Generalized Nested Rollout Policy Adaptation

Tristan Cazenave

LAMSADE, Université Paris-Dauphine, PSL, CNRS, France
Tristan.Cazenave@dauphine.psl.eu

Abstract. Nested Rollout Policy Adaptation (NRPA) is a Monte Carlo search algorithm for single player games. In this paper we propose to generalize NRPA with a temperature and a bias and to analyze theoretically the algorithms. The generalized algorithm is named GNRPA. Experiments show it improves on NRPA for different application domains: SameGame and the Traveling Salesman Problem with Time Windows.

1 Introduction

Monte Carlo Tree Search (MCTS) has been successfully applied to many games and problems [4].

Nested Monte Carlo Search (NMCS) [5] is an algorithm that works well for puzzles and optimization problems. It biases its playouts using lower level playouts. At level zero NMCS adopts a uniform random playout policy. Online learning of playout strategies combined with NMCS has given good results on optimization problems [22]. Other applications of NMCS include Single Player General Game Playing [16], Cooperative Pathfinding [2], Software testing [20], heuristic Model-Checking [21], the Pancake problem [3], Games [8] and the RNA inverse folding problem [18].

Online learning of a playout policy in the context of nested searches has been further developed for puzzles and optimization with Nested Rollout Policy Adaptation (NRPA) [23]. NRPA has found new world records in Morpion Solitaire and crosswords puzzles. NRPA has been applied to multiple problems: the Traveling Salesman with Time Windows (TSPTW) problem [9,11], 3D Packing with Object Orientation [13], the physical traveling salesman problem [14], the Multiple Sequence Alignment problem [15] or Logistics [12]. The principle of NRPA is to adapt the playout policy so as to learn the best sequence of moves found so far at each level.

The use of Gibbs sampling in Monte Carlo Tree Search dates back to the general game player Cadia Player and its MAST playout policy [1].

We now give the outline of the paper. The second section describes NRPA. The third section gives a theoretical analysis of NRPA. The fourth section describes the generalization of NRPA. The fifth section details optimizations of GNRPA. The sixth section gives experimental results for SameGame and TSPTW.

2 NRPA

NRPA learns a rollout policy by adapting weights on each action. Vanilla NRPA starts with all weights set to zero. During the playout phase, action is sampled with a probability proportional to the exponential of the associated weight. The playout algorithm is

given in Algorithm 1. The algorithm starts with initializing the sequence of moves that it will play (line 2). Then it performs a loop until it reaches a terminal states (lines 3-6). At each step of the playout it calculates the sum of all the exponentials of the weights of the possible moves (lines 7-10) and chooses a move proportional to its probability given by the softmax function (line 11). Then it plays the chosen move and adds it to the sequence of moves (lines 12-13). Each move is associated to a code which is usually independent of the state.

Then, the policy is adapted on the best current sequence found, by increasing the weight of the best actions and decreasing the weights of all the moves proportionally to their probabilities of being played. The Adapt algorithm is given in Algorithm 2. For all the states of the sequence passed as a parameter it adds α to the weight of the move of the sequence (lines 3-5). Then it reduces all the moves proportionally to $\alpha\times$ the probability of playing the move so as to keep the sum of logits unchanged (lines 6-12).

In NRPA, each nested level takes as input a policy, and returns a sequence. At each step, the algorithm makes a recursive call to the lower level and gets a sequence as a result. It adapts the policy to the best sequence of the level at each step. At level zero it makes a playout.

The NRPA algorithm is given in Algorithm 3. At level zero it simply performs a playout (lines 2-3). At greater levels it performs N iterations and for each iteration it calls itself recursively to get a score and a sequence (lines 4-7). If it finds a new best sequence for the level it keeps it as the best sequence (lines 8-11). Then it adapts the policy using the best sequence found so far at the current level (line 12).

NRPA balances exploitation by adapting the probabilities of playing moves toward the best sequence of the level, and exploration by using Gibbs sampling at the lowest level. It is a general algorithm that has proven to work well for many optimization problems.

Algorithm 1 The playout algorithm

1: playout ($state$, $policy$)
2: $sequence \leftarrow$ []
3: **while** true **do**
4: **if** $state$ is terminal **then**
5: **return** (score ($state$), $sequence$)
6: **end if**
7: $z \leftarrow 0.0$
8: **for** $m \in$ possible moves for $state$ **do**
9: $z \leftarrow z$ + exp ($policy$ [code(m)])
10: **end for**
11: choose a $move$ with probability $\frac{exp(policy[code(move)])}{z}$
12: $state \leftarrow$ play ($state$, $move$)
13: $sequence \leftarrow sequence + move$
14: **end while**

Algorithm 2 The Adapt algorithm

1: Adapt ($policy$, $sequence$)
2: $polp \leftarrow policy$
3: $state \leftarrow root$
4: **for** $move \in sequence$ **do**
5: $polp$ [code($move$)] $\leftarrow polp$ [code($move$)] + α
6: $z \leftarrow 0.0$
7: **for** $m \in$ possible moves for $state$ **do**
8: $z \leftarrow z$ + exp ($policy$ [code(m)])
9: **end for**
10: **for** $m \in$ possible moves for $state$ **do**
11: $polp$ [code(m)] $\leftarrow polp$ [code(m)] - $\alpha * \frac{exp(policy[code(m)])}{z}$
12: **end for**
13: $state \leftarrow$ play ($state$, $move$)
14: **end for**
15: $policy \leftarrow polp$

Algorithm 3 The NRPA algorithm.

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

3 Theoretical Analysis of NRPA

In NRPA each move is associated to a weight. The goal of the algorithm is to learn these weights so as to produce a playout policy that generates good sequences of moves. At each level of the algorithm the best sequence found so far is memorized. Let $s_1, ..., s_m$ be the sequence of states of the best sequence. Let n_i be the number of possible moves in a state s_i. Let $m_{i1}, ..., m_{in_i}$ be the possible moves in state s_i and m_{ib} be the move of the best sequence in state s_i. The goal is to learn to play the move m_{ib} in state s_i.

The playouts use Gibbs sampling. Each move m_{ik} is associated to a weight w_{ik}. The probability p_{ik} of choosing the move m_{ik} in a playout is the softmax function:

$$p_{ik} = \frac{e^{w_{ik}}}{\Sigma_j e^{w_{ij}}}$$

The cross-entropy loss for learning to play move m_{ib} is $C_i = -log(p_{ib})$. In order to apply the gradient we calculate the partial derivative of the loss: $\frac{\delta C_i}{\delta p_{ib}} = -\frac{1}{p_{ib}}$. We then calculate the partial derivative of the softmax with respect to the weights:

$$\frac{\delta p_{ib}}{\delta w_{ij}} = p_{ib}(\delta_{bj} - p_{ij})$$

Where $\delta_{bj} = 1$ if $b = j$ and 0 otherwise. Thus the gradient is:

$$\nabla w_{ij} = \frac{\delta C_i}{\delta p_{ib}} \frac{\delta p_{ib}}{\delta w_{ij}} = -\frac{1}{p_{ib}} p_{ib}(\delta_{bj} - p_{ij}) = p_{ij} - \delta_{bj}$$

If we use α as a learning rate we update the weights with:

$$w_{ij} = w_{ij} - \alpha(p_{ij} - \delta_{bj})$$

This is the formula used in the NRPA algorithm to adapt weights.

4 Generalization of NRPA

We propose to generalize the NRPA algorithm by generalizing the way the probability is calculated using a temperature τ and a bias β_{ij}:

$$p_{ik} = \frac{e^{\frac{w_{ik}}{\tau} + \beta_{ik}}}{\Sigma_j e^{\frac{w_{ij}}{\tau} + \beta_{ij}}}$$

4.1 Theoretical Analysis

The formula for the derivative of $f(x) = \frac{g(x)}{h(x)}$ is:

$$f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{h^2(x)}$$

So the derivative of p_{ib} relative to w_{ib} is:

$$\frac{\delta p_{ib}}{\delta w_{ib}} = \frac{\frac{1}{\tau}e^{\frac{w_{ib}}{\tau}+\beta_{ib}}\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}} - \frac{1}{\tau}e^{\frac{w_{ib}}{\tau}+\beta_{ib}}e^{\frac{w_{ib}}{\tau}+\beta_{ib}}}{(\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}})^2}$$

$$\frac{\delta p_{ib}}{\delta w_{ib}} = \frac{1}{\tau}\frac{e^{\frac{w_{ib}}{\tau}+\beta_{ib}}}{\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}}}\frac{\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}} - e^{\frac{w_{ib}}{\tau}+\beta_{ib}}}{\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}}}$$

$$\frac{\delta p_{ib}}{\delta w_{ib}} = \frac{1}{\tau}p_{ib}(1 - p_{ib})$$

The derivative of p_{ib} relative to w_{ij} with $j \neq b$ is:

$$\frac{\delta p_{ib}}{\delta w_{ij}} = -\frac{1}{\tau}\frac{e^{\frac{w_{ij}}{\tau}+\beta_{ij}}e^{\frac{w_{ib}}{\tau}+\beta_{ib}}}{(\Sigma_j e^{\frac{w_{ij}}{\tau}+\beta_{ij}})^2}$$

$$\frac{\delta p_{ib}}{\delta w_{ij}} = -\frac{1}{\tau}p_{ij}p_{ib}$$

We then derive the cross-entropy loss and the softmax to calculate the gradient:

$$\nabla w_{ij} = \frac{\delta C_i}{\delta p_{ib}}\frac{\delta p_{ib}}{\delta w_{ij}} = -\frac{1}{\tau}\frac{1}{p_{ib}}p_{ib}(\delta_{bj} - p_{ij}) = \frac{p_{ij} - \delta_{bj}}{\tau}$$

If we use α as a learning rate we update the weights with:

$$w_{ij} = w_{ij} - \alpha\frac{p_{ij} - \delta_{bj}}{\tau}$$

This is a generalization of NRPA since when we set $\tau = 1$ and $\beta_{ij} = 0$ we get NRPA.

The corresponding algorithms are given in Algorithms 4 and 5.

4.2 Equivalence of Algorithms

Let the weights and probabilities of playing moves be indexed by the iteration of the GNRPA level. Let w_{nij} be the weight w_{ij} at iteration n, p_{nij} be the probability of playing move j at step i at iteration n, δ_{nbj} the δ_{bj} at iteration n.

We have:

$$p_{0ij} = \frac{e^{\frac{1}{\tau}w_{0ij}+\beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau}w_{0ik}+\beta_{ik}}}$$

$$w_{1ij} = w_{0ij} - \frac{\alpha}{\tau}(p_{0ij} - \delta_{0bj})$$

$$p_{1ij} = \frac{e^{\frac{1}{\tau}w_{1ij}+\beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau}w_{1ik}+\beta_{ik}}} = \frac{e^{\frac{1}{\tau}w_{0ij}-\frac{\alpha}{\tau^2}(p_{0ij}-\delta_{0bj})+\beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau}w_{1ik}+\beta_{ik}}}$$

$$w_{2ij} = w_{1ij} - \frac{\alpha}{\tau}(p_{1ij} - \delta_{1bj}) = w_{0ij} - \frac{\alpha}{\tau}(p_{0ij} - \delta_{0bj} + p_{1ij} - \delta_{1bj})$$

By recurrence we get:

$$p_{nij} = \frac{e^{\frac{1}{\tau}w_{nij}+\beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau}w_{nik}+\beta_{ik}}} = \frac{e^{\frac{w_{0ij}}{\tau}-\frac{\alpha}{\tau^2}(\Sigma_k p_{kij}-\delta_{kbj})+\beta_{ij}}}{\Sigma_k e^{\frac{1}{\tau}w_{nik}+\beta_{ik}}}$$

From this equation we can deduce the equivalence between different algorithms. For example GNRPA_1 with $\alpha_1 = (\frac{\tau_1}{\tau_2})^2\alpha_2$ and τ_1 is equivalent to GNRPA_2 with α_2 and τ_2 provided we set w_{0ij} in GNRPA_1 to $\frac{\tau_1}{\tau_2}w_{0ij}$. It means we can always use $\tau = 1$ provided we correspondingly set α and w_{0ij}.

Another deduction we can make is we can set $\beta_{ij} = 0$ provided we set $w_{0ij} = w_{0ij} + \tau \times \beta_{ij}$. We can also set $w_{0ij} = 0$ and use only β_{ij} which is easier.

The equivalences mean that GNRPA is equivalent to NRPA with the appropriate α and w_{0ij}. However, it can be more convenient to use β_{ij} than to initialize the weights w_{0ij} as we will see for SameGame.

Algorithm 4 The generalized playout algorithm

```
1:  playout (state, policy)
2:    sequence ← []
3:    while true do
4:      if state is terminal then
5:        return (score (state), sequence)
6:      end if
7:      z ← 0
8:      for m ∈ possible moves for state do
9:        o[m] ← e^(policy[code(m)]/τ + β(m))
10:       z ← z + o[m]
11:     end for
12:     choose a move with probability o[move]/z
13:     state ← play(state, move)
14:     sequence ← sequence + move
15:   end while
```

Algorithm 5 The generalized 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: $o[m] \leftarrow e^{\frac{policy[code(m)]}{\tau} + \beta(m)}$
8: $z \leftarrow z + o[m]$
9: **end for**
10: **for** $m \in$ possible moves for $state$ **do**
11: $polp[code(m)] \leftarrow polp[code(m)] - \frac{\alpha}{\tau}(\frac{o[m]}{z} - \delta_{bm})$
12: **end for**
13: $state \leftarrow play(state, b)$
14: **end for**
15: $policy \leftarrow polp$

5 Optimizations of GNRPA

5.1 Avoid Calculating Again the Possible Moves

In problems such as SameGame the computation of the possible moves is costly. It is important in this case to avoid to compute again the possible moves for the best playout in the Adapt function. The possible moves have already been calculated during the playout that found the best sequence. The optimized playout algorithm memorizes in a matrix $code$ the codes of the possible moves during a playout. The cell $code[i][m]$ contains the code of the possible move of index m at the state number i of the best sequence. The state number 0 is the initial state of the problem. The $index$ array memorizes the index of the code of the best move for each state number, $len(index)$ is the length of the best sequence and $index[i]$ is the index of the best move for state number i.

5.2 Avoid the Copy of the Policy

Tha Adapt algorithm of NRPA and GNRPA considers the states of the sequence to learn as a batch. The sum of the gradients is calculated for the entire sequence and then applied. The way it is done in NRPA is by copying the policy to a temporary policy, modifying the temporary policy computing the gradient with the unmodified policy, and then copying the modified temporary policy to the policy.

When the number of possible codes is large copying the policy can be costly. We propose to change the Adapt algorithm to avoid to copy twice the policy at each Adapt call. We also use the memorized codes and index so as to avoid calculating again the possible moves of the best sequence.

The way to avoid copying the policy is to make a first loop to compute the probabilities of each move of the best sequence, lines 2-8 of Algorithm 6. The matrix $o[i][m]$ contains the probability for move index m in state number i, the array $z[i]$ contains

the sum of the probabilities of state number i. The second step is to apply the gradient directly to the policy for each state number i and each code, see lines 9-14.

Algorithm 6 The optimized generalized adapt algorithm

1: Adapt ($policy, code, index$)
2: **for** $i \in [0, len(index)[$ **do**
3: $z[i] \leftarrow 0$
4: **for** $m \in [0, len(code[i])[$ **do**
5: $o[i][m] \leftarrow e^{\frac{policy[code[i][m]]}{\tau} + \beta(m)}$
6: $z[i] \leftarrow z[i] + o[i][m]$
7: **end for**
8: **end for**
9: **for** $i \in [0, len(index)[$ **do**
10: $b \leftarrow index[i]$
11: **for** $m \in [0, len(code[i])[$ **do**
12: $policy[code[i][m]] \leftarrow policy[code[i][m]] - \frac{\alpha}{\tau}(\frac{o[i][m]}{z[i]} - \delta_{bm})$
13: **end for**
14: **end for**

6 Experimental Results

We now give experimental results for SameGame and TSPTW.

6.1 SameGame

The first algorithm we test is the standard NRPA algorithm with codes of the moves using a Zobrist hashing [24] of the cells of the moves [17,10,6]. The selective policy used is to avoid the moves of the dominant color except for moves of size two after move number ten. The codes of the possible moves of the best playout are recorded so as to avoid computing again the possible moves in the Adapt function. It is called NRPA.

Using Zobrist hashing of the moves and biasing the policy with β is better than initializing the weights at SameGame since there are too many possible moves and weights. We tried to reduce the possible codes for the moves but it gave worse results. The second algorithm we test is to use Zobrist hashing and the selective policy associated to the bias. It is GNRPA with $\tau = 1$ and $\beta_{ij} = min(n - 2 - tabu, 8)$, with $tabu = 1$ if the move is of size 2 and of the tabu color and $tabu = 0$ otherwise. The variable n being the number of cells of the move. The algorithm is called GNRPA.beta.

The third algorithm we test is to use Zobrist hashing, the selective policy, β and the optimized Adapt function. The algorithm is called GNRPA.beta.opt.

All algorithms are run 200 times for 655.36 seconds and average scores are recorded each time the search time is doubled.

The evolution of the average score of the algorithms is given in figure 1. We can see that GNRPA.beta is better than NRPA but that for scores close to the current record of the problem the difference is small. GNRPA.beta.opt is the best algorithm as it searches more than GNRPA.beta for the same time.

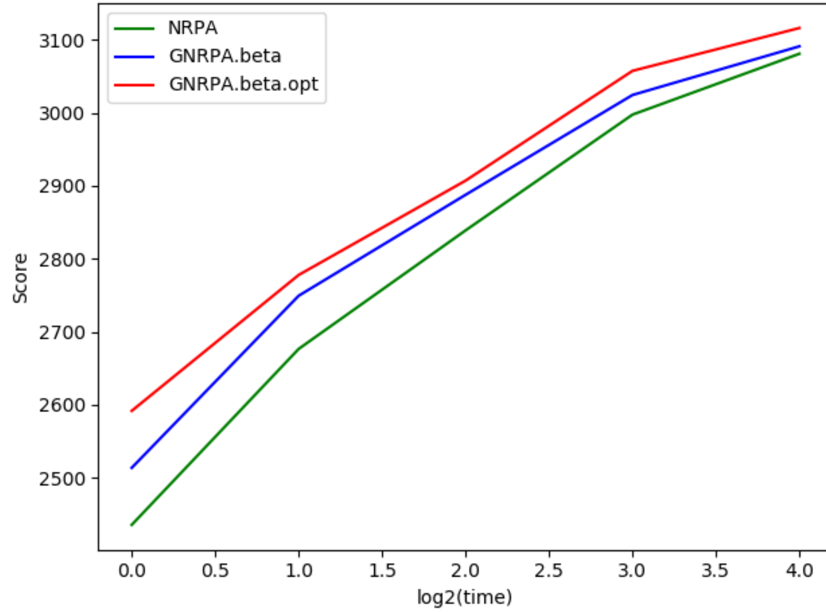

Fig. 1: Evolution of the average scores of the three algorithms at SameGame.

Table 1 gives the average scores for the three algorithms associated to the 95% confidence interval in parenthesis ($2 \times \frac{\sigma}{\sqrt{n}}$).

Table 1: Results for the first SameGame problem of the standard test suite.

Time	NRPA	GNRPA.beta	GNRPA.beta.opt
40.96	2435.12 (49.26)	2513.35 (53.57)	2591.46 (52.50)
81.92	2676.39 (47.16)	2749.33 (47.82)	2777.83 (48.05)
163.84	2838.99 (41.82)	2887.78 (39.50)	2907.23 (38.45)
327.68	2997.74 (21.39)	3024.68 (18.27)	3057.78 (13.52)
655.36	3081.25 (10.66)	3091.44 (10.96)	3116.54 (7.42)

6.2 TSPTW

The Traveling Salesman with Time Windows problem (TSPTW) is a practical problem that has everyday applications. NRPA can be used to efficiently solve practical logistics problems faced by large companies such as EDF [7].

In NRPA paths with violated constraints can be generated. As presented in [22] , a new score $Tcost(p)$ of a path p can be defined as follow:

$$Tcost(p) = cost(p) + 10^6 * \Omega(p),$$

with, $cost(p)$ the sum of the distances of the path p and $\Omega(p)$ the number of violated constraints. 10^6 is a constant chosen high enough so that the algorithm first optimizes the constraints.

The problem we use to experiment with the TSPTW problem is the most difficult problem from the set of [19].

In order to initialize β_{ij} we normalize the distances and multiply the result by ten. So $\beta_{ij} = 10 \times \frac{d_{ij}-min}{max-min}$, where min is the smallest possible distance and max the greatest possible one.

All algorithms are run 200 times for 655.36 seconds and average scores are recorded each time the search time is doubled.

Figure 2 gives the curves for the three GNRPA algorithms we haves tested with a logarithmic time scale for the x axis.

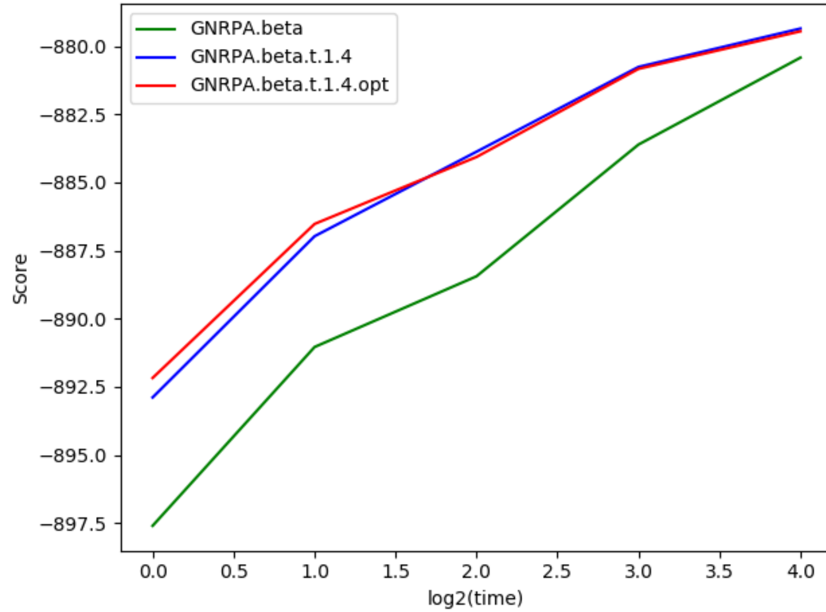

Fig. 2: Evolution of the average scores of the three algorithms for TSPTW.

We could not represent the curve for NRPA in figure 2 since the average values are too low. They are given in table 2. It is possible to improve much on standard NRPA by initializing the weights with the distances between cities [11,7]. However this solution is not practical for all problems as we have seen with SameGame and using a bias β is more convenient and general. We also tried initializing the weights with β instead of using β and we got similar results to the use of β.

We can see in figure 2 that using a temperature of 1.4 improves on a temperature of 1.0. Using the optimized Adapt function does not improve GNRPA for TSPTW since in the TSPTW problem the policy array and the number of possible moves is very small and copying the policy is fast.

The curve of the best algorithm is asymptotic toward the best value found by all algorithms. It reaches better scores faster.

Table 2 gives the average values for NRPA and the three GNRPA algorithms we have tested. As there is a penalty of 1 million for each constraint violation, NRPA has very low scores compared to GNRPA. This is why NRPA is not depicted in figure 2. For a search time of 655.36 seconds and not taking into account the constraints, NRPA usually reaches tour scores between -900 and -930. Much worse than GNRPA. We can observe that using a temparature is beneficial until we use 655.36 seconds and approach the asymptotic score when both algorithms have similar scores. The numbers in parenthesis in the table are the 95% confidence interval ($2 \times \frac{\sigma}{\sqrt{n}}$).

Table 2: Results for the TSPTW rc204.1 problem

Time	NRPA	GNRPA.beta	GNRPA.beta.t.1.4	GNRPA.beta.t.1.4.opt
40.96	-3745986.46 (245766.53)	-897.60 (1.32)	-892.89 (0.96)	-892.17 (1.04)
81.92	-1750959.11 (243210.68)	-891.04 (1.05)	-886.97 (0.87)	-886.52 (0.83)
163.84	-1030946.86 (212092.35)	-888.44 (0.98)	-883.87 (0.71)	-884.07 (0.70)
327.68	-285933.63 (108975.99)	-883.61 (0.63)	-880.76 (0.40)	-880.83 (0.32)
655.36	-45918.97 (38203.97)	-880.42 (0.30)	-879.35 (0.16)	-879.45 (0.17)

7 Conclusion

We presented a theoretical analysis and a generalization of NRPA named GNRPA. It uses a temperature τ and a bias β.

We have theoretically shown that using a bias is equivalent to initializing the weights. For SameGame initializing the weights can be difficult if we initialize all the weights at the start of the program since there are too many possible weights, whereas using a bias β is easier and improves search at SameGame. A lazy initialization of the weights would also be possible in this case and would solve the weight initialization problem for SameGame. For some other problems the bias could be more specific than the code of the move, i.e. a move could be associated to different bias depending on the state. In this case different bias could be used in different states for the same move which would not be possible with weight initialization.

We have also theoretically shown that the learning rate and the temperature can replace each other. Tuning the temperature and using a bias has been very beneficial for the TSPTW.

The remaining work is to apply the algorithm to other domains and to improve the way to design formulas for the bias β.

Acknowledgment

This work was supported in part by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR19-P3IA-0001 (PRAIRIE 3IA Institute).

References

1. Bjornsson, Y., Finnsson, H.: Cadiaplayer: A simulation-based general game player. IEEE Transactions on Computational Intelligence and AI in Games **1**(1), 4–15 (2009)
2. Bouzy, B.: Monte-carlo fork search for cooperative path-finding. In: Computer Games - Workshop on Computer Games, CGW 2013, Held in Conjunction with the 23rd International Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, August 3, 2013, Revised Selected Papers. pp. 1–15 (2013)
3. Bouzy, B.: Burnt pancake problem: New lower bounds on the diameter and new experimental optimality ratios. In: Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016, Tarrytown, NY, USA, July 6-8, 2016. pp. 119–120 (2016)
4. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (Mar 2012). https://doi.org/10.1109/TCIAIG.2012.2186810
5. Cazenave, T.: Nested Monte-Carlo Search. In: Boutilier, C. (ed.) IJCAI. pp. 456–461 (2009)
6. Cazenave, T.: Nested rollout policy adaptation with selective policies. In: Computer Games, pp. 44–56. Springer (2016)
7. Cazenave, T., Lucas, J.Y., Kim, H., Triboulet, T.: Monte carlo vehicle routing. In: Submitted (2020)
8. Cazenave, T., Saffidine, A., Schofield, M.J., Thielscher, M.: Nested monte carlo search for two-player games. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. pp. 687–693 (2016), http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12134
9. Cazenave, T., Teytaud, F.: 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, Paris, France, January 16-20, 2012, Revised Selected Papers. pp. 42–54 (2012)
10. Edelkamp, S., Cazenave, T.: Improved diversity in nested rollout policy adaptation. In: Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz). pp. 43–55. Springer (2016)
11. Edelkamp, S., Gath, M., Cazenave, T., Teytaud, F.: Algorithm and knowledge engineering for the tsptw problem. In: Computational Intelligence in Scheduling (SCIS), 2013 IEEE Symposium on. pp. 44–51. IEEE (2013)

12. Edelkamp, S., Gath, M., Greulich, C., Humann, M., Herzog, O., Lawo, M.: Monte-carlo tree search for logistics. In: Commercial Transport, pp. 427–440. Springer International Publishing (2016)
13. Edelkamp, S., Gath, M., Rohde, M.: Monte-carlo tree search for 3d packing with object orientation. In: KI 2014: Advances in Artificial Intelligence, pp. 285–296. Springer International Publishing (2014)
14. Edelkamp, S., Greulich, C.: Solving physical traveling salesman problems with policy adaptation. In: Computational Intelligence and Games (CIG), 2014 IEEE Conference on. pp. 1–8. IEEE (2014)
15. Edelkamp, S., Tang, Z.: Monte-carlo tree search for the multiple sequence alignment problem. In: Eighth Annual Symposium on Combinatorial Search (2015)
16. Méhat, J., Cazenave, T.: 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)
17. Negrevergne, B., Cazenave, T.: Distributed nested rollout policy for samegame. In: Workshop on Computer Games. pp. 108–120. Springer (2017)
18. Portela, F.: An unexpectedly effective monte carlo technique for the rna inverse folding problem. bioRxiv p. 345587 (2018)
19. Potvin, J.Y., Bengio, S.: The vehicle routing problem with time windows part ii: genetic search. INFORMS journal on Computing **8**(2), 165–172 (1996)
20. Poulding, S.M., Feldt, R.: Generating structured test data with specific properties using nested monte-carlo search. In: Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014. pp. 1279–1286 (2014)
21. Poulding, S.M., Feldt, R.: Heuristic model checking using a monte-carlo tree search algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015. pp. 1359–1366 (2015)
22. Rimmel, A., Teytaud, F., Cazenave, T.: Optimization of the Nested Monte-Carlo algorithm on the traveling salesman problem with time windows. In: Applications of Evolutionary Computation - EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part II. Lecture Notes in Computer Science, vol. 6625, pp. 501–510. Springer (2011). https://doi.org/10.1007/978-3-642-20520-0$_5$1, http://dx.doi.org/10.1007/978-3-642-20520-0_51
23. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo Tree Search. In: IJCAI. pp. 649–654 (2011)
24. Zobrist, A.L.: A new hashing method with application for game playing. ICCA journal **13**(2), 69–73 (1970)