Generalized Nested Rollout Policy Adaptation with Limited Repetitions

Tristan Cazenave
LAMSADE, Université Paris Dauphine - PSL, Paris, France
tristan.cazenave@lamsade.dauphine.fr

Abstract

Generalized Nested Rollout Policy Adaptation (GNRPA) is a Monte Carlo search algorithm for optimizing a sequence of choices. We propose to improve on GNRPA by avoiding too deterministic policies that find again and again the same sequence of choices. We do so by limiting the number of repetitions of the best sequence found at a given level. Experiments show that it improves the algorithm for three different combinatorial problems: Inverse RNA Folding, the Traveling Salesman Problem with Time Windows and the Weak Schur problem.

1 Introduction

Monte Carlo Tree Search (MCTS) [28, 17] has been successfully applied to many games and problems [5]. It originates from the computer game of Go [4] with a method based on simulated annealing [6]. The principle underlying MCTS is to learn the move to play using statistics on random games. In the early times of MCTS, random games were played with a uniform policy. Computer Go program soon used non uniform playout policies, learning the policy with optimization algorithms [18]. Playout policies were replaced with neural network evaluations for computer Go with the AlphaGo program [37], and then for other games such as Chess and Shogi with the Alpha Zero program [38]. There have been numerous applications of MCTS following the notorious success in Computer Go, ranging from predicting the structure of large protein complexes [7] to wind farm layout optimization [2].

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. After the searches following each possible move have been run, the move of the best sequence at the current level is played. At the lowest level, playouts are performed. They can be uniformly random playouts [8] or they can be biased using heuristic probabilities for possible moves [31]. Each playout returns the sequence of moves being made and the score of the terminal position. NMCS has given good results on many combinatorial problems such as puzzle solving and single player games [30], the Inverse RNA Folding problem [31] or chemical retrosynthesis [35].

Nested Rollout Policy Adaptation (NRPA) [34]. combines nested search, memorizing the best sequence of moves found at each level, and the online learning of a playout policy using this sequence. NRPA has world records in Morpion Solitaire and crossword puzzles and has also been applied to many other combinatorial problems such as the Traveling Salesman Problem with Time Windows [16, 21], 3D Packing with Object Orientation [23], the physical traveling salesman problem [24], the Multiple Sequence Alignment problem [25], Logistics [22, 13], Graph Coloring [14], Vehicle Routing Problems [22, 12], Network Traffic Engineering [19], Virtual Network Embedding [26] or the Snake in the Box [20].

Generalized Nested Rollout Policy Adaptation (GNRPA) [10] generalizes the way the probability is calculated using a bias. The bias is a heuristic that performs non uniform playouts and using it usually gives much better results than uniform playouts. The use of a bias has been theoretically

17th European Workshop on Reinforcement Learning (EWRL 2024).

demonstrated more general than the initialization of the weights. The GNRPA paper also provides a theoretical derivation of the learning of the policy, using a cross entropy loss associated to a softmax. GNRPA has been applied to some difficult problems such as Inverse RNA Folding [11] and Vehicle Routing Problems [36] with better results than NRPA.

This work presents GNRPA with Limited Repetitions (GNRPALR) a modification to GNRPA that makes it more flexible with regard to the number of iterations at every level. The principle is to stop the iterations at a level when the policy of this level becomes too deterministic. NRPA and GNRPA can waste a lot of time in the last iterations of a level when the policy has become too deterministic as they always replay the same sequence and do not explore alternative sequences anymore in the lower levels. To avoid this behavior we replace the for loop that performs a fixed number of iterations at a level with a while loop that has a threshold on the number of repetitions of the best score at this level.

This paper is organized as follows. The second section describes NRPA, GNRPA and GNRPALR. The third section presents experimental results for three difficult combinatorial problems: Inverse RNA Folding, Traveling Salesman with Time Windows (TSPTW) and Weak Schur Numbers. For these three problems GNRPALR improves much on GNRPA. Moreover the speedups of GNRPALR over GNRPA increase with the search time.

2 Monte Carlo Search

This section presents the GNRPA algorithm which is a generalization of the NRPA algorithm to the use of a prior. It also presents the GNRPALR algorithm which is a modification of the GNRPA algorithm to dynamically stop the search at every level.

2.1 GNRPA

The Nested Rollout Policy Adaptation (NRPA) [34] 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.

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 thanks to the best sequences of actions found during the search. The weights are used in a playout policy that generates good sequences of actions.

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 lower levels, adapting the policy each time with the best solution to date. At level 0, NRPA/GNRPA return the sequence of actions generated by the playout function and its associated score.

The playout function sequentially constructs a random sequence of actions biased by the weights of the moves until it reaches a terminal state. It chooses the actions with a probability equal to the application of the softmax function to the weights.

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 [10] generalizes the way the probability is calculated using a bias β_m. The probability of choosing move m is:

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

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. It calls the adapt algorithm to modify the policy weights so as to reinforce the weights associated to 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.

The definition of δ_{bm} in the adapt algorithm is:

$$\delta_{bm} = 0 \Leftrightarrow b \neq m$$

$$\delta_{bm} = 1 \Leftrightarrow b = m$$

Algorithm 1 The playout algorithm. The moves in the playouts are played with a probability equal to the softmax function applied to the weights plus the bias of the possible moves.

```
playout (policy)
    state ← root
    while true do
        if terminal(state) then
            return (score (state), sequence(state))
        end if
        z ← 0
        for m ∈ possible moves for state do
            o[m] ← e^{policy[code(m)]+β_m}
            z ← z + o[m]
        end for
        choose a move with probability o[move]/z
        play (state, move)
    end while
```

Algorithm 2 The adapt algorithm. The moves of the best sequence are reinforced and the probability of playing the possibles moves are subtracted to the weights of the possible moves.

```
adapt (policy, sequence)
    polp ← policy
    state ← root
    for b ∈ sequence do
        z ← 0
        for m ∈ possible moves for state do
            o[m] ← e^{policy[code(m)]+β_m}
            z ← z + o[m]
        end for
        for m ∈ possible moves for state do
            p_m ← o[m]/z
            polp[code(m)] ← polp[code(m)] − α(p_m − δ_bm)
        end for
        play (state, b)
    end for
    policy ← polp
```

Algorithm 3 The GNRPA algorithm. The recursive call and the adapt function are called a fixed number of times at each level.

```
GNRPA (level, policy)
  if level == 0 then
    return playout (policy)
  else
    bestScore ← −∞
    for N iterations do
      (score,new) ← GNRPA(level − 1, policy)
      if score ≥ bestScore then
        bestScore ← score
        seq ← new
      end if
      policy ← adapt (policy, seq)
    end for
    return (bestScore, seq)
  end if
```

Algorithm 4 The GNRPALR algorithm. The recursive call and the adapt function are called until the score of the best sequence has been sent back by the lower level a fixed number of times.

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

2.2 GNRPALR

GNRPALR repairs a defect in GNRPA and NRPA. Both algorithms spend a lot of time in the last iterations of a level finding many times the same best sequence at the lower level. They are stuck in a local minima and they do not explore enough to get out of it. In order to avoid this we use a simple measure of how exploratory the policy of the level is. The simple measure is the number of repetitions at the level of the score of the current best sequence of moves. When this number reaches a predefined threshold the recursive calls are stopped and the best sequence is returned. We also experimented with other measures of the exploratory power of the policy such as the entropy of the policy, but the best results were obtained with the number of repetitions. Moreover the number of repetitions is more simple than the entropy of the policy and is easier to understand and to tune.

GNRPALR is described in algorithm 4. It uses the same adapt and playout functions as GNRPA and the structure of the algorithm is similar to GNRPA. The main difference is at line 7 where instead of the for loop that runs a fixed number of iterations in GNRPA, there is a while loop that stops when the algorithm reaches a fixed number of repetitions of the score of the best sequence. The R hyperparameter has to be tuned for each problem. In our experiments the best values range from 0 to

5 repetitions. The number of repetitions is updated at lines 9 to 11. It is reset to 0 at line 13 when a new best sequence is found. The algorithm stops the recursive calls and returns the best score and the best sequence to the level above when the number of times the score of the best sequence has been found at the current level is equal to R.

This is a simple modification to GNRPA that enables to solve problems much faster for long thinking times. Being simple is a quality for an improvement to a search algorithm since it can readily be used by practitioners at a very small development cost and still bring large gains.

3 Experimental Results

We now compare GNRPA and GNRPALR for three difficult combinatorial problems: Inverse RNA Folding, TSPTW and Weak Schur Numbers. For each problem we give the evolution of the scores obtained by the algorithms with the logarithm of the search time. Experiments were run on AMD EPYC-Rome processors at 2GHz.

3.1 The Inverse RNA Folding Problem

The design of molecules with specific properties is an important topic for health related research. The RNA design problem also named the Inverse RNA Folding problem is a difficult combinatorial problem. This problem is important for scientific fields such as bioengineering, pharmaceutical research, biochemistry, synthetic biology and RNA nanostructures [31].

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 [3].

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 brackets and dots each representing a base. The matching brackets 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, that folds according to the target structure, is found. In some puzzles, the value of certain bases is imposed. Figure 1 gives an example of a difficult Eterna100 problem. This is the problem number 90 of the dataset and it is called Gladius.

Human experts have solved the 100 problems of the benchmark. Search algorithms are not yet able to reach this score. The best score so far is 95/100 by NEMO, NEsted MOnte Carlo RNA Puzzle Solver [31], using NMCS with heuristic playouts, and by GNRPA using the main part of the NEMO prior [11].

For our experiments we use a Transformer prior for GNRPA that gives better results than the NEMO prior. To generate the priors we first trained a Transformer policy network [39] to predict the next nucleotide of the folded sequences using the Rfam dataset [27]. We then sampled all the Eterna100 sequences choosing at each step the most probable nucleotide given by the policy. The bias is then computed using a bias of 3.0 if the move is the most probable one. If the move is not the most probable one, with $proba[m]$ the output of the Transformer policy network for move m, the bias is:

$$\beta_m = log(proba[m])$$

The score of a sequence of nucleotide at the end of a playout is computed the same way as NEMO [31] using the ViennaRNA package [29]:

$$score = \begin{cases} \frac{K}{1+\Delta G} & \text{if } K > 0 \\ K(1+\Delta G) & \text{else} \end{cases}$$

$$\text{with } K = 1 - \frac{BPD}{2*NumTargetPairs}$$

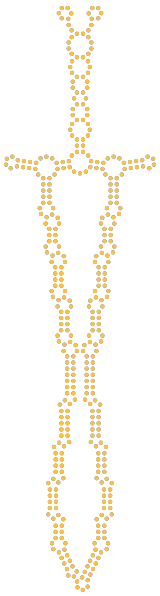

Figure 1: Gladius problem 90 from Eterna100. The associated target structure is:
(....)..(....(...(..(.(..(...(((.(((...((((....)))).(((((..(.(((..(.((((..(.((((..(.(((((((((.((((((.(((((.((((.
((((...)))).))).)))))).)))))).)))))).)))))))..).))))..).))))..).)))..).)))))).)))...(((.(((((.(..(((.(..((((.(..(((
(.(..(((((((.(((((((.(((((.(((((((.(((.((((((....))))))..)))).)))).)))))).)))))).)))))))).)..)))).)..)))).)..))).).
.)))))).((((....))))...))).)))...)..).)..)...)....)..(....)

Where BPD is the number of different pairs between the secondary structure of the sequence and the target structure. $NumTargetPairs$ is the number of pairs in the target structure. ΔG is the difference between the Minimum Free Energy of the secondary structure and the free energy that the sequence would have in the target structure.

The objective is to maximize the score function until a value of 1 is obtained, meaning that the problem is solved.

The search for sequences that have a given folding is chaotic. Changing a single nucleotide in a sequence can dramatically change the folding of the sequence and its associated structure. Monte Carlo Search does surprisingly well at finding good sequences in this chaotic search space. The reason could be that it is inherently a sequential decision making algorithm.

Figure 2 gives the comparison between GNRPALR and GNRPA for the 100 problems of Eterna100. The y-axis is the number of problems solved, out of the 100 problems, and the x-axis is the logarithm of the search time. The search times range from 1 second to 4,096 seconds, doubling at every step. We can observe that for short search times the algorithms solve a similar number of problems. For longer search times, and in particular for the 4,096 seconds limit, GNRPALR is much better than GNRPA. The number of repetitions we used for GNRPALR is $R = 0$. This means that for this problem the first repetition is the sign that the while loop should be stopped. The number of possible sequences is huge in the Inverse RNA Folding problems. The probability of finding again the same score for a sequence or the same sequence is quite low. This maybe why finding again the same score is the sign that the policy has become too deterministic.

3.2 The Traveling Salesman Problem with Time Windows

The TSPTW is a practical problem that has everyday applications. NRPA had good results for this problem [16, 21, 20] as well as for the related Vehicle Routing Problem [22, 12, 13].

The TSPTW has time constraints represented as time intervals during which cities are to be visited. With Monte Carlo Search, paths with violated constraints can be generated. As presented in [33] , a new score $score(p)$ of a path p can be defined as follow:

$$score(p) = -\Omega(p) \times 10^6 - cost(p)$$

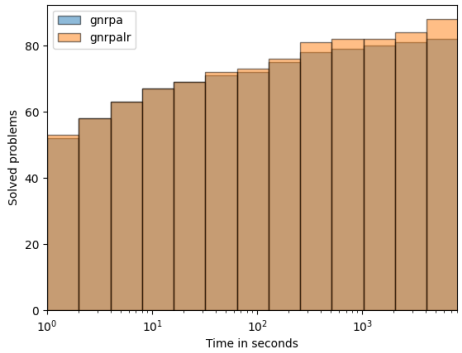

Figure 2: Comparison of GNRPA and GNRPALR for Inverse RNA Folding. The number of repetitions is set to 0 for GNRPALR. GNRPALR is eight times faster than GNRPA. It solves 88 problems in 4,096 seconds when GNRPA solves 82. The relative performance of GNRPALR improves with more search time. The tests are made using the 100 problems of Eterna100.

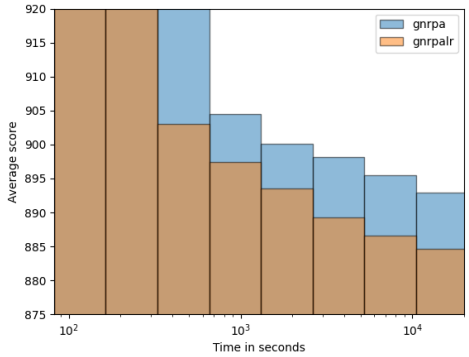

Figure 3: Comparison of GNRPA and GNRPALR for the TSPTW. The number of repetitions is set to 5 for GNRPALR. GNRPALR is much better than GNRPA for this problem. As we can see in the figure, the average score obtained with GNRPA after 10,000 seconds is obtained approximately 8 times slower than with GNRPALR. The averages are calculated over 100 runs of each algorithm with seeds ranging from 1 to 100. The problem solved is rc204.1, the most difficult problem from Solomon test suite for the TSPTW.

with, $cost(p)$ the sum of the distances of the path p and $\Omega(p)$ the number of violated constraints. The algorithm uses this evaluation so as to optimize first the number of violated constraints then the sum of the distances between locations of the path.

The prior uses the normalized distance between the current location and the arrival location of the move. Given max the maximum distance between two locations, min the minimum distance between two locations and d_m the distance between the current location and the arrival location, the bias is:

$$\beta_m = 10 \times \frac{d_m - min}{max - min}$$

We experiment with the most difficult instance of the standard test set: the rc204.1 problem [32].

Figure 3 gives the comparison between GNRPALR and GNRPA for this instance. GNRPALR finds feasible solutions faster. It finds better makespans faster than GNRPA. It improves the speedup for long search times compared to shorter search times.

Table 1: Lower bounds for Weak Schur numbers.

n	1	2	3	4	5	6
$WS(n)$	2	8	23	66	≥ 196	≥ 646
n	7	8	9	10	11	12
$WS(n)$	$\geq 2,146$	$\geq 6,976$	$\geq 22,536$	$\geq 71,256$	$\geq 243,794$	$\geq 815,314$

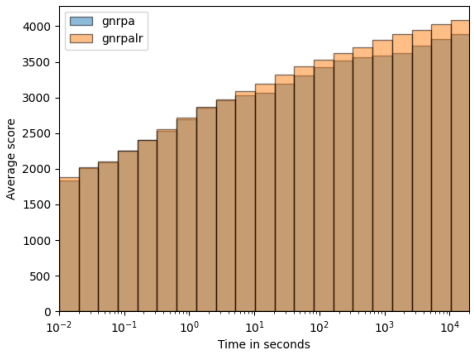

Figure 4: Comparison of GNRPA and GNRPALR for the Weak Schur problem of dimension 8. The number of repetitions is set to 0 for GNRPALR. The averages are calculated over 100 runs of each algorithm with seeds ranging from 1 to 100. The improvement of GNRPALR over GNRPA is greater for long search times. The average score obtained with GNRPA after 10,000 seconds is obtained approximately 8 times slower than with GNRPALR.

3.3 The Weak Schur Problem

The Weak Schur problem is to find a partition of consecutive numbers that contains as many consecutive numbers as possible, where a partition must not contain a number that is the sum of two previous numbers in the same partition.

The score of a terminal partition is the last number that was added to the partition before the next number could not be placed. The goal is to find partitions with high scores. These scores are lower bounds on the exact Weak Schur numbers.

An optimal partition of size 3 is for example:

```
1 2 4 8 11 22
3 5 6 7 19 21 23
9 10 12 13 14 15 16 17 18 20
```

And thus $WS(3) = 23$. The current records for the Weak Schur problem are given in table 1 from [1].

When possible, it is often a good move to put the next number in the same partition as the previous number. We use the same selective policy as in [9] which follows this heuristic. If it is legal to put the next number in the same partition as the previous number then it is the only legal move considered. Otherwise all legal moves are considered. The code of a move for the Weak Schur problem takes as input the partition of the move, the integer to assign and the previous number in the partition.

The comparison between GNRPA and GNRPALR for dimension 8 is given in Figure 4. We can observe that the difference in average score increases with the search time. It means that the lower bounds found by GNRPALR for long search times take much longer to be found by GNRPA, whereas the algorithms have similar results for short search times.

4 Conclusion

Enforcing a limited number of repetitions for GNRPA at the different levels of the nested searches avoids deterministic policies. It enables to search longer when discovering new better sequences and to stop search when the algorithm finds again and again the same best sequence. It speeds up GNRPA for three difficult combinatorial problems with thinking times up to 10,000 seconds for TSPTW and Weak Schur and 4,096 seconds for Inverse RNA Folding. The speedups are greater when the search time is longer. For Inverse RNA Folding, TSPTW and Weak Schur the speedup is approximately eight fold when searching during the longest tested time.

Future work will experiment with GNRPALR for other difficult combinatorial problems. It could also be interesting to tailor the R hyperparameter to the level as it costs less to have a great R at a level than in all levels. For example having twice the number of playouts at every level costs 2^L more times for a search at level L, when having twice the number of playouts at the lowest level only costs twice the time. If the policy is less deterministic due to a smaller R in the upper levels, the lowest level is naturally more exploratory. This is close to the idea of Stabilized NRPA [15] which performs more playouts at the lowest level while keeping the same number of adapt. Stabilized NRPA has already proven beneficial for SameGame, TSPTW and Expression Discovery .

References

[1] Romain Ageron, Paul Casteras, Thibaut Pellerin, Yann Portella, Arpad Rimmel, and Joanna Tomasik. New lower bounds for schur and weak schur numbers. *arXiv preprint arXiv:2112.03175*, 2021.

[2] Fangyun Bai, Xinglong Ju, Shouyi Wang, Wenyong Zhou, and Feng Liu. Wind farm layout optimization using adaptive evolutionary algorithm with monte carlo tree search reinforcement learning. *Energy Conversion and Management*, 252:115047, 2022.

[3] Edouard Bonnet, Paweł Rzążewski, and Florian Sikora. Designing RNA secondary structures is hard. *Journal of Computational Biology*, 27(3), 2020.

[4] Bruno Bouzy and Tristan Cazenave. Computer go: An AI oriented survey. *Artificial Intelligence*, 132(1):39–103, 2001.

[5] 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.

[6] Bernd Brügmann. Monte Carlo Go. Technical report, Max-Planke-Inst. Phys., Munich, 1993.

[7] Patrick Bryant, Gabriele Pozzati, Wensi Zhu, Aditi Shenoy, Petras Kundrotas, and Arne Elofsson. Predicting the structure of large protein complexes using alphafold and monte carlo tree search. *Nature communications*, 13(1):6028, 2022.

[8] Tristan Cazenave. Nested Monte-Carlo Search. In Craig Boutilier, editor, *IJCAI*, pages 456–461, 2009.

[9] Tristan Cazenave. Nested rollout policy adaptation with selective policies. In *CGW at IJCAI 2016*, 2016.

[10] Tristan Cazenave. Generalized nested rollout policy adaptation. In *Monte Carlo Search at IJCAI*, 2020.

[11] Tristan Cazenave and Thomas Fournier. Monte Carlo inverse folding. In *Monte Carlo Search at IJCAI*, 2020.

[12] Tristan Cazenave, Jean-Yves Lucas, Hyoseok Kim, and Thomas Triboulet. Monte carlo vehicle routing. In *ATT at ECAI*, 2020.

[13] Tristan Cazenave, Jean-Yves Lucas, Thomas Triboulet, and Hyoseok Kim. Policy adaptation for vehicle routing. *AI Communications*, 2021.

[14] Tristan Cazenave, Benjamin Negrevergne, and Florian Sikora. Monte Carlo graph coloring. In *Monte Carlo Search at IJCAI*, 2020.

[15] Tristan Cazenave, Jean-Baptiste Sevestre, and Matthieu Toulemont. Stabilized nested rollout policy adaptation. In *Monte Carlo Search at IJCAI*, 2020.

[16] 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.

[17] 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.

[18] Rémi Coulom. Computing elo ratings of move patterns in the game of Go. *ICGA Journal*, 30(4):198–208, 2007.

[19] 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.

[20] Chen Dang, Cristina Bazgan, Tristan Cazenave, Morgan Chopin, and Pierre-Henri Wuillemin. Warm-starting nested rollout policy adaptation with optimal stopping. In *AAAI 2023*, pages 12381–12389. AAAI Press, 2023.

[21] 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.

[22] 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.

[23] 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.

[24] Stefan Edelkamp and Christoph Greulich. Solving physical traveling salesman problems with policy adaptation. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–8. IEEE, 2014.

[25] Stefan Edelkamp and Zhihao Tang. Monte-Carlo tree search for the multiple sequence alignment problem. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015*, pages 9–17. AAAI Press, 2015.

[26] Maxime Elkael, Massinissa Ait Aba, Andrea Araldo, Hind Castel-Taleb, and Badii Jouaber. Monkey business: Reinforcement learning meets neighborhood search for virtual network embedding. *Computer Networks*, 216:109204, 2022.

[27] Ioanna Kalvari, Eric P Nawrocki, Nancy Ontiveros-Palacios, Joanna Argasinska, Kevin Lamkiewicz, Manja Marz, Sam Griffiths-Jones, Claire Toffano-Nioche, Daniel Gautheret, Zasha Weinberg, et al. Rfam 14: expanded coverage of metagenomic, viral and microrna families. *Nucleic Acids Research*, 49(D1):D192–D200, 2021.

[28] 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.

[29] 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.

[30] 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.

[31] Fernando Portela. An unexpectedly effective Monte Carlo technique for the RNA inverse folding problem. *BioRxiv*, page 345587, 2018.

[32] Jean-Yves Potvin and Samy Bengio. The vehicle routing problem with time windows part ii: genetic search. *INFORMS journal on Computing*, 8(2):165–172, 1996.

[33] Arpad Rimmel, Fabien Teytaud, and Tristan Cazenave. Optimization of the Nested Monte-Carlo algorithm on the traveling salesman problem with time windows. In *EvoApplications*, volume 6625 of *LNCS*, pages 501–510. Springer, 2011.

[34] 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.

[35] Milo Roucairol and Tristan Cazenave. Comparing search algorithms on the retrosynthesis problem. In *AI to Accelerate Science and Engineering at AAAI 2023*. 2023.

[36] 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.

[37] D. Silver, Aja Huang, Chris J. Maddison, A. Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, S. Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[38] 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.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.