Monte Carlo Temperature Discovery Search for Combinatorial Games

Philipp Hennig
Cambridge, 13 June 2008
1 Introduction

1.1 Overview

Board games pose a viable challenge for Machine Learning methods. In the last two years, a Reinforcement Learning technique which selectively expands promising parts of the search tree based on the results of random roll-outs, called UCT, has generated much celebrated results for the game of Go on a small board. On larger boards, however, this method has to rely on massively parallel computer architectures to be able to compete with trained humans. Since Go games are often divisible into nearly independent sub-games, it is tempting to consider a divide-and-conquer approach to the problem of larger boards. Combinatorial Game Theory provides a mathematical framework (and strategies with bounded errors on perfect play) for games that are separable in such a way. The concept of a game Temperature is central to this theory. Unfortunately, Temperatures are very hard to calculate exactly.

This work reports on an attempt to measure game Temperatures approximately for arbitrary combinatorial games, using both UCT and a Bayesian extension to it. The method is complemented by an adaptive update rule for the search parameters and a simple likelihood model for the relationship between search results and true Temperatures. At this point, there there are only preliminary results to report. Our Algorithm wins against UCT in simple game play, but Temperature estimates in mildly complex games are rather coarse. In the light of these results, we also assessed potential weaknesses of the class of greedy reinforcement learning techniques on trees, to which both UCT and our searcher belong.

The results presented here, where not explicitly marked by the use of the first person singular, are the product of collaborative work between Thore Graepel and David Stern\(^1\) and the author of this report.

1.2 Machine Learning for Games

Two player, full-information games, i.e. games that can be represented by a bi-partite MIN/MAX-tree that is available – at least in principle – in its entirety to both players at any time, like Chess and Go, are an established and still testing application for machine learning methods [1, 2, 3, 4]. They are a “convenient” domain from an experimental point of view, as they can often be implemented in software with little effort, eliminating experimental noise. Their special structure poses peculiar challenges to a learning algorithm:

- For reasonably complex games, the search space is very large (Go’s game tree has about \(10^{400}\) nodes [5]).
- At the same time, the “reward” (the outcome of the game) conveys very little information (often just one bit).
- The problem of finding a good line of play is highly non-continuous: Small changes in both the state of the game (the board situation) and action of either player (the moves taken) can drastically change the outcome of the game.

\(^1\)the former two are at Microsoft Research Ltd., Roger Needham Building, J.J. Thomson Avenue, Cambridge, UK
Their adversarial nature causes subtle problems: For example, reinforcement learning machines that learn by playing against themselves or a weak opponent can get caught up in local optima far from the true MIN/MAX-solution.

Within the Machine Learning community, games have been approached from two different directions that make use of two different kinds of data: Supervised learning methods try to find patterns in recorded human games (e.g. [6]), or in the optimal lines of play found by some (typically slow) exhaustive search method (e.g. [7]). Reinforcement Learning programs, on the other hand, learn directly from play, either against another instance of themselves or a second entity, be it human or another machine (e.g [1, 8]). Since machines surpassed human abilities in Chess [9] (using mostly brute-force exhaustive search), interest has shifted to the Asian board game of Go. In contrast to Chess, in Go, heuristic evaluation of the “value” of a particular in-game position is very difficult, as the value of each stone depends in complex ways on the position of the other stones on the board. Since the game tree for Go also has hundreds of orders of magnitude more nodes than that for chess, the consensus within the community is that exact, exhaustive methods can not be applied to Go, and Machine Learning methods can be the only way forward. With Go occupying such a central spot in the literature, it is obviously tempting to build algorithms specifically for this domain. This work, however, will stay at the more general level of Combinatorial Games (see section 2.2). Most of the experiments use a game called Amazons for technical reasons (there exists an analytical solution providing a gold standard), but the results should be transferable to any combinatorial game, and to a certain extent (see section 5) to Go.

1.3 Structure of this text

Section 2 will give a brief overview over the two quite disjoint sets of knowledge (greedy Reinforcement Learning on trees and Combinatorial Game Theory) used in this work. Section 3 will describe the newly developed methods in detail. Section 4 presents experimental results. Section 5 contains an assessment of the preliminary results and an investigation of two potential weaknesses of the greedy algorithms presented here. Finally, sections 6 and 7 wrap up some lessons learned, and give an outlook on future work.

2 Previous Work

2.1 Monte Carlo reinforcement learning on Game trees

Recently, a Monte Carlo approach to game tree search known as the Upper Confidence Bound method for Trees (UCT) [10, 11] has had impressive success in Go, winning against a human master player on a 9 × 9 board earlier this year [12]. However, on the full 19 × 19 board, UCT necessitates the use of...
large computer clusters. Even then, professional human players have kept a competitive edge over machines so far. The UCT algorithm is based on a large number of descents from the root of the game tree (the current board position) to random terminal nodes. During each descent, board situations already encountered during previous descents are treated as a local multi-armed bandit problem \[13\]: At a given node \(i\) in the tree, which has been seen \(n_i\) times before, UCT chooses to follow the available move \(j\) which maximises the score

\[
\frac{1}{T_j(n_i)} \sum_{s=1}^{T_j(n_i)} X_{j,s}^i + \sqrt{\frac{2 \log n_i}{T_j(n_i)}}
\]

where \(T_j(n_i)\) is the number of times \(j\) was played during the previous \(n_i\) descents and \(X_{j,s}^i\) are the rewards gained from playing child \(j\) in those descents. This score strikes a balance between exploration and exploitation – broadening and deepening the search tree. Once a previously unseen node is reached (\(n_i = 0\)), UCT plays a random roll-out until it finds a leaf node, whose value is taken as the reward \(X\) for all ancestors with \(n > 0\) in this descent. If only some children of a node have been explored so far, a simple rule is to enforce exploration of all unexplored nodes before any node is exploited twice.\(^3\)

For a “tree” of depth 1, with \(K\) independent children at the root node, which have true expected rewards of \(\mu_j\) – the classic multi-armed bandit problem – this algorithm is known as UCB [10]. If the rewards \(X_{i,s}\) are i.i.d., it is unbiased. Compared to an optimal algorithm that always plays the optimal lever \(j^*\), which has an expected reward of \(\mu^*\), UCB’s regret (the difference between the reward achieved by UCB and optimal play) is at most

\[
8 \sum_{i: \mu_i < \mu^*} \left( \frac{\ln n}{\mu^* - \mu_i} \right) + \left( 1 + \frac{\pi^2}{3} \right) \left( \sum_{j=1}^{K} \mu^* - \mu_j \right)
\]

So, for large \(n\), the number of times the optimal node gets chosen grows as \(O(e^n)\) relative to the number of times any other node gets chosen. For a proper tree of depth larger 1, the rewards received from a child \(j\) of node \(i\) after after \(t\) roll-outs, \(X_{j,t}^i\), are neither identically distributed nor independent \[14\], because of the “shifting” of the roll-out policy underneath \(j\) during the learning process, so the UCT score of \(j\) is biased from the true minimax value \(\mu_j^*\) of this child by

\[
\delta_{j,t}^i = \left| \mu_j^* - \frac{1}{t} \sum_{s=1}^{t} X_{j,s}^i \right|
\]

It can be shown \[14\] that this bias is bounded by

\[
\delta_{j,t}^i \leq K D_j \ln \frac{t}{l}
\]

where \(K < 1\) is a constant and \(D_j\) is the depth of this child in the tree (measured from the root). This corresponds to the intuitive statement that nodes deeper

\[\text{[It is also possible to use prior knowledge from various sources to decide whether an unknown node should be explored or not [14]. To keep the results as general as possible, we did not make use of such knowledge.]}\]
in the tree are less biased, because they are closer to the leaves (which have unbiased, deterministic values), and that the decision taken at the root node is the most biased one. See 5 for a closer inspection of this behavior.

Mogo [12], the best contemporary Go machine, makes extensive use of UCT (with several, often domain-specific optimizations). On $19 \times 19$ boards, it is capable of performing about 10,000 descents into the tree per second and CPU core on contemporary hardware. Unfortunately, this is still not enough to achieve human play performance, so parallelization has been a major concern. Because UCT is growing one coherent search tree, it has to have access to the roll-out statistics of all expanded nodes – the whole search tree – at any point in time, making parallelization cumbersome. One option is for each machine in a cluster to grow its own search tree and synchronize the trees at regular intervals, but this causes obvious efficiency issues.

Good human Go players are aware that a Go match consists, for a large part of its duration, of sub-parts (“battles”), whose value (the number of points at stake) are approximately independent of each other. So it is tempting to consider dividing a Go board situation up into these sub-games and solving each of them individually. However, while the values of the battles on a board are often independent of each other, the optimal line of play shifts back and forth between them in a very non-obvious manner, which has little to do with the value of a battle as such (i.e. game action is not necessarily in the biggest battle).

### 2.2 Combinatorial Game Theory

The mathematical structure underlying this complex path of optimal play from one sub-game to the other has been formalized and extensively studied under the name **Combinatorial Game Theory (CGT)** [15]. CGT is applicable to all round-based, two-player, full information games with integer scores for terminal positions, but it is most useful for games that can be interpreted as the sum of independent sub-games in the sense that at each point in the game, the player whose turn it is to play chooses a sub-game to play in and makes his move there. Under the notation conventions of CGT, a game $G$ is played by the two players $L$ (Left) and $R$ (Right). $L$ tries to maximize the game score, $R$ tries to minimize it.\(^5\) $G$ can be written as an ordered pair of two sets, $G = \{G^L \mid G^R\}$, where $G^L$ and $G^R$ are the Left-Options and Right-Options, that is the games reachable through a move of $L$ or $R$, respectively (since, in general, $G$ is only a sub-game of the full game in which the players take turns, the first player is not determined a priori, and it is possible for one player to play more than one move in a row). Games $G$ and $H$ can be combined to form a sum-game $G + H$ according to $G + H \equiv \{G^L + H, G + H^L \mid G^R + H, G + H^R\}$.

The value of terminal positions is often interpreted as “number of moves left for the winner” in CGT, rather than as some abstract “value” of the game. As far as the search for an optimal strategy is concerned, these two points of view are equivalent.

The simplest possible game is 0 \(\equiv \{\mid\} \equiv \{\emptyset \mid \emptyset\}\), which has no moves left for either player. It is won by the “second player”, that is by the player whose turn

---

\(^4\)Olivier Teytaud, personal communication.

\(^5\)Where game boards are presented in this report, $L$ will invariably be represented as bLack, and $R$ as white.
it is currently not to play. Similarly simple are games like \(-n - 1 \equiv \{ | -n \}\) (a board situation in which \(L\) has no moves left, and \(R\) can move into a position which leaves him \(n\) moves ahead, so he is currently \(n + 1\) moves ahead) or \(n + 1 \equiv \{n|\}\). Rational values are also possible, e.g. \(\{ n \mid n + 1 \} = n + \frac{1}{2}\) and \(\{-n - 1 | -n\} = -n - \frac{1}{2}\). In fact, any game of the form \(G = \{x | y\}\) with \(x, y \in \mathbb{R}\) and \(x \leq y\) can be written in such a closed form, and is called a Surreal Number (alternative names are “Conway number” and, within CGT, simply “number”).

With a full iterative constructive rule for all of them, they can be shown to form a partially ordered field \([16]\), which contains the real numbers and “infinite” and “infinitesimal numbers”, which are respectively larger and smaller in absolute value than any real number.

From an applied standpoint, however, these numbers are less interesting, because they represent only a subset of all games: those in which neither of the players wants to make a move, as it would weaken his position and are thus also called cold games. Games \(G < 0\) can be interpreted as “games in which \(R\) is guaranteed to win if he plays optimally; games with \(G > 0\) as “games in which \(L\) is guaranteed a win if he plays optimally” and \(G = 0\) as “games in which whoever plays first will lose, if his opponent plays optimally”. Of more interest to an actual player are hot games, in which each player still has a chance to advance his position. These games are not part of the field of Surreal Numbers, as they are neither > nor < nor = to some set of Surreal numbers, but confused with these numbers. Consider, for example, the game “+", which is defined as \(* \equiv \{0|0\}\). It is confused with 0 (and only with 0, earning it a special symbol): Whoever plays first wins immediately.

So, if a sum of games consists only of (surreal) numbers and one hot sub-game, optimal players will be eager to be the first to play in the hot game, and not be interested in the numbers, as they represent “settled” games. What, however, if there are several hot games around? Which game is “the hottest”?

To solve this problem, CGT makes much use of a concept called the Temperature of a (sub-) game. In preparation for a definition of Temperature, consider the game \(G\), chilled by a Temperature \(t \in \mathbb{R}\), which is defined to be

\[
G_t = \{G^L_t - t | G^R_t + t\}
\]

(5)

unless there is a smaller Temperature \(t'\) for which \(G_{t'}\) is infinitesimally close to a number \(m\), in which case \(G_t = m \forall t > t'\). Such a \(t'\) exists for all hot games. It is called the Temperature of \(G\), \(T(G)\). The number \(m\) is called the Mean of \(G\), \(\mu(G)\). In simplified terms, chilling a game corresponds to introducing a “tax” the players have to pay to be granted the right to make a move. In this view, \(T(G)\) is the highest tax a player is willing to pay for the first move in \(G\): At higher Temperatures, the game turns into a number, and both players become unwilling to make another move. \(T(G)\) is thus a measure of the urgency of playing in \(G\), while \(\mu(G)\) measures the overall value of \(G\).

Thermography, the study of game Temperatures, is a surprisingly complex field and beyond the scope of this work. For our purposes, it suffices to know that, if
the Temperatures of all sub-games constituting a full game are known, CGT provides approximate strategies for good play: If only the Temperatures are known, always playing in the hottest game ("hotstrat") is already a good strategy [15]. With slightly more information about the structure of the sub-games, a strategy called "sentestrat" provides an approximation with bounded error on the perfect line of play [17]. The remainder of this report will focus on measuring the Temperature of a single game. The application of the results, using the strategies gained from CGT, to boardgames consisting of sub-games is straightforward if the sub-games are known. For some real-world games, like Go, identifying these sub-games can be a challenging problem of its own.

2.3 Analytical Temperature discovery search

Given a game situation $G$ in a game whose optimal solution can be found by exhaustive search, the Mean and Temperature of $G$ can be found with the help of an Enriched Environment [18]: A sum game of $G$ and another (simple, artificial) game $C$ which has a known optimal solution and variable Temperature. The Temperature is found by varying the Temperature, searching for an optimal solution for the sum game $G + C$ and observing the lowest Temperature of $C$ that still allows for optimal play to start in $G$.

![Figure 1: An enriched environment: The (Go) game $G$ whose Temperature is to be measured, and a coupon stack $C = C(3, 1/2)$](image)

One realization for an Enriched Environment is a Coupon Stack $C \equiv C(V_m, \delta)$, consisting of a stack of coupons with decreasing face values (figure 1). The coupon values correspond to moves that are granted to the player who holds a particular coupon at the end of the game. At the beginning of the game, the top-most coupon has a value $V_m = n\delta$, with $n = 2k, k \in \mathbb{N}$, followed by coupons of values $V_m - \delta, V_m - 2\delta, \ldots, \delta$. At each point in the game, the player whose turn is up has the choice between making a move in $G$ and taking the current top-most coupon from the stack instead, thereby immediately gaining the face value of this coupon toward his final score.
The value $V(C(V_m, \delta), L)$ of $C$ with $L$ as the first player is

$$V(C(V_m, \delta), L) = \frac{n}{2} \delta,$$

and

$$V(C(V_m, \delta), R) = -\frac{n}{2} \delta$$

(6)

And thanks to the trivial form of optimal play for a stack on its own (simply always taking the topmost coupon), this means the Temperature of the stack is $T(C(V_m, \delta)) = V_m$, and its Mean is $9.0$.

So it is possible to measure the Temperature of $G$ by using an exhaustive search algorithm\textsuperscript{10}, like $\alpha\beta$-search, to find the optimal line of play – the Principal Variation – in the sum game $C + G$. The Temperature $T(G)$ is given by the face value of the last coupon taken before optimal play switched to $G$ for the first time. The Mean $\mu(G)$ is given by $\mu(G) = V(C + G, P) - V(C, P)$, where $P \in \{L, R\}$ is the player who got the first move in the sum game and $V(C, P)$ is the value of the optimal solution found by the search algorithm. This method is known as Temperature discovery search [19].

2.3.1 Some subtleties

There are some technical and analytical issues that have to be taken into account when performing Temperature Discovery Search.

**Extended Coupon Stacks** Many games allow for Zugzwang positions, where one player can exploit the fact that his opponent has to make a move in his round as leverage on the outcome of the game. To identify these situations correctly, the Enriched Environment has to be able to represent negative Temperatures, giving the players the option to pay for the privilege of not having to move. This can be realized through an extended coupon stack, in which the last coupon of value $\delta$ is followed by coupons of value $0, -\delta, -2\delta, \ldots, -1 + \delta$, then an even number of coupons of value $-1$, and finally a coupon of value $-\frac{1}{2}$, to keep the values $V(C, L)$ and $V(C, R)$ equal to those of a standard coupon stack of same $V_m$ and $\delta$ [19]. All coupon stacks used in our experiments are such extended stacks.

**Overestimation of Temperature** Optimal play in the sum game does not necessarily have to switch to $G$ at the Temperature, but only above $T(G)$. Consider, for example, the game\textsuperscript{11} $G = \{\{4 \mid 0\} \mid \{\}\}$. Here $T(G) = 0$, so we would expect optimal play to switch from $C$ to $G$ at a coupon value of $0$, but $L$ has a move available which leads to a new game $\{\{4\mid 0\}\}$, which has Temperature $2$. So one

\textsuperscript{9}One way to explain this result without delving to deep into Thermography is to interpret the Mean as the “average outcome” of a large sum game

$$\mu(G) = \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} G$$

(7)

Irrespective of who starts in the sum game, both players start as the first player in about half of the sub-games, which for the case of the coupon stack leads to Mean $0$.

\textsuperscript{10}In fact, enriched environments have been used in actual play among humans as well, to approximately measure the Temperature of a Go game; or rather, the humans’ estimation of its true value [18].

\textsuperscript{11}The notation $\{\|\mid\|\}$ is a CGT convention. The separation with the largest number of bars represents the root of the game tree, separations with lower numbers of bars are deeper in the tree.
principal variation for $C + G$ is for $L$ to start playing in $G$ at any coupon value below 2, because it is clear that $R$ will have to answer $L$’s threat in $G$, allowing $L$ to return to $C$ afterward without losing a coupon. In $\alpha\beta$-search, this problem can be solved by ordering available moves, so that the searcher tries solutions with low Temperatures first, leading to pruning of the equivalent solutions of higher Temperatures.\footnote{\cite{19} does not offer this solution, but proposes to do iterative search runs with lower and lower values of $V_m$. These are obviously very costly, and the solution mentioned here works just as good in my experiments.}

**Choice of $\delta$ and $n$** Obviously, Temperature Discovery Search demands $V_m$ be larger than $T(G)$. Since the temperature is the very quantity searched for and thus not known a priori, one has to resort to upper bounds on the Temperature, which are available for most games (a loose bound is simply the greatest possible margin of win between the two players).

The choice of $\delta$ (which also fixes $n$) is less straightforward. Firstly, the discretization-error caused by the choice of $\delta$ is, somewhat surprisingly, not bounded by $\delta$: We were able to reproduce the results of \cite{19}, which first showed this. The solution offered in the op.cit. is to choose $\delta$ small enough (in a domain-specific manner) to fully resolve all possible Temperatures, which eliminates all errors. For reasons that will become clear in section 3.2, this is not a viable path for a machine learning approach. It is comforting, however, that the errors fall monotonically toward 0 with falling $\delta$.

### 3 Method

#### 3.1 UCT vs. the Thomson Heuristic

The primordial idea for the work presented here was to combine UCT with enriched environments to gain an estimate for the Temperature of a single game. However, UCT provides a point estimate of only the principal variation: After a given number $n$ of tree-descents, this point estimate is given by the path along maximal $\langle X_{i,s} \rangle_{s \in \{1, \ldots, n\}}$. The problem of Temperature overestimation noted in 2.3.1 makes a Bayesian approach desirable: If there are several optimal paths through the game tree, a method based on full distributions instead of point estimates should include all of these options, and allow identification of the one with lowest Temperature post-hoc.

For this reason, we introduce a Bayesian variant of UCT, the Thomson Heuristic \cite{20}. Instead of a non-parametric point-estimate of the principal variation, our method learns a parametrized probability model for the distribution of rewards (figure 2): The (deterministic) reward for playing a particular child $j$ of node $i$ followed by optimal play is $\mu_j^i$. It is masked by zero-mean Gaussian noise of precision $\tau_j^i$ caused by the randomness of roll-outs, to give a reward distribution $\rho_j^i$. Finally, the shifting of the roll-out policy deeper in the tree after this child causes another, also zero-mean Gaussian noise, which has precision $t$ (assumed to be known and constant over the whole tree)\footnote{Another, maybe more realistic model would be to leave out the second Gaussian noise and model the shifting of reward distributions by “blowing up” $\sigma^2 = 1/t$ after each roll-out. Previous experiments in Go, however, show this to not work quite as well (David Stern, personal communication).}, producing the observed reward

\[ \text{observed reward} = \mu_j^i + \tau_j^i N(0, \tau_j^i) + t N(0, t) \]
3.1 UCT vs. the Thomson Heuristic

$X_i^t$ (For better readability, the superscripts $i$ will be omitted in the following). The assumption of zero mean for both distributions is only reasonable if the random roll-outs are unbiased, which is a good assumption if the game itself is not very biased. See section 5 on possible problems caused by biased roll-outs.

During learning, descents into the tree are generated – analogous to UCT – by drawing values of $\rho_j$ for all children $j$ from the current belief over its values and choosing the child with maximal $\rho$. Once a new, so far unseen node is reached, a random roll-out is played and its outcome used to update the believed values of the parameters $\rho, \tau$ and $\mu$ analytically using Bayes’ rule:

$$P(\rho_j,s | X_j,s, t, \mu_{j,s-1}, \tau_{j,s-1}) = \frac{P(X_j,s | \rho_{j,s-1}, t, \mu_{j,s}, \tau_{j,s})P(\rho_{j,s-1} | \mu_{j,s-1}, \tau_{j,s-1})}{P(X_j,s)}$$

$$= \mathcal{N}(X_j,s; \mu_{j,s-1}, t) \cdot \mathcal{N}(\rho_{j,s-1}; \mu_{j,s-1}, \tau_{j,s-1})$$

$$= \mathcal{N}\left(\rho_{j,s}; t \cdot X_j,s + \tau_{j,s-1} \cdot \mu_{j,s-1}, \frac{t + \tau_{j,s}}{t + \tau_{j,s}} \cdot \mu_{j,s-1} \right)$$

\[\therefore \mu_{j,s} = \frac{t \cdot X_j,s + \tau_{j,s-1} \cdot \mu_{j,s-1}}{t + \tau_{j,s}} \quad \text{and} \quad \tau_{j,s} = t + \tau_{j,s-1} \quad (8)\]

This update causes $\tau$ to rise continuously as the algorithm becomes more convinced of the true value of $\mu$. A more elaborate learning model (using the conjugate $\Gamma$-Prior for $t$) was tested as well and found to produce worse results than this simpler model. Child nodes that have not been played before are assigned a value drawn from the prior for $\rho$, which has $\mu_{j,0} = 0$. The parameter $t$ and the prior $\tau_{j,0}$ have to be tuned to each other. We were most successful setting them to be equal to each other and both very small to allow for a longer exploration phase before the algorithm settles. In the experiments shown in the

![Graphical model for the Thompson heuristic](image-url)
3.2 Algorithmic enhancements

The Thompson Heuristic on its own is capable of finding good moves in a game. But solving an enriched environment, i.e. finding the correct Temperature, is more complicated than that, because it asks for a quantity (the location of the switch from C to G, which we will denote by $V$), which lies deep in the tree, several levels below the root. The next two paragraphs present two add-ons to our method that partly alleviate these problems. The first one makes the search easier, the second one corrects for some of the algorithm’s errors.

3.2.1 Iterative Parameter-Updates

Because both UCT and the Thompson Heuristic start at the root of the search tree and expand it iteratively, it is possible for the algorithm to be stopped before it even reaches the regions of the tree where the optimal switch from C to G should take place. It is possible, though, to “raise” this location within the tree, by raising $\delta$ (at the expense of precision) or lowering $V_m$, if a region of high Temperatures can be safely excluded from the search. When searching complex games, where the maximal possible Temperature to expect, $T_m$, is large, we thus iteratively update these two parameters: The search starts off with $V_m = T_m, \delta = T_m/5$. After a burn-in of 200,000 roll-outs, the search is halted, and 10,000 descents into the tree are generated without updating the parameters of the Thompson Heuristic, so that the individual results become i.i.d. (see section 4.3 for a motivation of these choices of parameter-values). Each of these descents can be interpreted as a sample from the machine’s current belief over the distribution of Principal Variations, yielding one sample of $V$ from the current belief over these values. We generate a histogram $H(V) \propto P(V|\mu, \tau)$, where $\mu$ and $\tau$ are the current beliefs over the $\mu_i$ at all nodes $i$ in the tree. The search parameters are then updated to

- $V_m = \max_i(H(i)|H(i) > 0)$
- $\delta$ is set to be the largest $\delta$, such that the region between the 20th percentile and 100th percentile of $H(V)$ are covered by 5 or more coupons, and $(V_m + 1)/\delta = n = 2k, k \in \mathbb{N}$.

The iteration is stopped once $V_m$ does not change from one iteration to the next. This ensures the true Temperature is always within 5 layers below the root of the tree with high probability, while making the algorithm unlikely to accidentally “run over” a high Temperature.

3.2.2 A model for the relationship between $V$ and $T$

As mentioned in 2.3.1, it is possible for a principal variation to exhibit a value of $V$ that is larger than $T$, even when sampled from a perfect player. To generate

\[ t = \tau = (5M)^{-1} \]
a posterior belief $P(T|V)$, over Temperatures given a set $V$ of sampled values of $V$, we use a model for $P(V|T)$. If the sampled values of $V$ were generated by an actual optimal player without any preference for particular values of $V$ (as long as they allow for optimal play), the model would be

$$P(V|T) = \begin{cases} 0 & \text{for } -1 \leq V \leq T \\ \frac{1}{T_m-T} & \text{for } V \geq T \end{cases}$$

(10)

Since our algorithm is hardly an optimal player, we have to allow for some underestimation of $T$, lest we assign 0 probability to a particular Temperature based solely on one single observation of a low $V$. A simple form for this function is of the form

$$P(V|T) = \begin{cases} \varepsilon + 1 & \text{for } -1 \leq V \leq T \\ 1 - \varepsilon & \text{for } V \geq T \end{cases}$$

(11)

with some noise-level $\varepsilon < 1$. This model is only valid if $V_m \geq T_m$, i.e. if all possible values of $T$ are actually searched by the Thompson learner. If the algorithm is initialized with $V_m < T_m$, the model has to be adapted slightly (see also figure 3): Consider the event $V = V^*$, denoting the special case where the sampled principal variation starts directly in $G$. Observing this event conveys less information than any lower value of $V$: It only tells us that, had $V_m$ been chosen larger, the value of $V$ sampled would then have been somewhere between our current $V_m$ and $T_m$. So we have to marginalize over all these possible values of $V$, to get a value for $P(V^*|T)$. The whole function then takes the form

$$P(V|T) = \begin{cases} \frac{\varepsilon + 1 - \varepsilon}{T_m+1} & \text{for } -1 \leq V \leq T \land V \neq V^* \\ \frac{1}{T_m-T} & \text{for } V \geq T \land V \neq V^* \\ (1-\varepsilon) \frac{T_m-V^*}{T_m-T} & \text{for } T \leq V = V^* \\ 1 - \varepsilon \frac{V_m+1}{T_m+1} & \text{for } V = V^* < T \end{cases}$$

(12)

which is normalized to 1 for both the two cases of $T > V_m$ and $T < V_m$.

4 Experimental Results

4.1 Amazons

All experimental results presented in this section were carried out using a board game called *Amazons*. Amazons is typically played on a square board (often a chess board), but can be played on any 2D Cartesian grid of arbitrary extensions (figure 4). Each player controls a group of figures called “Amazons” (typically represented by chess queens. Our experiments, covered mostly cases with only one Amazon per player). A legal move consists of

1. Moving an Amazon any number of vertices either horizontally, vertically or diagonally, without “jumping” over other non-empty vertices (i.e. just like a queen in chess).

2. “shooting an arrow” (a Go stone), which starts its move at the final position of the moved Amazon and also moves like a chess queen, any number of vertices vertically, horizontally or diagonally. Once the arrow has
4.1 Amazons

Figure 3: Model for $P(V|T)$. Left: $P(V|T)$ with parameters set to $\varepsilon = 0.1$, $T_m = 9$, $V_m = 7$, for $T = 5$ (solid) and $T = 8.5$ (dashed). Since $V^*$ has no well-defined value, the value of $P(V^*(T))$ is plotted at an arbitrary point on the abscissa. Right: The same function plotted as a function of $T$, for $V = 5$ (solid) and $V = V^*$ (dashed). The dashed curve was scaled by a factor of 0.3 for readability.

Figure 4: A small but hot Amazons board. If White gets the first move, he will find it easy to win. If Black moves first, however, the game is still open.

reached its chosen destination, it blocks this vertex and cannot be moved or removed from the board.

Note that a legal move consists of both moving an Amazon and shooting an arrow. The first player unable to make a move loses. A typical game of Amazons consists of both players trying to “fence in” their opponent while trying to avoid being fenced in themselves.

Amazons is interesting from a CGT perspective because games tend to automatically separate themselves into independent sub-games, called rooms, as parts of the board become isolated from each other by walls of arrows. For our experiments, Amazons provides a convenient domain because analytical, “gold standard” solutions for Temperatures and Means can be generated using the “CGSuite” toolbox by Aaron Siegel\textsuperscript{14}.

Humans play Amazons on a $10 \times 10$ board, with 4 Amazons for each player. The Amazons positions in the experimental results shown here invariably use

\textsuperscript{14}http://sourceforge.net/projects/cgsuite
much smaller board sizes. Note that, because of the two-step nature of an Amazons move, even a $4 \times 4$ board with one Amazon per player has about 80-100 available moves at the first level of the tree (depending on the positions of the Amazons on the board), and preliminary experiments found more than $2 \cdot 10^9$ terminal nodes in the full game tree of such a position. The algorithm used by CGSuite to calculate game Temperatures is highly optimized\footnote{A superficial inspection of its code suggests that CGSuite uses an exhaustive expansion of the game tree to generate a canonical form of all reachable terminal nodes and then uses CGT rules to calculate the Temperature from these options.}, but such large trees still exhaust its resources quickly. The limiting factor on the size of the boards shown in the following sections were thus invariably the time requirements of the exhaustive search methods (both our own $\alpha/\beta$ method’s and CGSuite’s) used to generate reference values, and not the performance of the approximate searchers.

### 4.2 UCT vs. the Thompson Heuristic

![Figure 5: The Board used to test the performance of the two reinforcement-learning players against each other.](image)

To evaluate the performance of the Thompson heuristic without temperature estimation, on a single game board without an enriched environment, implementations of UCT and the Thompson heuristic were set to play against each other on a $4 \times 4$ Amazons board with one Amazon per player. The UCT player was set to play white, while the Thompson player played black. The initial situation was symmetric with respect to the colours played (figure 5). Both players were given the first move in half of the test runs, and both players were allowed the same number of search descents per move (50,000). The Thompson heuristic won 73 of the 100 test runs, showing a clear advantage for the Bayesian approach. The overall computation time required for these 100 matches is about 1 hour.

While a closer inspection would be necessary to fully explain this result, and intuitive explanation for this result is that UCT has to play every available node several times (albeit more and more rarely) during the learning process because the UCB score of unplayed nodes rises over that of unplayed ones. In particular if two nodes have very similar values, UCT will regularly be forced to return to the worse node of the two, even if it consistently returns inferior rewards. In contrast, the Bayesian searcher effectively stops playing worse nodes once they have been “proven” with sufficient evidence to be worse.
4.3 Convergence of training

Figure 6: Convergence of histograms over $V$ with progressing training. For an empty $4 \times 4$ board and $V_m = 15$, an initial (normalised) Histogram $H(1000)$ was sampled from 10,000 descents without parameter updates after 1000 roll-outs of training. From there onwards, the number of training roll-outs was raised by steps of 1000, and after each such step a new normalised Histogram was sampled from 10,000 descents. This plot shows the absolute deviation $\sum_{i=1}^{N} |H_i(n-1000) - H_i(n)|$ between Histograms as a function of the number of roll-outs, for three different values of $\delta$. Note the secondary peaks for smaller values of $\delta$, showing the dynamics of the model as new lines of play are discovered.

Figure 6 shows the change in sampled Histograms of $V$ with the number of training roll-outs. Even for a very fine-grained coupon stack with 16 coupons, there is not much change in the Histogram after 30,000 roll-outs. As our iterative implementation starts out with a very coarse coupon stack, this data motivated the use of 200,000 roll-out burn-ins between iterative up-date as a save upper-bound.

4.4 Small boards

A first data-set consists of 100 rooms of size 6 within $4 \times 4$ boards. Figure 7 shows 3 examples from this data-set. These rooms can easily be solved as-
alytically: Our implementation of $\alpha\beta$-Temperature Discovery Search needs to expand about 800 nodes per board to find a result for $V_m = 3$ and $\delta = 1/2$ (the Temperature of an Amazon's room of size $n$ containing $k$ Amazons is upper-bounded by $n - k - 1$). The Thompson Temperature searcher is at a disadvantage here because it needs some time to converge. In our experiments it was trained for 100,000 roll-outs per room, without any iterative updates to the search parameters, which were set as for the $\alpha\beta$ machine mentioned before. The results are shown in figure 8.

![Figure 8](image)

Figure 8: Results of the measurements on rooms of size 6. Left: Averages over sampled values of $V$ against true game Temperatures. Middle: Maximum a-posteriori values for $P(T|V)$, using the likelihood-model. Right: Averages over sampled Means, plotted against true game means.

The deviations of the estimated Temperatures from the true values can be explained to only a small part by the discretization of the coupon stack: [19] reports an error of 0.01 for the Temperature at $\delta = 0.5$ for these rooms.

4.5 iterative search and medium-sized boards

![Figure 9](image)

Figure 9: Results of the measurements on rooms of size 15. Left: Averages over sampled values of $V$ against true game Temperatures. Middle: Maximum a-posteriori values for $P(T|V)$, using the likelihood-model. Right: Averages over sampled Means, plotted against true game means. This data-set was generated with the iterative search algorithm, with 200,000 roll-outs per iteration. If the machine is trained for only 50,000 roll-outs per iteration, the strong overestimation of $V$ visible in the left-most plot disappears.

The small rooms are at best a sanity check, as they allow the algorithm to explore the whole tree. In contrast to this, our second test set consists of 150 rooms randomly sampled from the set of all $4 \times 4$ boards with one Amazon per player and one arrow, all three at a random position. These boards are basically
impossible to search with $\alpha\beta$-Temperature Discovery Search. For this test-set, the Thompson-searcher was allowed 200,000 burn-in runs per iteration, and search parameters were adaptively updated after every iteration. The adaptive process converged typically after 1 to 3 iterations. Results are shown in figure 9.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure9.png}
\caption{Log-Likelihood of the true Temperature under the model as a function of $\epsilon$. The first 50 rooms of the data-set of rooms with size 15 after 200,000 roll-outs were used to generate this plot. With data from 50,000 roll-outs, the maximum-likelihood value of $\epsilon$ moves from 0.5 to 0.2.}
\end{figure}

To fix the value of the free noise parameter $\epsilon$ in the likelihood model, 50 rooms from this test set were used as a training set. Figure 10 shows the sums of the Log likelihoods of the true Temperatures of these rooms under the model for varying values of $\epsilon$. A value of $\epsilon = 0.4$ was established as the maximum likelihood solution.

5 Discussion

The results presented in the previous section could be interpreted as a good start in the right direction:

1. The Thompson Heuristic is playing better than UCT on Amazons Boards
2. Approximate Temperature Discovery Search converges to the right values in the limit of a fully expanded tree
3. The search results of iterative search on large boards are at least clustered around the correct values, albeit with a large error.

However, our results for larger boards might also be a hint to a more fundamental underlying problem, either with the use of Enriched Environments for Temperature search, and with greedy reinforcement learning in game-trees in general. The next paragraph investigates where such problems might stem from.

5.1 Weaknesses of greedy tree search

In the limit of infinitely many roll-outs, UCT converges to optimal play. Recall from equation (2) that UCT’s regret is bounded by
5.1 Weaknesses of greedy tree search

\[
8 \sum_{i \mu_i < \mu^*} \left( \frac{\ln n}{\mu^* - \mu_i} \right) + \left( 1 + \frac{\pi^2}{3} \right) \left( \sum_{j=1}^{K} \mu^* - \mu_j \right)
\]

(13)

After finite time, there are two mechanisms influencing UCT’s performance negatively:

1. If (at least) one of the non-optimal available moves \( j \) at a given node returns a true reward \( \mu_j \) which is very close to the reward \( \mu^* \) of the optimal move, the bound on the regret rises inversely with \( \mu^* - \mu_j \). While this is only of academic interest if the goal of UCT is just to find reasonably good moves, it could be problematic if UCT is used to search for more complex concepts, like the values \( V \) used in our Temperature Discovery Search, which are more susceptible to subtle changes in the line of play.

2. If the optimal line of play follows a child-node \( k \) whose average return \( \bar{X}_k \) under random play is very low, UCT will play it very rarely and need a long time for convergence.

It becomes clear that Enriched Environments are a particular realization of problem 1: By construction, an enriched environment lowers the difference between the rewards for individual moves, because they give both players the chance to set off losses made in the game by playing on the coupon stack. Worse, the effect of this levelling is most pronounced at the point in the tree which is most interesting for our application: At a top coupon value of \( V = T \), playing on the stack and playing in the game \( G \) return exactly the same reward, so both UCT and the Thompson-Heuristic will be most unsure about the optimal line of play at this point in the tree. This could be partial cause of the underestimations of the Temperature visible in figure 9.

The next paragraph investigates how likely the second problem is to appear in a real game, by trying to construct a tree that causes \( \bar{X} \) to be very biased.

5.1.1 A worst case scenario

Constructing a game tree that breaks layer-wise reinforcement learning is not straight-forward: Assume the machine is playing the max-player, L. On the one hand, simply mixing a good move among a lot of bad ones two layers under the root is not enough, because the algorithm would discover this pretty quickly, once it has expanded this second layer. The principal variation needs to be a long, co-ordinated sequence of moves. However, because of the min/max-property of games, we cannot simply “hide” one single long sequence with a high reward among a lot of sequences with low rewards, because the min-player would take advantage of that. At each min-node in the tree, the optimal line of play has to be min-solution, so all other options have to lead to larger rewards. This property will cause the random roll-outs used by the machine to tend to more positive results, revealing the optimal line of play more quickly.

The solution of this advocatus diaboli exercise is to bias the roll-outs, by offering many more non-optimal moves to the max player than to the min player, and to bias the rewards, so that non-optimal play carries a larger penalty for one player than for the other. Figure 11 shows a tree that satisfies this requirement: The right-hand side of the tree is the “coward’s game”: It has length \( H \) and
5.1 Weaknesses of greedy tree search

Figure 11: An artificial game tree that breaks layer-wise reinforcement learning. MAX-nodes are denoted by circles, MIN-nodes by squares. The principal variation is shaded.

offers a low but guaranteed reward \( w > 0 \) to \( L \). The left side of the tree is the “daring game”. It potentially offers a large reward \( W \) to \( L \), if \( L \) sticks to an optimal line of play for \( D \) moves. Here, an ignorant random MIN-player has 50% chance to play non-optimally at each node, because he is offered only two different moves. A random MAX-player, on the other hand, has a very low chance of playing optimally, because he is offered a large number of non-optimal moves along-side the one optimal one. To further enhance the problem, \( R \) can only play “non-optimally” in the sense of making it too easy for \( L \): Even if \( R \) plays the non-optimal move at any point, the reward for \( L \) stays \( W \) (a lower reward is not possible without breaking the MIN/MAX-structure). But if \( L \) plays non-optimally in the daring game, the reward will drastically fall, to \(-W\). So random roll-outs starting at move 1 will have average returns of \( \approx 0 \), while all random roll-outs starting at the first layer under the root will return \( w \), leading a UCT player to only very rarely explore move 1, and thus need a large number of roll-outs to discover the good sequence.

Figure 12: A realization of the the artificial game tree on a one-dimensional board.

Figure 12 shows a realisation of this game tree in a very simple “game”, played on the interval \([1, 2H] \subset \mathbb{N}\). \( L \) starts the game by placing a stone on any of the
empty fields. \( R \) then fills a field as well, until all fields are filled. The final score is calculated as follows

- If \( L \) started with any other move than 1, the reward is \( w \) for \( L \), independent of what the following moves where.

- If the sequence played by the two players started with the principal variation \( 1, 2, 3, \ldots, 2D - 1, 2D \), the reward is \( W \) for \( L \).

- If \( L \) played the principal variation, but \( R \) answered wrongly at any point, the reward is \( W \) as well.

- If \( L \) started in the principal variation, \( R \) answered correctly, but \( L \) then chose the wrong move at any point in the principal variation, the “reward” (penalty) for \( L \) is \( -W \).

Critically, the available moves at any point in the game are handed to the players by a referee. If \( L \) is not playing the principal variation, the referee always returns all remaining open fields as legal moves. But if \( L \) started off by playing in the principal variation and has so far kept to it, the referee will become partial and only offer two moves to \( R \): The optimal one \((2n, \text{where } n \text{ is the number of moves made so far})\) and the largest available field. \( L \) remains to be offered all available fields as possible moves, even if he is playing the principal variation.

![Figure 13: Number of roll-outs needed by a UCT player to discover the principal variation in the game shown in figure 12, as a function of the length \( D \) of the hard sequence, for different sizes \( H \) of games. The rewards were set to \( W = 20, w = 1 \). The principal variation is considered as “discovered” after \( n \) roll-outs if \( X_1 > X_j \forall 1 < j < 2H \) for all roll-outs from \( n \) to \( n + 10,000 \).

Figure 13 shows the time needed for a UCT player to discover the principal variation in this game, for various very low values of \( D \) and \( H \). Note that \( D \leq H \) by construction. For larger games \((H = 4, D = 4 \text{ and } H \geq 5 \text{ and } D > 2)\), the UCT player does not discover the principal variation within \( 10^8 \) roll-outs, even though the number \( K \) of nodes in a game tree of a game of length \( H \) is bounded by

\[
K(H) \leq \sum_{i=0}^{2H} (2H - i) = \frac{2H(2H - 1)}{2}
\]

so \( K(5) \leq 45 \), but a UCT player needs more than \( 10^8 \) roll-outs to discover the principal variation consisting of 3 co-ordinated moves in such a short game. Due
5.2 Implications for Temperature Discovery Search

To time constraints, this experiment could not be repeated for the Thompson-Heuristic so far, but the results are bound to be even worse, given that the Thompson Heuristic grows more and more convinced of a move’s uselessness with growing numbers of roll-outs. In fact, the Thompson Heuristic will most likely get stuck in the Coward’s game and never converge to the principal variation.

If the concept of the “partial referee” is eliminated from the game and both players are offered all empty fields to play in at any point, the problem fully disappears and UCT settles on the correct move within less than $2H$ moves, even for comparatively large values of $H$ (I tested values up to $H = 100$).

5.2 Implications for Temperature Discovery Search

The artificial game presented in the preceding paragraph, with its partial referee and absence of good options for $R$, clearly would not be fun to play for humans. Indeed one might argue that all “interesting” games played by humans are unbiased, posing equally hard challenges to both players and offering equally large rewards. Particularly in Go, the set of available moves is nearly identical for both players, so strongly biased roll-outs are unlikely. Also, state-of-the-art Go machines like MoGo use pattern matching systems to avoid pathological bias in the roll-out results, explaining why this issue has not received much attention in the Computer-Go community.

For our application, though, Amazons could prove to be an unlucky choice of domain: Random Amazons positions regularly exhibit a strong asymmetry in the number and value of available moves for both players. Consider, for example, the board shown in the middle of figure 7: Of the three available moves for white, one is optimal and two are not game-losing, leading to longer sequences of play. Black, however, has only one move available (besides taking a coupon), which is uniquely bad. So Black will have no problem to discover taking a coupon as a good idea, while White has to explore more. In fact, the samples from $P(V)$ generated by our machines often exhibit an even pattern, caused by one player preferring to take coupons while the other starts to prefer to play on the stack, if the board situation has a strong asymmetry between players.

6 Conclusion

We have presented a reinforcement learning algorithm that

- is capable of winning against UCT on medium-sized Amazons Boards and
- produces coarse estimates of the Temperature of Amazons boards beyond the reach of analytical search methods.

It remains to be seen whether this method can be improved to a point where it could be used in a divide-and-conquer approach to Go games. Go could be an advantageous domain for this method because Go board situations exhibit less bias in random roll-outs. Apart from this, there are more open questions:

- Our method raises the complexity of the search tree for each sub-game. Whether or not this compares favorably with a potential efficiency gain
from the divide-and-conquer method depends on the fidelity of the Temperature estimates necessary for the divide-and-conquer step to work.

- How to divide a Go board into sub-games without much computational overhead remains an open challenge. Possible options range from statistical analysis of the roll-outs to a separate machine learning algorithm using off-line knowledge.

- Ideally, segmentation of the board into sub-games, Temperature Discovery for each sub-game, and identification of the optimal move in each game should be done by algorithms sharing as much data as possible, to avoid overhead caused by the first two steps compared to a vanilla UCT approach. Because the principal variation in the Enriched environment is not identical to the principal variation in a given sub-game alone, let alone the whole sum-game, the results of Temperature Discovery Search can not be re-used directly.

7 Future work

Working on this project for the past 14 weeks was an interesting experience and helped me gain insight into the state of the Art in Game-solving algorithms. It is certainly possible that a highly efficient divide-and-conquer algorithm might be competitive with the current best Go methods. However, this approach is relatively disconnected from other areas of machine learning and computational neuroscience, where my main research interests lie. It also offers very limited potential for generalization away from the field of Combinatorial Games to more general, real-world data.

I think that the current reinforcement learning methods for games could profit strongly from domain knowledge and an efficient representation of the state space. Vanilla UCT has to re-discover the rules and patterns of good Go play with every new board situation it faces. MoGo’s policy is already using off-line knowledge [21], by using a feature map generated with Temporal Difference Learning [8]. While it is certainly possible that a highly efficient divide-and-conquer algorithm might perform better than a highly expressive but slow reinforcement learning pattern database, I think that from the point of view of a machine learner, the latter option is much more interesting to investigate.

References


16the remainder of my first three terms was spent on an initial orientation and literature review phase in Michaelmas 2007, and a project on Deep Belief Nets for Go gameplay earlier this year.


