCS [8] with a handcrafted heuristic playout policy to solve the Inverse RNA Folding problem. It was tested on the Eterna100 dataset. NEMO solved 95 problems out of the 100 problems of Eterna100. This is the best score achieved so far.

The NEMO heuristic mainly consists in probabilities for pairs of bases.

Refinements of GNRPA achieved the same score as NEMO using the pairs of bases heuristics of NEMO as a prior [20]. The refinements that improved the results were Stabilized GNRPA [26], Beam GNRPA [27, 28], delayed learning [20, 29], and root and leaf parallelization [30].

In this paper, we will only present experimental results for standard GNRPA and GNRPALR. There is no doubt that the previously cited enhancements presented in [20] would improve the results.

Self-play reinforcement learning was also used for the Inverse RNA Folding problem [31].

4.3 Training a Transformer on RFAM

We trained a Transformer to find the nucleotides that fold into a predefined structure. During training, by taking for each position the most likely character as predicted by the model, the accuracy on the validation set reaches 90%. On the test set, we compute again the true accuracy and observe a respectable 60% achieved by the model for sequence comparison. When comparing structure accuracy, we also achieved decent results with 56% accuracy. Outputting a sequence from the model at once (using all likely characters from initial distributions) led to noticeably worse results (42%) compared to sequentially decoding each character.

4.4 The Transformer Prior

The transformer prior consists in sampling the most probable sequence according to the Transformer. At each step of the sequence the most probable move is played and the probabilities for all the moves are memorized. In the hand we have for each index of the sequence, and for each possible move at this index, an associated probability. The bias is then computed using a value of 3.0 if the move is the most probable one and $\beta_m = log(proba[m])$ if the move is not the most probable one. The principle is to use the Transformer policy for moves that are not the most probable one since β_m is used in a softmax function, and to give a stronger bias to the most probable move since this is the one used to assess the probabilities for the end of the most probable sequence.

4.5 Sampling Sequences

Given a target secondary structure, instead of constructing the sequence with the most likely characters as per the model output probabilities, we sample a character from those same distributions and sequentially construct a candidate solution.

We thus propose to generate batch of candidate sequences sampled from our model's distributions for a given target secondary structure and observe if any sequence comes close to a solution.

5 Experimental Results

We present results about sampling with the Transformer policy and about different GNRPA algorithms with the Transformer and the NEMO priors.

5.1 Sampling Sequences

Sampling one whole sequence could take up to 20 seconds for the longest structures on the Eterna100 benchmark. To test this approach, we only focus on 11 Eterna100 puzzles: Structure no. 78 and the last 11 ones (90 to 100). This allows us to confront our approach to the hardest structures (78, 97, 99, 100) that were not solved by neither NEMO nor GNRPA. The 7 other ones are also some of the harder ones in the benchmark

(90 was only solved in 2 hours by GNRPA, 91 is not solved by GNRPA but by NEMO, 94 is solved by GNRPA but not by NEMO).

For each structure, we sample 5000 sequences (that are not necessarily distinct). Each run of 5000 sequence generation lasted around 20 hours.

The results we obtained were quite interesting. First, we observe that problems 92 and 98 are solved in most cases (i.e structure accuracy of 1). More than 3000 sequences for problem 92 and more than 2000 sequences for problem 98 fold into the exact target secondary structure. To a lesser extent, many sequences generated for problem 93 and 95 also folded into the desired structure (respectively around 700 and 150). Using the model's distributions allows for solving quite difficult RNA structures.

However, we did not observe the same success for the harder structures. Problems 78, 90, 91, 94, 96, 97, 99 were never quite solved. No sequences sampled ever folded into the exact target structure. Perhaps the most interesting result of our experiments was our results on problem 100. Of the 5000 sampled sequences, 8 folded into the target secondary structure. While with this method we did not solve problems 90, 91, 94 and 96 although they were solved by either NEMO, GNRPA or both, we managed to solve problem 100 that was solved by neither.

Sampling from the model's distributions proven to be a promising approach, we aim to combine these distributions with more sophisticated algorithms such as NMCS and GNRPA to improve upon previous results.

5.2 GNRPA

We experimented with NRPA, GNRPA and GNRPALR with the NEMO and the Transformer priors. The NEMO prior is only a subset of the priors used in NEMO. It uses the heuristic functions on the pairs of bases. The pairs of bases heuristics are the main components of the NEMO prior. The same subset of heuristics were already used with GNRPA [20], equaling the 95/100 score of NEMO.

For the GNRPALR algorithm the best results were obtained with $R = 1$. The comparison of the algorithms is given in Table 1. GNRPALR with the Transformer prior gives the best results.

Table 1. Number of Eterna100 problems solved by different algorithms and various search time limits in seconds. The best algorithm is GNRPALR Transformer prior with 0 repetitions. GNRPA is much better than NRPA. The Transformer prior is slightly better than the NEMO prior when combined with GNRPA. The Transformer prior is much better than the NEMO prior when combined with GNRPALR.

Algorithm	32s	64s	128s	256s	512s	1,024s	2,048s	4,096s
NRPA	28	33	41	48	57	59	61	65
GNRPA NEMO prior	68	69	74	77	78	79	81	81
GNRPA Transformer prior	71	72	75	78	79	80	81	82
GNRPALR NEMO prior	70	72	76	79	79	81	81	82
GNRPALR Transformer prior	72	73	76	81	82	82	84	88

6 Conclusion

We described Monte Carlo Search algorithms that can be used to address the Inverse RNA Folding problem. These algorithms make use of a prior to perform non uniforms playouts. The prior can be learned training a Transformer neural network on the Rfam database of folded RNA sequences. Sampling sequences with the Transformer policy enables to solve difficult problems from the Eterna100 dataset which was not used for training. The most probable sequence played with the Transformer network can also be used as a prior for Monte Carlo Search algorithms. This prior improves on previous work when used in combination with the GNRPALR Monte Carlo Search algorithm.

References

1. Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. Donkers, editors, *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2006.
2. Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *17th European Conference on Machine Learning (ECML'06)*, volume 4212 of *LNCS*, pages 282–293. Springer, 2006.
3. Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.
4. Bruno Bouzy and Tristan Cazenave. Computer go: An AI oriented survey. *Artificial Intelligence*, 132(1):39–103, 2001.
5. Bernd Brügmann. Monte Carlo Go. Technical report, Max-Planke-Inst. Phys., Munich, 1993.
6. Bruno Bouzy and Bernard Helmstetter. Monte-Carlo Go developments. In *ACG*, volume 263 of *IFIP*, pages 159–174. Kluwer, 2003.
7. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.
8. Tristan Cazenave. Nested Monte-Carlo Search. In Craig Boutilier, editor, *IJCAI*, pages 456–461, 2009.
9. Jean Méhat and Tristan Cazenave. Combining UCT and Nested Monte Carlo Search for single-player general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):271–277, 2010.
10. Milo Roucairol and Tristan Cazenave. Comparing search algorithms on the retrosynthesis problem. In *AI to Accelerate Science and Engineering at AAAI 2023*. 2023.
11. Fernando Portela. An unexpectedly effective Monte Carlo technique for the RNA inverse folding problem. *BioRxiv*, page 345587, 2018.
12. Christopher D. Rosin. Nested rollout policy adaptation for Monte Carlo Tree Search. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 649–654, 2011.
13. Stefan Edelkamp, Max Gath, and Moritz Rohde. Monte-Carlo tree search for 3d packing with object orientation. In *KI 2014: Advances in Artificial Intelligence*, pages 285–296. Springer International Publishing, 2014.

14. Tristan Cazenave and Fabien Teytaud. Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In *Learning and Intelligent Optimization - 6th International Conference, LION 6*, pages 42–54, 2012.
15. Stefan Edelkamp, Max Gath, Tristan Cazenave, and Fabien Teytaud. Algorithm and knowledge engineering for the tsptw problem. In *Computational Intelligence in Scheduling (SCIS), 2013 IEEE Symposium on*, pages 44–51. IEEE, 2013.
16. Stefan Edelkamp, Max Gath, Christoph Greulich, Malte Humann, Otthein Herzog, and Michael Lawo. Monte-Carlo tree search for logistics. In *Commercial Transport*, pages 427–440. Springer International Publishing, 2016.
17. Tristan Cazenave, Jean-Yves Lucas, Hyoseok Kim, and Thomas Triboulet. Monte carlo vehicle routing. In *ATT at ECAI*, 2020.
18. Chen Dang, Cristina Bazgan, Tristan Cazenave, Morgan Chopin, and Pierre-Henri Wuillemin. Monte carlo search algorithms for network traffic engineering. In *ECML PKDD*, volume 12978 of *LNCS*, pages 486–501, 2021.
19. Tristan Cazenave. Generalized nested rollout policy adaptation. In *Monte Carlo Search at IJCAI*, 2020.
20. Tristan Cazenave and Thomas Fournier. Monte Carlo inverse folding. In *Monte Carlo Search at IJCAI*, 2020.
21. Julien Sentuc, Tristan Cazenave, and Jean-Yves Lucas. Generalized nested rollout policy adaptation with dynamic bias for vehicle routing. In *AI for Transportation at AAAI*, 2022.
22. Édouard Bonnet, Paweł Rzążewski, and Florian Sikora. Designing RNA secondary structures is hard. *Journal of Computational Biology*, 27(3), 2020.
23. Tristan Cazenave. Generalized nested rollout policy adaptation with limited repetitions. In *arxiv*. 2024.
24. Ronny Lorenz, Stephan H Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. Viennarna package 2.0. *Algorithms for molecular biology*, 6:1–14, 2011.
25. Max Ward, Eliot Courtney, and Elena Rivas. Fitness functions for rna structure design. *Nucleic Acids Research*, 51(7):e40–e40, 2023.
26. Tristan Cazenave, Jean-Baptiste Sevestre, and Matthieu Toulemont. Stabilized nested rollout policy adaptation. In *Monte Carlo Search at IJCAI*, 2020.
27. Tristan Cazenave and Fabien Teytaud. Beam nested rollout policy adaptation. In *Computer Games Workshop, ECAI 2012*, pages 1–12, 2012.
28. Stefan Edelkamp and Tristan Cazenave. Improved diversity in nested rollout policy adaptation. In *KI 2016: Advances in Artificial Intelligence - 39th Annual German Conference on AI, Klagenfurt, Austria, September 26-30, 2016, Proceedings*, pages 43–55, 2016.
29. Chen Dang, Cristina Bazgan, Tristan Cazenave, Morgan Chopin, and Pierre-Henri Wuillemin. Warm-starting nested rollout policy adaptation with optimal stopping. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 12381–12389. AAAI Press, 2023.
30. Tristan Cazenave and Nicolas Jouandeau. On the Parallelization of UCT. In *Computer Games Workshop*, Amsterdam, Netherlands, June 2007.
31. Stephen Obonyo, Nicolas Jouandeau, and Dickson Owuor. Designing RNA sequences by self-play. In *14th International Joint Conference on Computational Intelligence*, pages 305–312, 2022.