Nested Monte-Carlo Tree Search for Online Planning in Large MDPs

Hendrik Baier[1] and **Mark H. M. Winands**[1]

Abstract. Monte-Carlo Tree Search (MCTS) is state of the art for online planning in large MDPs. It is a best-first, sample-based search algorithm in which every state in the search tree is evaluated by the average outcome of Monte-Carlo rollouts from that state. These rollouts are typically random or directed by a simple, domain-dependent heuristic. We propose *Nested Monte-Carlo Tree Search* (NMCTS), in which MCTS itself is recursively used to provide a rollout policy for higher-level searches. In three large-scale MDPs, SameGame, Clickomania and Bubble Breaker, we show that NMCTS is significantly more effective than regular MCTS at equal time controls, both using random and heuristic rollouts at the base level. Experiments also suggest superior performance to Nested Monte-Carlo Search (NMCS) in some domains.

1 INTRODUCTION

Monte-Carlo Tree Search (MCTS) [13, 19] is an online planning algorithm that combines the ideas of best-first tree search and Monte-Carlo evaluation. Since MCTS is based on sampling, it does not require a transition function in explicit form, but only a generative model of the domain. Because it grows a highly selective search tree guided by its samples, it can handle search spaces with large branching factors. By using Monte-Carlo rollouts, MCTS can take long-term rewards into account even with distant horizons. Combined with multi-armed bandit algorithms to trade off exploration and exploitation, MCTS has been shown to guarantee asymptotic convergence to the optimal policy [19], while providing approximations when stopped at any time.

MCTS has achieved considerable success in domains as diverse as the games of Go [16, 20], Amazons [21], LOA [35], and Ms. Pacman [18]; in General Game Playing [15], planning [24, 31], and optimization [14, 25, 27].

For the consistency of MCTS, i.e. for the convergence to the optimal policy, uniformly random rollouts beyond the tree are sufficient. However, heuristically informed rollout strategies typically greatly speed up convergence [17]. In this paper, we propose *Nested Monte-Carlo Tree Search* (NMCTS), using the results of lower-level searches recursively to provide rollout policies for searches on higher levels. We demonstrate the significantly stronger performance of NMCTS as compared to regular MCTS, at equal time controls, in the deterministic MDP domains SameGame, Clickomania and Bubble Breaker.

This paper is organized as follows. Section 2 provides the necessary background for the overview of related work on nested or meta-search in a Monte-Carlo framework in Section 3. Section 4 proposes Nested Monte-Carlo Tree Search, and Section 5 shows experimental results in our three test domains. Conclusions and future research follow in Section 6.

[1] Games and AI Group, Department of Knowledge Engineering, Maastricht University, The Netherlands, email: {hendrik.baier, m.winands}@maastrichtuniversity.nl

2 BACKGROUND

This section briefly outlines Markov Decision Processes and the common structure of value-based reinforcement learning algorithms. Monte-Carlo methods are introduced, and Monte-Carlo Tree Search is presented as the baseline search algorithm for this paper.

2.1 Markov Decision Processes

Markov decision processes (MDPs) represent a classic framework for modeling *reinforcement learning*—the task of an agent learning from experience how to act in an environment that provides feedback (cf. [32]). An MDP is defined as a 4-tuple $(S, A, P_{\cdot}(\cdot,\cdot), R_{\cdot}(\cdot,\cdot))$, where S is the set of *states* of the environment, A is the set of *actions* available to the agent, $P_a(s,s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that choosing action a in state s at time t will lead to state s' at time $t+1$ (the *transition function*), and $R_a(s,s')$ is the direct reward given to the agent after choosing action a in state s and transitioning to state s' (the *reward function*). The defining property of MDPs is the Markov property, stating that given a state s and an action a, the next state s' is conditionally independent of all preceding states and actions in the history of the agent.

In the case of episodic tasks, the agent chooses an action $a_t \in A$ based on the current state $s_t \in S$ of the environment at each discrete time step $t \in \{1, 2, 3, \ldots, T\}$. The environment then returns a new state s_{t+1} and a reward r_{t+1}. The agent chooses its actions according to a *policy*, a mapping $\pi(s,a) = Pr(a_t = a | s_t = s)$ from states of the environment to probabilities of selecting each possible action when in those states.

The goal of the agent is to find a policy that at any point in time t maximizes the *expected return*, the expected cumulative reward $R_t = \sum_{k=t+1}^{T} r_k$. In value-based reinforcement learning, this is accomplished by learning a *value function* $V^\pi(s) = E_\pi [R_t | s_t = s]$ representing the expected return when starting in a given state s and following policy π thereafter. For every MDP, there is a unique *optimal value function* V^* defined by $\forall s \in S.\ V^*(s) = \max_\pi V^\pi(s)$, and at least one *optimal policy* π^* achieving V^*.

Value-based RL algorithms typically find an optimal policy via *policy iteration*. This process alternatingly computes the value function V^π of the current policy π (policy evaluation), and uses the newfound V^π to derive a better policy π' (policy improvement).

2.2 Monte-Carlo Planning and Search in MDPs

Monte-Carlo methods are a class of model-free evaluation algorithms tailored to episodic tasks. Since episodic tasks provide well-defined sample returns for all visited states at the end of each episode, the return of a given state can be estimated by averaging the returns received after visiting that state in a number of episodes. According to the law of large numbers, such Monte-Carlo estimates converge to the true value function as the agent collects more and more experience.

Given a generative model of the environment—a model that is able to draw samples from the transition function—learning methods such as Monte-Carlo can be applied to simulated experience (*rollouts*), without actually interacting with the environment. This process is called *planning*. If planning is focused on improving an agent policy solely for the *current* state, i.e. on computing the optimal *next* action, it is called *search* [32].

2.3 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [13, 19] is a best-first search algorithm with Monte-Carlo evaluation of states. For each action decision of the agent, MCTS constructs a search tree $T \subseteq S$, starting from the current state as root. This tree is selectively deepened into the direction of the most promising actions, which are determined by the success of Monte-Carlo rollouts starting with these actions. After n rollouts, the tree contains $n+1$ states, for which distinct estimates of V^π are maintained.

MCTS works by repeating the following four-phase loop until computation time runs out [10]. Each loop represents one simulated episode of experience.

Phase one: *selection*. The tree is traversed from the root to one of the leaves. At each node, MCTS uses a selection policy to choose the action to sample from this state. Critical is a balance of exploitation of actions with high value estimates and exploration of actions with uncertain value estimates.

Phase two: *expansion*. When a leaf has been reached, one or more of its successors are added to the tree. In this paper, we always add the immediate successor of the leaf in the simulation.

Phase three: *rollout*. A rollout (also called "playout") policy is used to choose actions until the episode ends. Uniformly random action choices are sufficient to achieve convergence of MCTS to the optimal action in the limit, but rollout policies utilizing basic domain knowledge can improve convergence speed considerably.

Phase four: *backpropagation*. The cumulative reward of the finished episode is used to update value estimates of all states traversed during the simulation.

Listing 1 shows pseudocode of MCTS for deterministic environments, where not only the immediate next action choice is of interest, but also the best solution sequence for the entire task found so far. It uses a uniformly random rollout policy.

In a variety of applications, a variant of MCTS called *Upper Confidence Bounds for Trees* (UCT) [19] has shown excellent performance. UCT uses the UCB1 formula, originally developed for the multi-armed bandit problem [3], to select states in the tree and to trade off exploration and exploitation. In all experiments in this paper, a variant of UCT with the selection policy UCB1-TUNED is used. This policy takes the empirical variance of actions into account and has been shown to be empirically superior to UCB1 in several multi-armed bandit scenarios [3].

Described in the framework of policy iteration, there are two interacting processes within MCTS.

```
MCTS(startState) {
   bestResult ← -Infinity
   bestSolution ← {}
   for(numberOfIterations) {
      currentState ← startState
      solution ← {}
      while(currentState ∈ Tree) {
         currentState ← selectAction(currentState)
         solution ← solution + currentState
      }
      addToTree(currentState)
      while(simulationNotEnded) {
         currentState ← randomAction(currentState)
         solution ← solution + currentState
      }
      result = cumulativeReward(solution)
      forall(state ∈ solution) {
         state.value ← backPropagate(state.value, result)
      }
      if(result > bestResult) {
         bestResult ← result
         bestSolution ← solution
      }
   }
   return (bestResult, bestSolution)
}
```

Listing 1. MCTS with random rollout policy

Policy evaluation: In the backpropagation phase after each episode of experience, the return from that episode is used to update the value estimates of each visited state $s \in T$.

$$n_s \longleftarrow n_s + 1 \tag{1a}$$

$$\hat{V}^\pi(s) \longleftarrow \hat{V}^\pi(s) + \frac{r - \hat{V}^\pi(s)}{n_s} \tag{1b}$$

where n_s is the number of times state s has been traversed in all episodes so far, and r is the return received at the end of the current episode.

Policy improvement: During each episode, the policy adapts to the current value estimates. In case of a deterministic MDP and MCTS using UCB1-TUNED in the selection phase, and a uniformly random policy in the rollout phase, let

$$U^{\text{Var}}(s,a) = \left(\frac{1}{n_{s,a}}\sum_{t=1}^{n_{s,a}} r_{s,a,t}^2\right) - \left(\frac{1}{n_{s,a}}\sum_{t=1}^{n_{s,a}} r_{s,a,t}\right)^2 + \sqrt{\frac{2\ln n_s}{n_{s,a}}} \tag{2a}$$

be an upper confidence bound for the variance of action a in state s, where $n_{s,a}$ is the number of times action a has been chosen in state s in all episodes so far, and $r_{s,a,t}$ is the reward received when action a was chosen in state s for the t-th time. Let

$$U^{\text{Val}}(s,a) = \sqrt{\frac{2\ln(n_s)}{n_{s,a}}\min\left(\frac{1}{4}, U^{\text{Var}}(s,a)\right)} \tag{2b}$$

be an upper confidence bound for the value of action a in state s. Then, the policy of the MCTS agent is

$$\pi(s) = \begin{cases} \underset{a \in A(s)}{\operatorname{argmax}} \left(\hat{V}^\pi\big(P_a(s)\big) + C \times U^{\text{Val}}(s,a)\right) & \text{if } s \in T \\ \operatorname{random}(s) & \text{otherwise} \end{cases} \tag{2c}$$

where $P_a(s)$ is the state reached from position s with action a, $\operatorname{random}(s)$ chooses one of the actions available in s with uniform probability, and C is an exploration coefficient whose optimal value is domain-dependent.

3 RELATED WORK

Tesauro and Galperin [34] were the first to use Monte-Carlo rollouts for improving an MDP policy online. For each possible action a in the current state of the agent, they generated several rollouts starting with a and then following the policy as given by a "base controller" (an arbitrary heuristic). After estimating the expected reward of each action by averaging rollout results, they improved the heuristic by choosing and executing the action with the best estimated value. This resembles one cycle of policy iteration, focused on the current state.

Yan *et al.* [36] introduced the idea of online improvement of a base policy through *nested* search. The first level of nesting corresponds to a rollout strategy as proposed in [34], estimating the value of each action by starting with this action and then following the base policy. The second level estimates the value of each action by starting with this action and then executing a first-level search; higher levels are defined analogously. Bjarnason [5] improved this approach for Solitaire by using different heuristics and nesting levels for every phase of the game.

Cazenave [6, 7] proposed similar search methods to Yan's *iterated rollouts* under the names of *Reflexive Monte-Carlo Search* (RMCS) and *Nested Monte-Carlo Search* (NMCS). The main difference to preceding approaches is that RMCS and NMCS assume a uniformly random base policy instead of an informed search heuristic, and the best sequence found so far is kept in memory. NMCS has since been applied to a variety of problems, such as expression discovery [8], bus network regulation [9] and General Game Playing [23], and it has been improved for certain types of domains by adding the AMAF technique [1] and by re-introducing and optimizing base search heuristics [25].

Rosin [26] developed *Nested Rollout Policy Adaptation* (NRPA), a variant of NMCS that adapts the rollout policy during search using gradient ascent. At each level of the nested search, NRPA shifts the rollout policy towards the best solution found so far, instead of advancing towards this solution directly on the search tree. The algorithm depends on a domain-specific representation of actions that allows for the generalization of action values across different states.

In the context of MCTS, nested search has so far only been used for the preparation of opening books for the deterministic 2-player game of Go [2, 11, 12]. In these applications, nested search was performed offline to provide opening databases for the underlying online game playing agent. The different levels of search therefore used different tree search algorithms adapted to their respective purpose, and nested and regular MCTS have not been compared on the same task.

So far, no nested search algorithm has made use of the selectivity and exploration-exploitation control that MCTS provides. In this paper, we propose Nested Monte-Carlo Tree Search (NMCTS) as a general online planning algorithm for MDPs. We expect it to outperform MCTS in a similar way to how NMCS outperforms naive Monte-Carlo search—through nesting. Furthermore, we expect it to outperform NMCS in a similar way to how MCTS outperforms naive Monte-Carlo search—through selective tree search.

4 NESTED MONTE-CARLO TREE SEARCH

We define a level-0 *Nested Monte-Carlo Tree Search* (NMCTS) as a single rollout with the base rollout policy—either uniformly random, or guided by a simple heuristic. A level-1 NMCTS search corresponds to MCTS, employing level-0 searches as state evaluations. A level-n NMCTS search for $n \geq 2$ recursively utilizes the results of level-$(n-1)$ searches as evaluation returns.

```
NMCTS(startState, solution, level) {
   bestResult ← -Infinity
   bestSolution ← {}
   for(numberOfIterationsForLevel(level)) {
      currentState ← startState
      while(currentState ∈ Tree) {
         currentState ← selectAction(currentState)
         solution ← solution + currentState
      }
      addToTree(currentState)
      if(level = 1) {
         while(simulationNotEnded) {
            currentState ← randomAction(currentState)
            solution ← solution + currentState
         }
         result = cumulativeReward(solution)
      } else {
         (result,solution) =
            NMCTS(currentState, solution, level-1)
      }
      forall(state ∈ solution) {
         state.value ← backPropagate(state.value, result)
      }
      if(result > bestResult) {
         bestResult ← result
         bestSolution ← solution
      }
   }
   return (bestResult, bestSolution)
}
```

Listing 2. NMCTS with random rollout policy

Listing 2 shows pseudocode of NMCTS for deterministic environments, using a uniformly random rollout policy. It is called with an empty solution as argument on the highest nesting level. Finding the most effective trade-off between the numbers of samples at each level is subject to empirical optimization.

As the selection, expansion and backpropagation steps of MCTS are preserved in NMCTS, many successful techniques from MCTS research such as the UCB1-TUNED selection policy can be applied in NMCTS as well. Parameters can be tuned for each level of search independently.

In [28], it was found to be effective in SameGame not to spend the entire search time on the initial position of a problem, but to distribute it over all actions in the episode (or the first z actions). We call this technique *action-by-action search* as opposed to *global search*, and it is applicable at all levels of NMCTS. In case action-by-action search is used, a decision has to be made which action to choose and execute at each step of the search. Two possible options are a) choosing the most-sampled action—as traditionally done in MCTS—, or b) choosing the next action in the overall best solution found so far. Setting NMCTS to action-by-action search, using only one rollout per legal action in each action search, and then choosing the next action of the best known solution leads to NMCS as a special case of NMCTS. This special case does not provide for an exploration-exploitation tradeoff, nor does it build a tree going deeper than the number of nesting levels used, but it allows relatively deep nesting due to the low number of rollouts per search level.

5 EXPERIMENTAL RESULTS

We tested Nested Monte-Carlo Tree Search on three different deterministic, fully observable MDPs: The puzzles named "SameGame", "Clickomania" and "Bubble Breaker" [22, 28, 29, 30, 33]. These domains have identical transition functions, but different reward functions, resulting in different distributions of high-quality solutions. The decision problem associated with these optimization problems is NP-complete [4].

The rules of the puzzles are as follows. A two-dimensional board

or grid is filled with $M \times N$ tiles of C different colors, usually randomly distributed, at the start. Each action consists of selecting a group of two or more vertically or horizontally connected, identically-colored tiles. When the action is executed, the tiles of this group are removed from the board. If there are tiles above the deleted group, they fall down; if an entire column of the board is emptied of tiles, the columns to the right shift to the left to close the gap. An episode ends when no actions are left to the agent. The reward the agent receives depends on the specific variant of the puzzle:

Clickomania. The goal of this puzzle is to clear the board of tiles as far as possible. At the end of each episode, the agent receives a reward equivalent to the number of tiles removed.

Bubble Breaker. The goal of this puzzle is to create and then remove the largest possible groups of tiles. After each action removing a group of size `groupSize`, the agent receives a reward of $\texttt{groupSize}*(\texttt{groupSize}-1)$ points.

SameGame. In this puzzle, both the removal of large groups and the clearing of the board are rewarded. Each action removing a group of size `groupSize` results in a reward of $(\texttt{groupSize}-2)^2$ points; additionally, ending the episode by clearing the board completely is rewarded with an extra 1000 points. If the game ends without clearing the board, the agent receives a negative reward. It is computed by assuming that all remaining tiles of the same color are connected into virtual groups, and subtracting points for all colors according to the formula $(\texttt{groupSize}-2)^2$.

We compared regular MCTS and level-2 NMCTS in all three domains, using a random rollout policy. For SameGame, we also employed a state-of-the-art informed rollout policy, consisting of the TabuColorRandomPolicy [30] (setting a "tabu color" at the start of each rollout that is not chosen as long as groups of other colors are available) in combination with a multi-armed bandit learning the best-performing tabu color for the position at hand (based on UCB1-TUNED).

The experiments for Bubble Breaker and SameGame were conducted on the first 100 training positions used in [30][2]. These positions consist of 15×15 boards with randomly distributed tiles of 5 different colors. Algorithms were allocated 9120 seconds (about 2.5 hours) of computation time per position. The experiments on Clickomania were conducted using 100 randomly generated 20×20 boards with 10 different tile colors, to provide a greater challenge. Each algorithm here only ran for 1280 seconds per position.

As it has been shown for SameGame that restarting several short MCTS runs on the same problem can lead to better performance than a single, long run [30], we tested several numbers of randomized restarts for MCTS and tuned the selection policy for each of them. The same settings were then used for NMCTS, with the number of nested level-1 NMCTS searches equivalent to the number of restarts for multi-start MCTS. The exploration factor C of level 2 was set to 0 in all NMCTS conditions.

Fig. 1, 2 and 3 show that in Bubble Breaker and SameGame—in the latter using both random and informed rollouts—level-2 NMCTS significantly outperformed multi-start MCTS in all experimental conditions ($p<0.0001$ in a paired-samples, two-tailed t-test). The figures show both the performance of NMCTS and multi-start MCTS as well as the average difference in performance and the corresponding 95% confidence interval. The best results in SameGame were achieved building a level-2 tree out of 36,480 level-1 searches of 250 ms each, with informed base-level rollouts. In comparison to the best

[2] Available online at http://www.unimaas.nl/games/SameGame/TestSet.txt

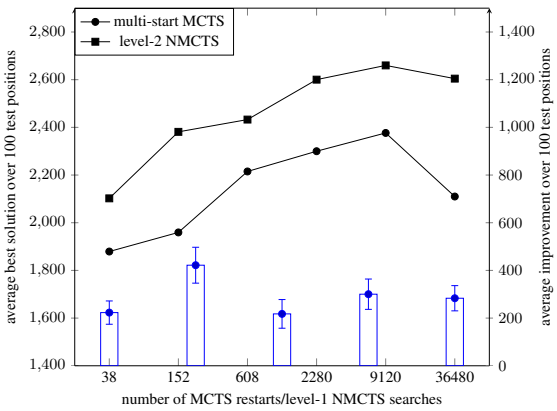

Figure 1. Performance of NMCTS in Bubble Breaker with random rollout policy. Bars show the average performance increase over multi-start MCTS with a 95% confidence interval.

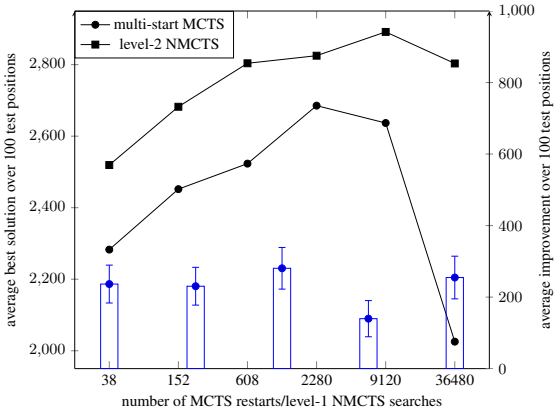

Figure 2. Performance of NMCTS in SameGame with random rollout policy. Bars show the average performance increase over multi-start MCTS with a 95% confidence interval.

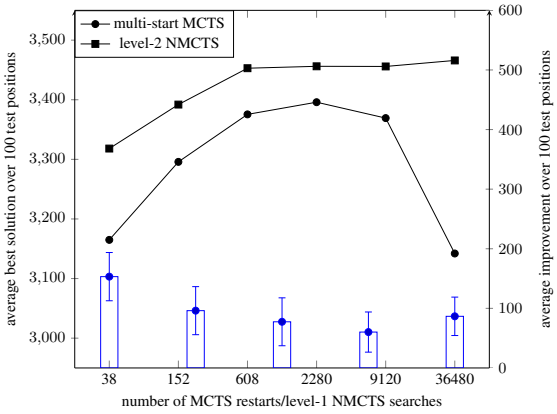

Figure 3. Performance of NMCTS in SameGame with informed rollout policy. Bars show the average performance increase over multi-start MCTS with a 95% confidence interval.

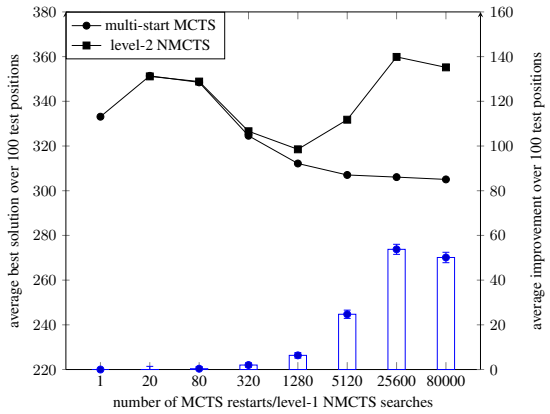

Figure 4. Performance of NMCTS in Clickomania with random rollout policy. Bars show the average performance increase over multi-start MCTS with a 95% confidence interval.

performance of multi-start MCTS, achieved with 2280 restarts of 4-second searches, the use of a nested tree increased the average best solution per position from 3395.9 to 3465.96. As a comparison, a doubling of the search time to 4560 restarts only resulted in a performance increase to 3431.0.

In Clickomania, level-2 NMCTS also achieved the highest score (see Fig. 4). While the results of multi-start MCTS for different numbers of restarts suggest that a single, global MCTS search could perform relatively well in Clickomania, memory limitations reduced the effectivity of this approach. NMCTS however is able to constantly reuse tree nodes of lower-level searches, and therefore does not suffer from this problem. We observed that the best-performing NMCTS setting tested used less than 15% memory of what a single, global MCTS search would have required for optimal performance.

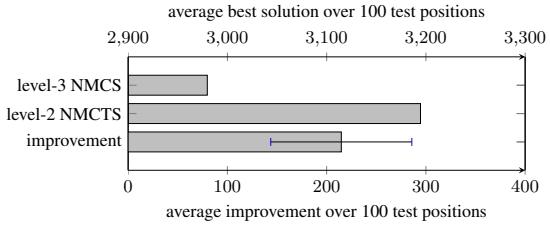

Figure 5. Performance of level-3 NMCS and level-2 NMCTS in SameGame with a random rollout policy. NMCS was stopped after 9120 seconds. NMCTS employs 2280 level-1 searches, each 100 milliseconds long, for each of the first 40 actions of an episode.

Fig. 5, 6 and 7 show a comparison of level-2 NMCTS to level-3 NMCS, including both the average results of the two algorithms as well as the average performance increase and corresponding 95% confidence interval. Here, NMCTS used action-by-action search on level 2, and advanced from action to action by choosing the next action of the best solution found so far. NMCS was not able to complete a level-3 search in the given time of 9120 seconds; consequently, the best solutions found after 9120 seconds were used for the comparisons. NMCTS outperformed NMCS in SameGame with random playouts ($p<0.0001$), SameGame with informed playouts ($p<0.01$)

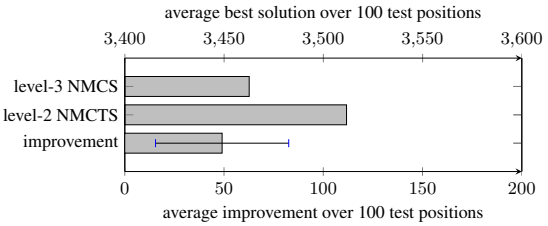

Figure 6. Performance of level-3 NMCS and level-2 NMCTS in SameGame with an informed rollout policy. NMCS was stopped after 9120 seconds. NMCTS employs 608 level-1 searches, each 750 milliseconds long, for each of the first 20 actions of an episode.

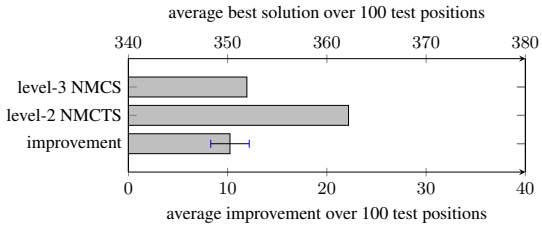

Figure 7. Performance of level-3 NMCS and level-2 NMCTS in Clickomania. NMCS was stopped after 9120 seconds. NMCTS employs 25600 level-1 searches, each 5 milliseconds long, for each of the first 10 actions of an episode.

and Clickomania ($p<0.0001$). For Bubble Breaker, manual testing has not revealed parameter settings superior to NMCS yet. Automatic parameter tuning is in preparation.

6 CONCLUSION AND FUTURE RESEARCH

In this paper, we proposed *Nested Monte-Carlo Tree Search* (NMCTS) as an online planning algorithm for large MDPs. Empirical results in the test domains of SameGame, Bubble Breaker and Clickomania show that NMCTS significantly outperforms regular Monte-Carlo Tree Search (MCTS). Experiments in SameGame and Clickomania suggest performance superior to Nested Monte-Carlo Search (NMCS). Since both MCTS and NMCS represent specific parameter settings of NMCTS, correct tuning of NMCTS has to lead to greater or equal success in any MDP domain.

Several promising directions remain for future research. First, in the experiments so far we have only used an exploration factor of 0 for level 2 of NMCTS. This means that the second level of tree search proceeded greedily in all experiments—it only made use of the selectivity of MCTS, but not of the exploration-exploitation tradeoff. Careful tuning of exploration at all search levels could lead to considerable performance improvements. Second, it appears that NMCTS is most effective in domains where multi-start MCTS outperforms a single, long MCTS run (like SameGame and BubbleBreaker), although its lower memory requirements can still represent an advantage in domains where multi-start MCTS is ineffective (like Clickomania). The differences between these classes of tasks remain to be characterized. Third, we plan to extend NMCTS to partially observable as well as stochastic MDPs.

ACKNOWLEDGEMENTS

This work is funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938.

REFERENCES

[1] H. Akiyama, K. Komiya, and Y. Kotani, 'Nested Monte-Carlo Search with AMAF Heuristic', in *Proceedings of the 2010 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pp. 172–176, (2010).

[2] P. Audouard, G.M.J.-B. Chaslot, J.-B. Hoock, J. Perez, A. Rimmel, and O. Teytaud, 'Grid Coevolution for Adaptive Simulations: Application to the Building of Opening Books in the Game of Go', in *Applications of Evolutionary Computing*, eds., M. Giacobini, A. Brabazon, S. Cagnoni, G. A. Di Caro, A. Ekárt, A. Esparcia-Alcázar, M. Farooq, A. Fink, P. Machado, J. McCormack, M. O'Neill, F. Neri, M. Preuss, F. Rothlauf, E. Tarantino, and S. Yang, pp. 323–332, (2009).

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer, 'Finite-Time Analysis of the Multiarmed Bandit Problem', *Machine Learning*, **47**(2-3), 235–256, (2002).

[4] T. C. Biedl, E. D. Demaine, M. L. Demaine, R. Fleischer, L. Jacobsen, and J. I. Munro, 'The Complexity of Clickomania', in *More Games of No Chance, Proceedings of the MSRI Workshop on Combinatorial Games*, ed., R. J. Nowakowski, pp. 389–404, (2002).

[5] R. Bjarnason, P. Tadepalli, and A. Fern, 'Searching Solitaire in Real Time', *ICGA Journal*, **30**(3), 131–142, (2007).

[6] T. Cazenave, 'Reflexive Monte-Carlo Search', in *Proceedings of Computer Games Workshop 2007*, eds., H. J. van den Herik, J. W. H. M. Uiterwijk, M. H. M. Winands, and M. P. D. Schadd, pp. 165–173, (2007).

[7] T. Cazenave, 'Nested Monte-Carlo Search', in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, ed., C. Boutilier, pp. 456–461, (2009).

[8] T. Cazenave, 'Nested Monte-Carlo Expression Discovery', in *Proceedings of the 19th European Conference on Artificial Intelligence*, eds., H. Coelho, R. Studer, and M. Wooldridge, pp. 1057–1058, (2010).

[9] T. Cazenave, F. Balbo, and S. Pinson, 'Using a Monte-Carlo Approach for Bus Regulation', in *12th International IEEE Conference on Intelligent Transportation Systems, 2009 (ITSC '09)*, pp. 1–6, (2009).

[10] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, 'Progressive Strategies for Monte-Carlo Tree Search', *New Mathematics and Natural Computation*, **4**(3), 343–357, (2008).

[11] G.M.J.-B. Chaslot, J.-B. Hoock, J. Perez, A. Rimmel, O. Teytaud, and M.H.M. Winands, 'Meta Monte-Carlo Tree Search for Automatic Opening Book Generation', in *Proceedings of IJCAI 2009 Workshop on General Intelligence in Game Playing Agents*, pp. 7–12, (2009).

[12] C.-W. Chou, P.-C. Chou, H. Doghmen, C.-S. Lee, T.-C. Su, F. Teytaud, O. Teytaud, H.-M. Wang, M.-H. Wang, L.-W. Wu, and S.-J. Yen, 'Towards a Solution of 7×7 Go with Meta-MCTS'. Advances in Computer Games, 13th International Conference (ACG 2011). In press.

[13] R. Coulom, 'Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search', in *5th International Conference on Computers and Games (CG 2006). Revised Papers*, eds., H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, volume 4630 of *Lecture Notes in Computer Science*, pp. 72–83. Springer, (2007).

[14] F. de Mesmay, A. Rimmel, Y. Voronenko, and M. Püschel, 'Bandit-Based Optimization on Graphs with Application to Library Performance Tuning', in *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009*, eds., A. P. Danyluk, L. Bottou, and M. L. Littman, pp. 729–736, (2009).

[15] H. Finnsson and Y. Björnsson, 'Simulation-Based Approach to General Game Playing', in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, eds., D. Fox and C. P. Gomes, pp. 259–264, (2008).

[16] S. Gelly and D. Silver, 'Achieving Master Level Play in 9 x 9 Computer Go', in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, eds., D. Fox and C. P. Gomes, pp. 1537–1540, (2008).

[17] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, 'Modification of UCT with Patterns in Monte-Carlo Go', Technical report, HAL - CCSd - CNRS, France, (2006).

[18] N. Ikehata and T. Ito, 'Monte-Carlo Tree Search in Ms. Pac-Man', in *2011 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 39–46, (2011).

[19] L. Kocsis and C. Szepesvári, 'Bandit Based Monte-Carlo Planning', in *17th European Conference on Machine Learning, ECML 2006*, eds., J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, volume 4212 of *Lecture Notes in Computer Science*, pp. 282–293. Springer, (2006).

[20] C.-S. Lee, M.-H. Wang, G. M. J.-B. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong, 'The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments', *IEEE Transactions on Computational Intelligence and AI in Games*, **1**(1), 73–89, (2009).

[21] R. J. Lorentz, 'Amazons Discover Monte-Carlo', in *Proceedings of the 6th International Conference on Computers and Games*, eds., H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, pp. 13–24, (2008).

[22] S. Matsumoto, N. Hirosue, K. Itonaga, K. Yokoo, and H. Futahashi, 'Evaluation of Simulation Strategy on Single-Player Monte-Carlo Tree Search and its Discussion for a Practical Scheduling Problem', in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2010*, volume 3, pp. 2086–2091, (2010).

[23] J. Méhat and T. 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).

[24] H. Nakhost and M. Müller, 'Monte-Carlo Exploration for Deterministic Planning', in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, ed., C. Boutilier, pp. 1766–1771, (2009).

[25] A. Rimmel, F. Teytaud, and T. Cazenave, 'Optimization of the Nested Monte-Carlo Algorithm on the Traveling Salesman Problem with Time Windows', in *Proceedings of Applications of Evolutionary Computation - EvoApplications 2011*, eds., C. Di Chio, A. Brabazon, G. A. Di Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, C. Prins, J. Romero, G. Squillero, E. Tarantino, A. Tettamanzi, N. Urquhart, and A. S. Etaner-Uyar, pp. 501–510, (2011).

[26] C. D. Rosin, 'Nested Rollout Policy Adaptation for Monte Carlo Tree Search', in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, ed., T. Walsh, pp. 649–654, (2011).

[27] A. Sabharwal and H. Samulowitz, 'Guiding Combinatorial Optimization with UCT', in *ICAPS 2011 Workshop on Monte-Carlo Tree Search: Theory and Applications*, (2011).

[28] M. P. D. Schadd, M. H. M. Winands, M. J. W. Tak, and J. W. H. M. Uiterwijk, 'Single-Player Monte-Carlo Tree Search for SameGame', *Knowledge-Based Systems. In press*.

[29] M. P. D. Schadd, M. H. M. Winands, H. J. van den Herik, and H. Aldewereld, 'Addressing NP-Complete Puzzles with Monte-Carlo Methods', in *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning*, volume 9, pp. 55–61, (2008).

[30] M. P. D. Schadd, M. H. M. Winands, H. J. van den Herik, G. M. J.-B. Chaslot, and J. W. H. M. Uiterwijk, 'Single-Player Monte-Carlo Tree Search', in *Proceedings of the 6th International Conference on Computers and Games*, eds., H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, pp. 1–12, (2008).

[31] D. Silver and J. Veness, 'Monte-Carlo Planning in Large POMDPs', in *Advances in Neural Information Processing Systems 23, NIPS 2010*, eds., J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, pp. 2164–2172, (2010).

[32] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[33] F. W. Takes and W. A. Kosters, 'Solving SameGame and its Chessboard Variant', in *Proceedings of the 21st Benelux Conference on Artificial Intelligence, BNAIC 2009*, eds., T. Calders, K. Tuyls, and M. Pechenizkiy, pp. 249–256, (2009).

[34] G. Tesauro and G. R. Galperin, 'On-line Policy Improvement using Monte-Carlo Search', in *Advances in Neural Information Processing Systems 9, NIPS 1996*, eds., M. Mozer, M. I. Jordan, and T. Petsche, pp. 1068–1074, (1997).

[35] M. H. M. Winands, Y. Björnsson, and J.-T. Saito, 'Monte Carlo Tree Search in Lines of Action', *IEEE Transactions on Computational Intelligence and AI in Games*, **2**(4), 239–250, (2010).

[36] X. Yan, P. Diaconis, P. Rusmevichientong, and B. Van Roy, 'Solitaire: Man Versus Machine', in *Advances in Neural Information Processing Systems 17, NIPS 2004*, eds., L. K. Saul, Y. Weiss, and L. Bottou, (2004).