Simulation-based Search of Combinatorial Games

Łukasz Lew LEW@MIMUW.EDU.PL
University of Warsaw, Poland

Rémi Coulom REMI.COULOM@UNIV-LILLE3.FR
Université de Lille, INRIA, CNRS, France

Abstract

Monte-Carlo Tree Search is a very successful game playing algorithm. Unfortunately it suffers from the horizon effect: some important tactical sequences may be delayed beyond the depth of the search tree, causing evaluation errors. Temporal-difference search with a function approximation is a method that was proposed to overcome these weaknesses, by adaptively changing the simulation policy outside the tree.

In this paper we present an experimental evidence demonstrating that a temporal difference search may fail to find an optimal policy, even in very simple game positions. Classical temporal difference algorithms try to evaluate a local situation with a numerical value, but, as it appears, a single number is not enough to model the dynamics of a partial two-player game state.

As a solution we propose to replace numerical values by approximate thermographs. With this richer representation of partial states, reinforcement-learning algorithms converge and accurately represent dynamics of states, allowing to find an optimal policy.

1. Introduction

The game of Go is a great challenge for artificial intelligence. Despite a recent progress, the best Go-playing programs are not stronger than serious amateurs, and immensely weaker than professional players. The game of Go remains an exciting source of inspiration for new algorithms.

Appearing in *Proceedings of the ICML Workshop on Machine Learning and Games*.

The best Go-playing programs are currently based on the Monte-Carlo Tree Search (MCTS) approach. This method was introduced in 2006 (Coulom, 2006; Kocsis & Szepesvári, 2006; Gelly et al., 2006), and allowed programs to improve from a weak amateur level to a strong amateur. Unlike traditional search algorithms based on the min-max search combined with a static evaluation function, Monte-Carlo programs evaluate positions by averaging the final outcomes of many random continuations. This principle fits well with the dynamic nature of the game of Go, and allowed the new generation of Monte-Carlo programs to outperform classical programs significantly.

Although Monte-Carlo techniques resulted in a huge improvement, current algorithms are still based on a tree search and suffer severely from the horizon effect. With hundreds of legal moves, the game of Go has a huge branching factor. During the playout, after exiting the tree, even if a position seems calm there are usually plenty of moves that if played need an immediate response to avoid a big loss. This notion of a threat and a response is crucial for a good quality of playout results. It is partially covered in simulations by a fixed playout policy based on an expert knowledge. Unfortunately many simple tactics are not covered and a fixed policy consistently makes mistakes.

The horizon effect can be overcome by human players by decomposing the analysis of a position into independent sub-problems. Most of the situations on the board may be searched locally, and the global analysis of the situation can be obtained by combining those local searches. Decomposing the global search into small local searches can considerably reduce the complexity.

The problem of decomposing a big global search into smaller local searches was identified very long ago. A mathematical foundation for this method was established with Combinatorial Game Theory (Conway, 1976). Application of this theory to the game of Go allowed to solve some endgames positions complexity

of which is far beyond the reach of traditional search algorithms (Berlekamp & Wolfe, 1994; Müller, 1999). Besides formal solutions of endgames, most of so-called "classical" Go programs, such as GNU Go, combined local searches of sub-goals into a heuristic global analysis of a position.

In order to apply decomposition ideas to Monte-Carlo Tree Search, many programmers suggested using results of local searches to adaptively modify playouts to the current board situation. The tree-search part (moves within the game tree) of MCTS algorithm already adapts the playout policy by focusing on better moves. Adapting the playout policy outside of the tree is necessary to overcome the horizon effect.

Adaptive playouts are widely considered as a very promising idea by authors of Go programs, but very few effective algorithms are known. A variety of reinforcement-learning algorithms were proposed (Silver et al., 2007; Silver et al., 2008; Silver, 2009). This research produced a Go program, RLGO, that does not perform any tree search and can compete with classical programs. But RLGO is still much weaker than the state-of-the-art Monte-Carlo programs that use a static playout policy.

This paper presents new ideas on applying a temporal-difference search to combinatorial games. Experiments with combinatorial games modeling simple Go position show that traditional methods for the state-value approximation can fail even in very simple situations. An alternative approach based on the approximate thermography is proposed. Experimental results demonstrate that approximate thermographs can be used in a reinforcement-learning framework, and can overcome the difficulties of traditional state values.

2. A Simple Artificial Problem

Algorithms are tested on a simple artificial sum of games. Such a problem is depicted in Figure 1. Figure 2 shows an equivalent Go position. This board can be separated into three completely independent areas. This is an a simple situation that does not represent the complexity of the game of Go. However, this situation is complex enough to be a challenge for adaptive-playout Monte-Carlo algorithms.

For such a simple problem, it is easy to determine an optimal strategy. The total number of possible states in this sum of games is equal to the product of the number of the states in each game, that is to say $3 \times 3 \times 5 = 45$. The optimal strategy consists in playing in the games *H-I-I-G*, in a sequence. This yields a score of $4 + 0 - 2 = 2$ for Left.

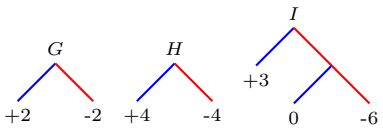


Figure 1. A simple sum of three combinatorial games (G, H, and I), each being represented by a tree. Two players, Left and Right, alternatively make a move. The game starts in the root position for all the components. A move consists of choosing a game and moving from its current position to the left or right child, depending on whose turn it is to play. It is not possible to play in a game in which current state is a leaf. The game is over when all the components have reached a leaf. The final score for Left is the sum of the leaf values. For instance, if Left starts by playing in the game I, then it would reach the leaf $+3$. Then Right could play in the game H, and reach the leaf -4. Then the only move for Left would be to $+2$, in the game G. The final score of this game would be $3 - 4 + 2 = 1$. The players don't know the value of each leaf. They are only given the final score of the game.

A key aspect of this problem is a notion of a threat. When playing in game I, Right does not score a lot of points, but makes a threat. Left should answer immediately or else it would lose 6 points. Go players call this *sente* and it is an essential element of a Go strategy.

3. Temporal-Difference Search

The simplest approach to reduce the complexity of a sum of games consists of approximating its state value by the sum of local state values. In our simple example, this would mean estimating $3 + 3 + 5 = 11$ parameters, one for each node, instead of having one parameter for each of the $3 \times 3 \times 5 = 45$ global states. The complexity of this method for approximating the value function is equal to the sum of the game complexities, instead of their product. When a number of games is high, this decomposition can produce a huge simplification.

It is possible to estimate the parameters of this simplified state-value approximation with a temporal-difference search (Silver, 2009). Temporal-difference search consists of adjusting evaluation parameters using the simulated playouts and their results.

Algorithm 1 shows a simple form of a temporal-difference search based on the TD(λ) algorithm. Parameters of this algorithm are N (the epoch length), ϵ (the exploration coefficient), α (the learning rate), and λ (the decay rate of the eligibility traces). The parameters of the algorithms were $N = 1000$, $\epsilon = 0.2$, $\alpha = 0.001$, $\lambda = 0.9$.

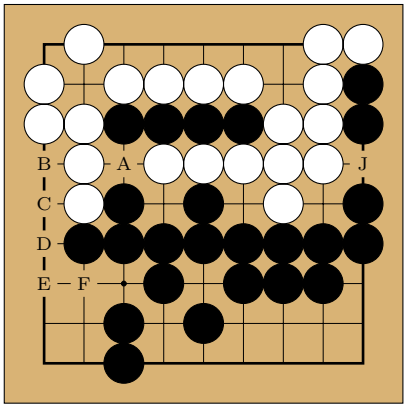


Figure 2. This Go board is similar to the abstract sum of the games of Figure 1. Black is Left, and White is Right. A territory scoring (Japanese style) is assumed. The leaves of the abstract games correspond to the following move sequences: Leaf $+2$ in the game G is ●J; -2 is ○J; $+4$ is ●A; -4 is ○A. The game I is a little more complicated. $+3$ is the sequence ●C-○B-●D. The move of Right at the root of I is ○D-●E-○C. The leaf 0 is reached after ●F. Leaf -6 is ○F.

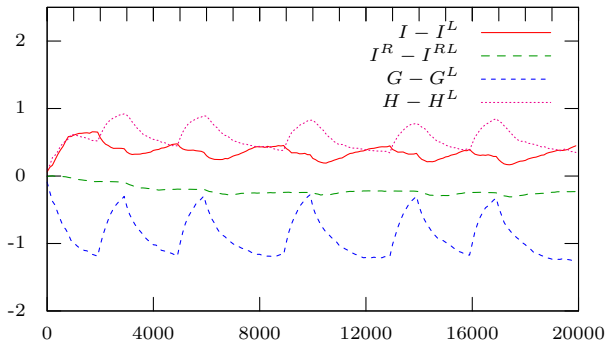


Figure 3. The change(delta) of the value function (an incentive) for different moves of Left player. The greedy policy chooses a move with a highest value.

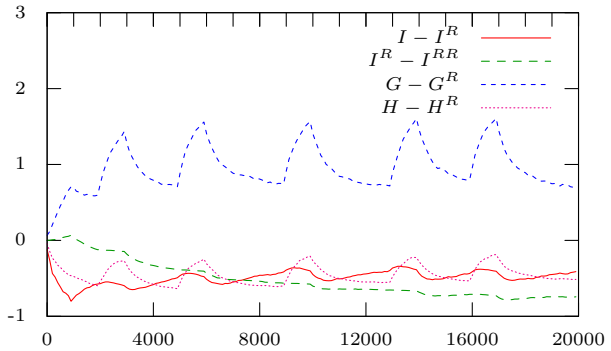


Figure 4. The change (delta) of the value function (an incentive) for different moves of Right player. The greedy policy chooses a move with a lowest value.

Algorithm 1 TD(λ) Policy Iteration

$\vec{\theta} \leftarrow 0$ {Reset the parameters}
$\vec{\theta}_\pi \leftarrow \vec{\theta}$
loop
  for $n = 1$ to N **do**
    $\vec{e} \leftarrow 0$ {Clear the eligibility traces}
    $s_0 \leftarrow$ an initial state
    $i \leftarrow 0$
    while s_i is not terminal **do**
      if random() $< \epsilon$ **then**
        $\vec{e} \leftarrow 0$
        $a_i \leftarrow$ random action
      else
        {a greedy action}
        $\vec{e} \leftarrow \lambda\vec{e} + \partial\hat{V}(\vec{\theta}, s_i)/\partial\vec{\theta}$
        $a_i \leftarrow \arg\max_a(\hat{V}(\vec{\theta}_\pi, s_i.\text{play}(a)))$
      end if
      $s_{i+1} \leftarrow s_i.\text{play}(a_i)$
      $\delta \leftarrow \hat{V}(\vec{\theta}, s_{i+1}) - \hat{V}(\vec{\theta}, s_i)$
      $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
      $i \leftarrow i + 1$
    end while
  end for
  $\vec{\theta}_\pi \leftarrow \vec{\theta}$ {the policy update}
end loop

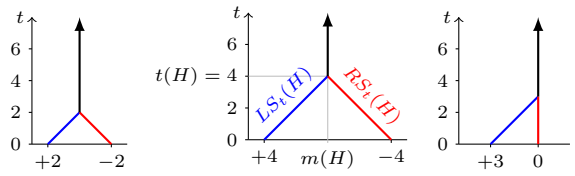
Figures 3 and 4 show the result of applying Algorithm 1 to the sum of combinatorial games. Many other parameter values were tried, on a variety of combinatorial games, and oscillations were observed most of the time.

This result demonstrates that temporal-difference search fails to converge to any policy when the state value is approximated by the sum of the values of independent games.

4. Combinatorial Game Theory

A classical value function assigns to each state only a single number – an expected reward. Such a representation must necessarily depend on a policy and therefore represents the value of one policy only.

The previous section shows that even simple state spaces arising in the game of Go (or, more generally, in the combinatorial games) may lead to oscillation problems in classical Temporal Difference learning. To overcome this problem, a knowledge accumulated from a simulated experience should be independent of a policy used. To achieve this while simultaneously benefiting from a divide-and-conquer approach, we need a representation of components that is richer than a single number, and can reflect dynamics of states.



(a) Game G (b) Game H (c) Game I

Figure 5. The thermographs of G, H, and I. Thermographs of *gote* games like G and H are symmetric triangles. Thermographs of *sente* games like I are asymmetric.

Combinatorial games can be represented quite accurately using *thermographs*. Next section explains what a thermograph is and how it can be recursively built.

5. Thermographs

The simplest approximation to represent a combinatorial game would be two numbers - a left stop and a right stop. Stops are results of the game if left or right player plays the first move and an alternating optimal play follows. For instance on Figure 1: $LS(I) = 3$ and $RS(I) = LS(I^R) = 0$ (I^R (resp. I^L) denotes what remains of I after Right (resp. Left) has played a move in the game I).

Stops of games would be sufficient if after one player starts his opponent would always respond in the same game or pass if there is no response. But in a combinatorial game sum it may happen that one player plays two or more moves in a row in the same game while his opponent plays elsewhere.

To model this we assume that there exists an environment consisting of many simple games and the player may choose to either play a move in a game G or to play it in the environment and get t points. This is equivalent to giving t points of "tax" (A value t is also called *ambient temperature* or value of *tenuki*.) to the opponent with each move. Taxed stops $LS_t(G)$ and $RS_t(G)$ are the results of a taxed game under optimal play when Left or respectively Right player plays first.

Note that if the tax is too high then neither player will be willing to play first in the game (or equivalently, when the ambient temperature is high enough, both players will prefer to play in the environment.) They will wait until the tax is low enough. Such point is called the *temperature* of the game: $t(G)$ and corresponds to the notion of the urgency of the play.

When a current tax is equal to the temperature of the game, then both stops are equal. This value is called the mean of the game: $m(G) = LS_{t(G)}(G) = RS_{t(G)}(G)$. Because waiting for the tax to be low enough is an optimal strategy, we have: $LS_t(G) = RS_t(G) = m(G)$ for all $t >= t(G)$.

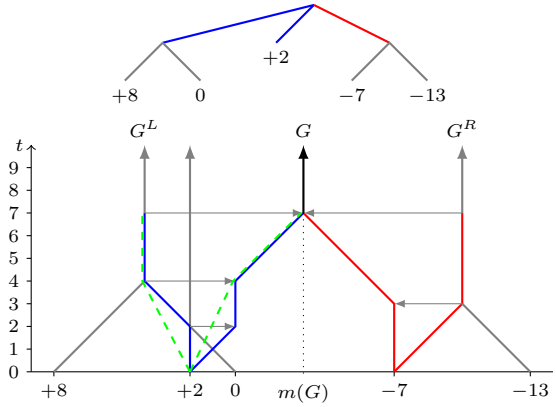


Figure 6. A simple combinatorial game and a recursive calculation of its thermograph. The green dashed line shows an approximation used in CG-TD(0) algorithm. Horizontal gray arrows represent taxing.

A *thermograph* is a graphical plot of stop functions. On the horizontal axis is the result of the game with positive values on the left. On the vertical axis is the tax (or the ambient temperature).

It is easy to recursively compute thermographs. If G^L and G^R are the sets of the left and the right options of G then:

$$LS_t(\{G^L|G^R\}) = \max(RS_t(G^L) - t, m(G)) \ , \quad (1)$$

$$RS_t(\{G^L|G^R\}) = \min(LS_t(G^R) + t, m(G)) \ , \quad (2)$$

and $(t(G), m(G))$ are the coordinates of the intersection of $RS_t(G^L) - t$ and $LS_t(G^R) + t$. Note that the maximum in $\max(RS_t(G^L) - t, m(G))$ goes over the right stops of all the options in G^L and $m(G)$.

An example of the thermographs and the process of the recursive computation is graphically shown on Figure 6.

Given the thermographs, HOTSTRAT strategy (play in a hottest game) is close to optimal. Other strategies like SENTESTRAT or THERMOSTRAT can be found in (Berlekamp et al., 1982).

Cazenave (2002) gives an empirical comparison of different strategies on simple combinatorial games.

6. Approximation Of Thermographs

Thermographs of big games can be quite complex - they can have many corners. We propose a triangle approximation. Each component will be represented with four numbers: temperature $t(G)$, mean $m(G)$, left stop and right stops $LS(G) = LS_0(G)$, $RS(G) = RS_0(G)$.

Left and right taxed stops are assumed to be vertical masts above $(t(G), m(G))$, and straight lines below going through $(0, LS(G))$ and $(0, RS(G))$ respectively.

Given such approximations of all options (children) it is straightforward to compute a temperature and a mean of a parent. An approximation process is illustrated in Figure 6. Firstly, an approximation of a right stop function of all left options must be done using Equation 1. In the case of approximated thermographs, it is sufficient to find $\max(RS(G^L))$, and a mean and a temperature of a game with the highest mean. A resulting approximated right stop function, and the same stop function taxed are marked with a green dashed line. Next the same steps must be repeated for G^R. Intersection of the two approximated stops will yield the mean and the temperature of G.

7. Combinatorial Games Temporal Difference Learning.

Algorithm 2 CG-TD(0)

while not converged **do**
  $s \leftarrow (s[1], s[2], ..., s[N])$ {a sum of games}
  while $\exists i : s[i]$ has options **do**
    $i \leftarrow \pi(s)$ {choose a component of the sum}
    $G \leftarrow s[i]$
    $H \leftarrow \pi(G)$ {choose an option of G to play}
    $s[i] \leftarrow H$ {update CG components}
    {calculate the update direction of G based on the left options}
    {find the option with the highest mean}
    $L \leftarrow \arg\max_{L \in G^L}(m(L))$
    {find the highest right stop}
    $lrs \leftarrow \max(RS(G^L))$ {G^L right stop}
    $rs \leftarrow m(G) - t(G)$
    {intersection with the taxed mast of the left options}
    $t_1 \leftarrow \frac{m(L)-rs}{2}$
    {intersection with the taxed right stop of the left options}
    $t_2 \leftarrow t(L)\frac{lrs-rs}{lrs-(m(L)-2\cdot t(L))}$
    $t' \leftarrow \min(t_1, t_2)$ {first intersection}
    $m' \leftarrow rs + t'$ {the mean of the first intersection}
    {update the thermograph of G}
    $m(G) \leftarrow m(G) + \alpha(m' - m(G))$
    $t(G) \leftarrow t(G) + \alpha(t' - t(G))$
    $LS(G) \leftarrow t(G) + \alpha(lrs - LS(G))$
    {calculate update direction of G based on the right options}
    ... {analogous code}
    {update the thermograph of G}
    ... {analogous code}
  end while
  {Update the thermographs of the leaves}
  $\delta \leftarrow$Result$(s) - \sum_{i=1}^{N} m(s[i])$ {prediction error}
  for $i = 1$ to N **do**
    $t(s[i]) \leftarrow 0.0$
    $m(s[i]) \leftarrow m(s[i]) - \frac{\alpha\delta}{N}$
    $LS(s[i]) \leftarrow m(s[i])$
    $RS(s[i]) \leftarrow m(s[i])$
  end for
end while

The combinatorial-games temporal-difference learning (*CG-TD(0)*) algorithm uses an approximation scheme presented in the previous section.

With each node of each component of the game sum, this algorithm maintains an approximated thermograph represented with four numbers: mean, temperature, left and right stops. A state is a position in the game trees of all the components. The starting state consists of the roots of all the components. State is final if none of the components has options (children).

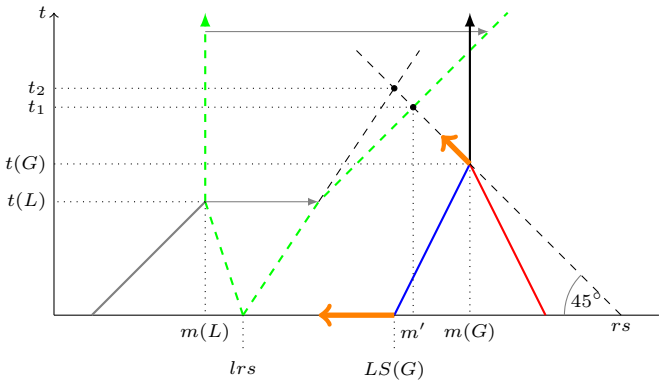


Figure 7. Update algorithm. A whole left stop (and the mean-temperature point) of the approximate thermograph G is moved in the direction of the big arrows to improve the consistency with the thermograph L as described on Figure 6 .

A next state is chosen according to the policy π: firstly, the component is chosen, then the component's option. The state is then updated by replacing the chosen component by its chosen option (a child in the tree). At this point the component G that just disappeared will be updated according to the approximate thermography described in the previous section.

The algorithm is easiest understood by looking at Figure 7. The taxed line approximating all the left options should match the left stop function of the game G - $LS_t(G)$ - like in Figure 6. If it doesn't the algorithm adjusts the left stop $LS(G)$, the mean $m(G)$ and the temperature $t(G)$ in the direction of the correct position on the taxed line. The amount of adjustment is controlled by the learning rate. Next, a similar update is done for the right options (not shown on Figure 7).

Leaf nodes are updated using a classical gradient descent to match the result. They will converge to the correct (up to a global constant) values as long as the policy will visit enough variety of the final states (the combinations of the component's leaf nodes). This is sufficient for interior nodes to converge as long as all the children are visited infinitely many times.

The exact equations given in Algorithm 2 are provided to help with the implementation. They are obtained using geometric properties of the thermographs on Fig-

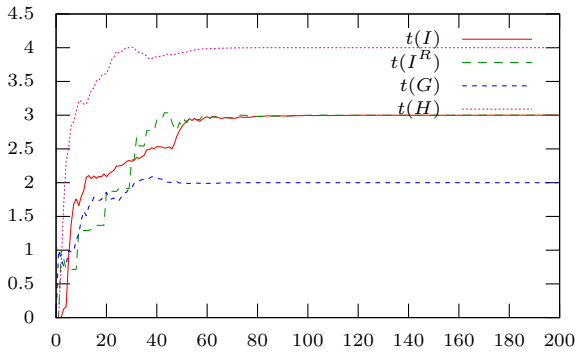


Figure 8. Temperatures

ure 7. The figure alone should be enough to understand the algorithm.

Figure 8 shows very fast convergence of the temperatures to the correct values yielding near-optimal HOTSTRAT (play in hottest game) policy.

8. Conclusion

We have shown that TD(λ) with a linear function approximation may be unable to find an optimal policy because a single number is not sufficient to represent the value of a local game. To cover that weakness we designed a richer representation of state value based on Combinatorial Game Theory. We designed a thermograph approximation which allows to create simple reinforcement learning algorithms converging to correct approximate thermographs.

A correct temperature estimate allows us to find a policy close to optimal. Applying this idea to Monte-Carlo Go could produce adaptive playouts, covering many weaknesses of existing expert-based static playout policies.

This algorithm could be further extended to cover zugzwang (situations where neither player wants to play first) by allowing players to pass when the temperature is negative, and finishing game early. This would cover situations like *seki* in the game of Go.

Acknowledgments

This work was supported in part by the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886. This work was supported in part by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the "CPER 2007–2013". This publication only reflects the authors' views.

References

Berlekamp, E., & Wolfe, D. (1994). *Mathematical Go: Chilling gets the last point.* A K Peters.

Berlekamp, E. R., Conway, J. H., & Guy, R. K. (1982). *Winning ways.* A K Peters.

Cazenave, T. (2002). Comparative evaluation of strategies based on the values of direct threats. *Board Games in Academia V.* Barcelona.

Conway, J. H. (1976). *On numbers and games.* London: Academic Press.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. *Proceedings of the 5th International Conference on Computer and Games* (pp. 72–83). Turin, Italy: Springer.

Gelly, S., Wang, Y., Munos, R., & Teytaud, O. (2006). *Modification of UCT with patterns in Monte-Carlo Go* (Technical Report RR-6062). INRIA.

Kocsis, L., & Szepesvári, C. (2006). Bandit-based Monte-Carlo planning. *Proceedings of the 15th European Conference on Machine Learning* (pp. 282–293). Berlin, Germany: Springer.

Müller, M. (1999). Decomposition search: A combinatorial games approach to game tree search, with applications to solving Go endgames. *16th International Joint Conference on Articial Intelligence* (pp. 578–583). Stokholm, Sweden.

Silver, D. (2009). *Reinforcement learning and simulation-based search in computer Go.* Doctoral dissertation, Department of Computing Science, University of Alberta, Canada.

Silver, D., Sutton, R., & Müller, M. (2007). Reinforcement learning of local shape in the game of Go. *20th International Joint Conference on Articial Intelligence* (pp. 1053–1058).

Silver, D., Sutton, R., & Müller, M. (2008). Sample-based learning and search with permanent and transient memories. *Proceedings of the 25th International Conference on Machine Learning* (pp. 968–975). Helsinki, Finland.