Nested Rollout Policy Adaptation with Selective Policies

Tristan Cazenave*

PSL-Université Paris-Dauphine, LAMSADE CNRS UMR 7243, Paris, France

Abstract. Monte Carlo Tree Search (MCTS) is a general search algorithm that has improved the state of the art for multiple games and optimization problems. Nested Rollout Policy Adaptation (NRPA) is an MCTS variant that has found record-breaking solutions for puzzles and optimization problems. It learns a playout policy online that dynamically adapts the playouts to the problem at hand. We propose to enhance NRPA using more selectivity in the playouts. The idea is applied to three different problems: Bus regulation, SameGame and Weak Schur numbers. We improve on standard NRPA for all three problems.

1 Introduction

MCTS is a state of the art search algorithm that has greatly improved the level of play in games such as Go [12, 13] and Hex [22]. The principle underlying MCTS is to play random games and to use the statistics on the moves played during the games so as to find the best moves [25].

MCTS can also be applied to problems other than games [6]. Examples of non-games applications are Security, Mixed Integer Programming, Traveling Salesman Problem, Physics Simulations, Function Approximation, Constraint Problems, Mathematical Expressions, Planning and Scheduling.

Some MCTS algorithms have been tailored to puzzles and optimization problems. For example Nested Monte Carlo Search (NMCS) [7] gives good results for multiple optimization problems. NRPA is an improvement of NMCS that learns a playout policy online [28].

In this paper we improve NRPA adding selectivity in the playouts. We propose to modify the standard playout policy used by NRPA in order to avoid bad moves during playouts.

The paper is organized in three remaining sections: section two presents related works , section three details selective policies for different problems and section four gives experimental results.

2 Related Work

NMCS is an algorithm that improves Monte Carlo search with Monte Carlo search. It has different levels of nested playouts and an important feature of the algorithm is that it records the best sequence of moves at each search level. The algorithm was initially applied to puzzles such as Morpion Solitaire, SameGame and Sudoku.

* cazenave@lamsade.dauphine.fr

A further application of NMCS is the Snake-In-The-Box problem [23]. NMCS has beaten world records for this problem that has application in coding theory. The goal of the problem is to find the longest possible path in a high dimensional hypercube so that nodes in the path never have more than two neighboring nodes also in the path.

Bruno Bouzy improved NMCS using playout policies. For example for the Pancake problem [4] he uses domain specific playout policies so as to beat world records with an improved NMCS.

Another variation on NMCS is Monte-Carlo Fork Search [2] that branches deep in the playouts. It was successfully applied to complex cooperative pathfinding problems.

The Weak Schur problem [19] is a problem where informed playout policies can give much better results than standard policies when used with NMCS [3].

Policy learning has been successfully used for the Traveling Salesman with Time Windows (TSPTW) in combination with NMCS [27].

An effective combination of nested levels of search and of policy learning has been proposed with the NRPA algorithm [28]. NRPA holds world records for Morpion Solitaire and crosswords puzzles.

NRPA is given in algorithm 3. The principle is to learn weights for the possible actions so as to bias the playouts. The playout algorithm is given in algorithm 1. It performs Gibbs sampling, choosing the actions with a probability proportional to the exponential of their weights.

The weights of the actions are updated at each step of the algorithm so as to favor moves of the best sequence found so far at each level. The principle of the adaptation is to add 1.0 to the action of the best sequence and to decrease the weight of the other possible actions by an amount proportional to the exponential of their weight. The adaptation algorithm is given in algorithm 2.

Playout policy adaptation has also been used for games such as Go [20] or various other games with success [8].

NRPA works by biasing the probability of an action by looking up a weight associated to the action. An alternative is to make the bias a function of the current state and the proposed action [26].

An improvement of standard NRPA is to combine it with beam search yielding Beam NRPA [11].

Stefan Edelkamp and co-workers have applied the NRPA algorithm to multiple problems. They have optimized the algorithm for the TSPTW problem [10, 14].

Other applications deal with 3D packing with object orientation [16], the physical traveling salesman problem [17], the multiple sequence alignment problem [18], logistics [15] or cryptography [21].

3 Selective Policies

The principle underlying selective policies is to modify the legal moves so that moves that are unlikely to be good are pruned during playouts.

This can be done differently for each application of the algorithm. In this section we describe the move pruning for three problems: the bus regulation problem, SameGame and the Weak Schur problem.

Algorithm 1 The playout algorithm

playout ($state$, $policy$)
$sequence \leftarrow$ []
while true **do**
  if $state$ is terminal **then**
    return (score ($state$), $sequence$)
  end if
  $z \leftarrow 0.0$
  for m in possible moves for $state$ **do**
    $z \leftarrow z$ + exp ($k \times policy$ [code(m)])
  end for
  choose a $move$ with probability proportional to $\frac{exp(k \times policy[code(move)])}{z}$
  $state \leftarrow$ play ($state$, $move$)
  $sequence \leftarrow sequence$ + $move$
end while

Algorithm 2 The adapt algorithm

adapt ($policy$, $sequence$)
$polp \leftarrow policy$
$state \leftarrow root$
for $move$ in $sequence$ **do**
  $polp$ [code($move$)] $\leftarrow$ $polp$ [code($move$)] + α
  $z \leftarrow 0.0$
  for m in possible moves for $state$ **do**
    $z \leftarrow z$ + exp ($policy$ [code(m)])
  end for
  for m in possible moves for $state$ **do**
    $polp$ [code(m)] $\leftarrow$ $polp$ [code(m)] - $\alpha * \frac{exp(policy[code(m)])}{z}$
  end for
  $state \leftarrow$ play ($state$, $move$)
end for
$policy \leftarrow polp$

Algorithm 3 The NRPA algorithm

NRPA ($level$, $policy$)
if level == 0 **then**
  return playout (root, $policy$)
end if
$bestScore \leftarrow -\infty$
for N iterations **do**
  (result,new) $\leftarrow$ NRPA($level - 1$, $policy$)
  if result $\geq$ bestScore **then**
    bestScore $\leftarrow$ result
    seq $\leftarrow$ new
  end if
  policy $\leftarrow$ adapt (policy, seq)
end for
return (bestScore, seq)

3.1 Bus regulation

In the bus regulation problem [9] the bus regulator knows the location of all the buses of a bus line. At each stop he can decide to make a bus wait before continuing his route. Waiting at a stop can reduce the overall passengers waiting time. The score of a simulation is the sum of all the passengers waiting time. Optimizing a problem is finding a set of bus stopping times that minimizes the score of the simulation. It is possible to use rules to decide the bus waiting time given the number of stops before the next bus. Monte Carlo bus regulation with NMCS has been shown to improve on rule-based regulation.

In this paper we use NRPA to choose the bus waiting times. We compare the standard policy that can choose a waiting time between 1 and 5 minutes to a selective policy that always chooses a waiting time of 1 if there are fewer than δ stops before the next bus.

An important detail of the NRPA algorithm is the way moves are coded. A move code for the bus regulation problem takes into account the bus stop, the time of arrival to the bus stop and the number of minutes to wait before leaving the stop.

Algorithm 4 gives the rule used to compute the legal moves for the bus regulation problem.

Algorithm 4 Legal moves with a selective policy for the bus regulation problem.

```
legalMoves (moves)
moves ← [1 minute]
if next bus is at strictly less than δ stops then
    return moves
end if
for i in 2,max waiting time do
    add (i minutes) to moves
end for
return moves
```

3.2 SameGame

SameGame is a puzzle composed of a rectangular grid containing cells of different colors. A move removes connected cells of the same color. The cells of other colors fall to fill the void created by a move. At least two cells have to be removed for a move to be legal. The score of a move is the square of the number of removed cells minus two. A bonus of one thousand is credited for completely clearing the board.

MCTS has been quite successful for SameGame. SP-MCTS [30, 29], NMCS [7] and Nested MCTS [1] have reached great scores at SameGame. For all algorithms an effective improvement on random playouts is to use the tabu color strategy. As it is often beneficial to remove all the cells of the most frequent color in one move, the tabu color strategy avoids the moves of the most frequent color until all of its cells form only one group.

We propose to apply NRPA to SameGame and to improve on standard NRPA using selective policies.

There are many possible different moves at SameGame. So many moves that it is not possible to code them with a simple function without exceeding storage capacities. The way we deal with this problem is by using Zobrist hashing [31]. Zobrist hashing is popular in computer games such as Go and Chess [5]. It uses a 64 bits random integer for each possible color of each cell of the board. The code for a move is the XOR of the random numbers associated to the cells of the move. A transposition table is used to store the codes and their associated weights. The index of a move in the transposition table is its 16 lower bits. For each entry of the transposition table, a list of move codes and weights is stored.

It has been previously shown that in SameGame it is possible to improve simulation policies by allowing more randomness in the endgame [24].

What we do is that we use a modified version of the tabu color strategy. We allow moves of size two of the tabu color when the number of moves already played is greater than a threshold with value t. Algorithm 5 gives the function used to compute the legal moves for SameGame.

Algorithm 5 Legal moves with a selective policy for SameGame.

```
legalMoves (moves, tabuColor)
if only one move of the tabu color then
    tabuColor = noColor
end if
for m in possible moves do
    if color (m) == tabuColor then
        if nbCells (m) == 2 and nb moves played > t then
            add m to moves
        end if
    else
        add m to moves
    end if
end for
if moves is empty then
    for m in possible moves do
        add m to moves
    end for
end if
```

Figure 1 gives an example of a starting board at SameGame. We can see on the side the number of cells for each color. In this example the tabu color is green since it has 54 cells.

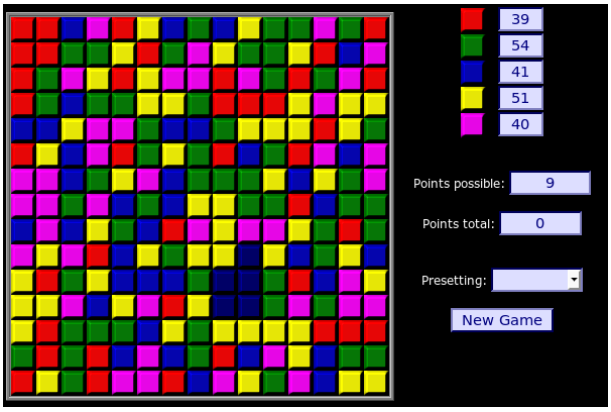

Fig. 1. Example of the initial state of a SameGame problem

3.3 Weak Schur Numbers

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 last number that was added to the partition before the next number could not be placed is the score of a partition. The goal is to find partitions with high scores.

The current records for the Weak Schur problem are given in table 1. The records for 7 and 8 are held by Bruno Bouzy using NMCS [3].

One of the best partitions of size three 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
```

When possible, it is often a good move to put the next number in the same partition as the previous number. The selective policy for SameGame follows this heuristic. The algorithm for the legal moves is given in algorithm 6. If it is legal to put the next number n 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.

Table 1. Current records for the Weak Schur problem

K	1	2	3	4
WS(K)	=2	=8	=23	=66
K	5	6	7	8
WS(K)	≥ 196	≥ 582	≥ 1736	≥ 5105

Algorithm 6 Legal moves with a selective policy for the Weak Schur problem.

```
legalMoves (moves, n)
moves ← []
for i in 0,nbPartitions do
  if previous number in partition i == n - 1 then
    if playing n in partition i is legal then
      add (i, n) to moves
    end if
  end if
end for
if moves == [] then
  for i in 0,nbPartitions do
    if playing n in partition i is legal then
      add (i, n) to moves
    end if
  end for
end if
```

4 Experimental Results

In order to evaluate a policy we run 200 times the NRPA algorithm with this policy. The scores are recorded starting at 0.01 seconds and for every power of two multiplied by 0.01. The algorithm is stopped after 163.84 seconds. We chose to record the scores this way in order to see the average improvement in score each time the search time is doubled. It has no influence on the NRPA algorithm.

4.1 Bus regulation

Table 2 gives the evolution with time of the best score of the standard NRPA algorithm in the No δ column and compares it to the evolution of the best score using rules with different δ for the legal moves. We can see that using $\delta = 3$ always gives better results than the No δ policy. For small times the $\delta = 4$ policy is much better than the other policies. For the longest search time (163.84 seconds), the playout rule that uses $\delta = 3$ has a score of 1,610 which is better than the playout policy without rules that has a score of 1,632.

4.2 SameGame

We performed two experiments for SameGame. The first experiment tests different playout strategies for the first problem of the test set. NRPA is run 200 times for each strategy and the evolution of the mean score with time is recorded.

The second experiment runs a level 4 search on the standard test set and the results are compared to the state of the art.

Table 2. Evaluation of selective policies for the bus regulation problem

time	No δ	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$
0.01	2,620	2,441	2,344	2,147	1,929
0.02	2,441	2,292	2,173	2,049	1,866
0.04	2,329	2,224	2,098	2,000	1,828
0.08	2,242	2,178	2,045	1,959	1,791
0.16	2,157	2,135	2,011	1,925	1,764
0.32	2,107	2,108	1,986	1,903	1,736
0.64	2,046	2,074	1,959	1,868	1,713
1.28	1,974	2,013	1,917	1,811	1,694
2.56	1,892	1,926	1,869	1,754	1,679
5.12	1,802	1,832	1,822	1,703	1,667
10.24	1,737	1,757	1,769	1,660	1,658
20.48	1,698	1,712	1,729	1,640	1,651
40.96	1,682	1,695	1,699	1,629	1,644
81.92	1,660	1,674	1,661	1,617	1,637
163.84	1,632	1,642	1,629	**1,610**	1,635

Table 3 gives the evolution of the mean score for problem one of the standard test set. We can observe that the tabu strategy is a large improvement over the standard policy (2,484.18 instead of 2,011.25). Allowing moves of the tabu color of size two when the playout length is greater than 10 gives even better results for long time settings even if it is worse for short time settings. The tabu policy is equivalent to the selective policy with $t = \infty$. For short time settings the tabu policy is the best one. However when more time is given to the algorithm it discovers ways of using the increased freedom of moves contained in the selective policy with $t > 10$ and eventually reaches a better score of 2,636.22 instead of 2,484.18 for the tabu policy.

Table 3. Evaluation of selective policies for SameGame

time	No tabu	tabu	$t > 0$	$t > 10$
0.01	155.83	352.19	260.37	257.59
0.02	251.28	707.56	487.27	505.05
0.04	340.18	927.63	666.91	677.57
0.08	404.27	1,080.64	810.29	822.44
0.16	466.15	1,252.14	924.41	939.30
0.32	545.78	1,375.78	1,043.97	1,058.54
0.64	647.63	1,524.37	1,185.77	1,203.91
1.28	807.20	1,648.16	1,354.69	1,356.81
2.56	1,012.42	1,746.74	1,508.10	1,497.90
5.12	1,184.77	1,819.43	1,616.44	1,605.86
10.24	1,286.25	1,886.48	1,737.35	1,712.17
20.48	1,425.55	1,983.42	1,859.12	1,879.10
40.96	1,579.67	2,115.80	2,078.30	2,100.47
81.92	1,781.40	2,319.44	2,329.73	2,384.24
163.84	2,011.25	2,484.18	2,539.75	**2,636.22**

Table 4 gives the best scores obtained with different algorithms for SameGame. The website js-games.de maintains the best scores obtained by its internet users. We can see that these scores are higher than the one obtained with Monte Carlo search. Little is known about the holders of these records. However we could exchange emails with a record holder who told us he is using beam search with a complex domain specific evaluation function to play SameGame.

We can also observe that NRPA with a selective policy has better scores than NMCS and SP-MCTS since the total of its scores is 80,030 for a level 4 search. It is approximately 2,000 points better than previous MCTS algorithms.

Table 4. Best scores for SameGame

position	NMCS	SP-MCTS	Selective NRPA	js-games.de
1	3,121	2,919	3,179	3,413
2	3,813	3,797	3,985	4,023
3	3,085	3,243	3,635	4,019
4	3,697	3,687	3,913	4,215
5	4,055	4,067	4,309	4,379
6	4,459	4,269	4,809	4,869
7	2,949	2,949	2,651	3,435
8	3,999	4,043	3,879	4,771
9	4,695	4,769	4,807	5,041
10	3,223	3,245	2,831	3,937
11	3,147	3,259	3,317	3,783
12	3,201	3,245	3,315	3,921
13	3,197	3,211	3,399	3,821
14	2,799	2,937	3,097	3,263
15	3,677	3,343	3,559	4,161
16	4,979	5,117	5,025	5,517
17	4,919	4,959	5,043	5,227
18	5,201	5,151	5,407	5,503
19	4,883	4,803	5,065	5,343
20	4,835	4,999	4,805	5,217
Total	77,934	78,012	80,030	**87,858**

4.3 Weak Schur Numbers

Table 5 and table 6 give the evolution with time of the best score of the standard NRPA algorithm and of the rule-based selective NRPA algorithm. The most striking example of the usefulness of a selective policy is for 9 partitions in table 6. The standard policy reaches 473 in 163.84 seconds when the selective policy reaches 7,538 for the same running time.

Table 5. Evaluation of selective policies for the Weak Schur problem

time	ws(6)	ws-rule(6)	ws(7)	ws-rule(7)
0.01	81	300	111	652
0.02	110	376	150	825
0.04	117	398	160	901
0.08	123	419	168	950
0.16	129	435	177	1,001
0.32	137	448	186	1,050
0.64	147	460	197	1,100
1.28	154	465	216	1,150
2.56	164	468	236	1,184
5.12	174	479	252	1,203
10.24	186	489	267	1,220
20.48	197	498	284	1,258
40.96	215	503	303	1,297
81.92	232	505	337	1,332
163.84	239	**506**	384	**1,356**

Table 6. Evaluation of selective policies for the Weak Schur problem

time	ws(8)	ws-rule(8)	ws(9)	ws-rule(9)
0.01	151	1,382	199	2,847
0.02	193	1,707	246	3,342
0.04	207	1,898	263	3,717
0.08	218	2,055	273	4,125
0.16	227	2,162	286	4,465
0.32	236	2,297	293	4,757
0.64	245	2,423	303	5,044
1.28	263	2,574	314	5,357
2.56	288	2,717	331	5,679
5.12	316	2,852	362	6,065
10.24	335	2,958	384	6,458
20.48	351	3,010	403	6,805
40.96	371	3,096	422	7,117
81.92	394	3,213	444	7,311
163.84	440	**3,318**	473	**7,538**

5 Conclusion

We have applied selective policies to three quite different problems. For each of these problems selective policies improve NRPA. We only used simple policy improvements, better performance could be obtained refining the proposed policies.

For all three problems, simple and effective rules could be found that avoid bad moves in playouts. In some other problems such as Morpion Solitaire [7, 28] for example such rules could be more difficult to find. Also, even if rules generally improve playouts they can make NRPA blind to some moves that are good in specific cases and prevent it from finding the best sequence.

For future work we intend to improve the selective policies and to apply the principle to other difficult problems.

References

1. Hendrik Baier and Mark H. M. Winands. Nested Monte-Carlo Tree Search for online planning in large MDPs. In *ECAI 2012 - 20th European Conference on Artificial Intelligence, Montpellier, France, August 27-31 , 2012*, pages 109–114, 2012.
2. Bruno Bouzy. Monte-carlo fork search for cooperative path-finding. In *Computer Games Workshop, CGW 2013, Held in Conjunction with the 23rd International Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, August 3, 2013, Revised Selected Papers*, pages 1–15, 2013.
3. Bruno Bouzy. An abstract procedure to compute weak schur number lower bounds. 2015.
4. Bruno Bouzy. An experimental investigation on the pancake problem. In *Computer Games Workshop, CGW 2015, Held in Conjunction with IJCAI 2015*, 2015.
5. Dennis M. Breuker. *Memory versus Search in Games*. PhD thesis, Universiteit Maastricht, 1998.
6. 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.
7. Tristan Cazenave. Nested Monte-Carlo Search. In Craig Boutilier, editor, *IJCAI*, pages 456–461, 2009.
8. Tristan Cazenave. Playout policy adaptation with move features. *Theoretical Computer Science*, 2016.
9. Tristan Cazenave, Flavien Balbo, and Suzanne Pinson. Monte-Carlo bus regulation. In *ITSC*, pages 340–345, St. Louis, 2009.
10. 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, Paris, France, January 16-20, 2012, Revised Selected Papers*, pages 42–54, 2012.
11. Tristan Cazenave and Fabien Teytaud. Beam nested rollout policy adaptation. In *Computer Games Workshop, CGW 2012, ECAI 2012*, pages 1–12, 2012.
12. 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.

13. Rémi Coulom. Computing elo ratings of move patterns in the game of Go. *ICGA Journal*, 30(4):198–208, 2007.
14. 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.
15. 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.
16. 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.
17. 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.
18. Stefan Edelkamp and Zhihao Tang. Monte-carlo tree search for the multiple sequence alignment problem. In *Eighth Annual Symposium on Combinatorial Search*, 2015.
19. Shalom Eliahou, Cyril Fonlupt, Jean Fromentin, Virginie Marion-Poty, Denis Robilliard, and Fabien Teytaud. Investigating monte-carlo methods on the weak schur problem. In *Evolutionary Computation in Combinatorial Optimization - 13th European Conference, EvoCOP 2013, Vienna, Austria, April 3-5, 2013. Proceedings*, pages 191–201, 2013.
20. Tobias Graf and Marco Platzner. Adaptive playouts in monte-carlo tree search with policy-gradient reinforcement learning. In *Advances in Computer Games - 14th International Conference, ACG 2015, Leiden, The Netherlands, July 1-3, 2015, Revised Selected Papers*, pages 1–11, 2015.
21. Bradley Hauer, Ryan Hayward, and Grzegorz Kondrak. Solving substitution ciphers with combined language models. In *COLING 2014, 25th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, August 23-29, 2014, Dublin, Ireland*, pages 2314–2325, 2014.
22. Shih-Chieh Huang, Broderick Arneson, Ryan B. Hayward, Martin Müller, and Jakub Pawlewicz. Mohex 2.0: A pattern-based MCTS Hex player. In *Computers and Games - 8th International Conference, CG 2013, Yokohama, Japan, August 13-15, 2013, Revised Selected Papers*, pages 60–71, 2013.
23. David Kinny. A new approach to the snake-in-the-box problem. In *ECAI*, volume 242, pages 462–467, 2012.
24. Simon Klein. Attacking SameGame using Monte-Carlo Tree Search: Using randomness as guidance in puzzles. 2015.
25. 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.
26. Simon M. Lucas, Spyridon Samothrakis, and Diego Perez Liebana. Fast evolutionary adaptation for monte carlo tree search. In *Applications of Evolutionary Computation - 17th European Conference, EvoApplications 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers*, pages 349–360, 2014.
27. Arpad Rimmel, Fabien Teytaud, and Tristan Cazenave. 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*, volume 6625 of *Lecture Notes in Computer Science*, pages 501–510. Springer, 2011.
28. Christopher D. Rosin. Nested rollout policy adaptation for Monte Carlo Tree Search. In *IJCAI*, pages 649–654, 2011.

29. Maarten PD Schadd, Mark HM Winands, Mandy JW Tak, and Jos WHM Uiterwijk. Single-player monte-carlo tree search for SameGame. *Knowledge-Based Systems*, 34:3–11, 2012.
30. Maarten PD Schadd, Mark HM Winands, H Jaap Van Den Herik, Guillaume MJ-B Chaslot, and Jos WHM Uiterwijk. Single-player monte-carlo tree search. In *Computers and Games*, pages 1–12. Springer, 2008.
31. Albert L Zobrist. A new hashing method with application for game playing. *ICCA journal*, 13(2):69–73, 1970.