Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)

On Adversarial Search Spaces and Sampling-Based Planning

Raghuram Ramanujan and **Ashish Sabharwal** and **Bart Selman**
Department of Computer Science
Cornell University, Ithaca NY 14853-7501, U.S.A.
{raghu,sabhar,selman}@cs.cornell.edu *

Abstract

Upper Confidence bounds applied to Trees (UCT), a bandit-based Monte-Carlo sampling algorithm for planning, has recently been the subject of great interest in adversarial reasoning. UCT has been shown to outperform traditional minimax based approaches in several challenging domains such as Go and Kriegspiel, although minimax search still prevails in other domains such as Chess. This work provides insights into the properties of adversarial search spaces that play a key role in the success or failure of UCT and similar sampling-based approaches. We show that certain "early loss" or "shallow trap" configurations, while unlikely in Go, occur surprisingly often in games like Chess (even in grandmaster games). We provide evidence that UCT, unlike minimax search, is unable to identify such traps in Chess and spends a great deal of time exploring much deeper game play than needed.

Introduction

Monte Carlo sampling techniques have been successfully applied in the past to Markov Decision Processes (Chang et al. 2005) and also to adversarial reasoning domains. In particular, they have led to expert-level play in games of incomplete information such as Bridge (Ginsberg 1999) and Scrabble (Sheppard 2002). However, they have seldom outperformed traditional adversarial planning techniques such as the minimax algorithm in deterministic 2-player game settings such as Chess. This has changed recently with the emergence of UCT, a bandit-based planning algorithm (Kocsis and Szepesvári 2006; Auer, Cesa-Bianchi, and Fischer 2002). UCT was used to produce the first program capable of master level play in 9x9 Go (Gelly and Silver 2007; 2008), a domain that has thus far proven to be challenging for minimax-based strategies due, at least in part, to a large branching factor and lack of good heuristics. UCT has also proved promising in new domains such as Kriegspiel that were beyond the scope of any traditional planning techniques (Ciancarini and Favini 2009). This raises the question of whether UCT-style techniques can be successfully applied to other games like Chess, to general adversarial planning, or to solve QBF (quantified Boolean formula) instances. Unfortunately, the success of UCT is currently not well understood, in contrast to minimax search for which several intricate properties have been discovered (e.g., Nau 1983; Pearl 1983). It is reasonable to expect that UCT's performance relies heavily on certain properties of the underlying adversarial search space. We investigate this by studying a property that seems to play a vital role—the presence or absence of shallow search traps.

Most adversarial search is currently explored in competitive settings, where the overall goal is to build a winning player for games such as Chess, Go, or Poker. Such competition has clearly pushed the development of search techniques but it has come at a certain price. Our understanding of why one method outperforms another and in what domain is somewhat limited and anecdotal. This is in part because a winning player has to incorporate a whole range of techniques, including clever domain specific heuristics, to be competitive, which makes it difficult to study techniques in a more pure form. The goal of this paper is to contribute to a more fundamental understanding, in particular to a better understanding of the differences in adversarial search spaces and the implications for different search strategies.

Unlike minimax search, which builds an iteratively-deepening search tree and applies a heuristic evaluation function at non-terminal leaf nodes, the UCT algorithm grows a highly asymmetric, sparse tree in an opportunistic depth-first fashion, focusing on the most promising lines of actions from a given state. In addition, UCT does not require a domain-specific heuristic to perform well, a property that has made it an attractive proposition for the task of general game playing (Finnsson and Björnsson 2008). Interestingly, even though UCT uses a (biased) sampling method to build its search tree and applies averaging rather than minimax to compute utility values, in the limit the utility values computed by it do converge to the true minimax values. This desirable behavior emerges indirectly from the use of *max* and *min* operators in action selection when building the search tree. While this works well for some domains, UCT has thus far not been successful at other domains such as Chess.

In this work, we study the concept of states *at risk* and *shallow search traps*. Informally, a player is in a state that is at risk of falling into a search trap if there exists an unfortunate action m for him such that after executing m, the

*Supported by NSF Expeditions in Computing award for Computational Sustainability, 0832782; NSF IIS award 0514429; and IISI, Cornell Univ. (AFOSR grant FA9550-04-1-0151).

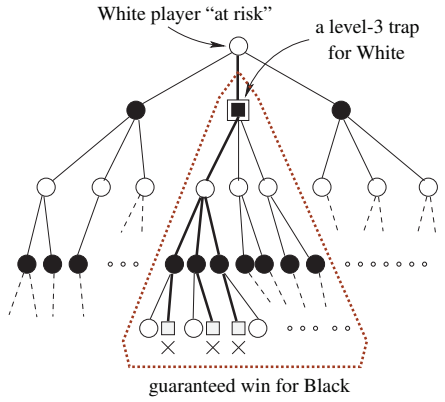

Figure 1: A level-3 search trap for the White player. The shaded square boxes at the leaves denote terminal nodes indicating a win for Black.

opponent has a guaranteed winning strategy (with optimal play). We will call the state after executing m a *trap*. We found that such traps—even at depths as low as 3 to 7—exist in interesting Chess positions, at various depths of play.

When traps occur at all depths, the best way of detecting and avoiding them appears to be an iterative deepening minimax style search. With UCT, even though it converges to minimax values in the limit, it turns out that it can recognize traps in Chess at level 3 but is likely to miss traps at levels 5 and higher. Specifically, our experiments reveal that the utility assigned by UCT to a trap move is often very close to the utility assigned by it to the best move. This suggests that even if UCT ranks a trap move somewhat lower than the best move, in general it has trouble distinguishing really good moves from really bad moves. Traps sprinkled throughout the search space at various levels are, of course, extreme examples of bad moves, and it appears very likely that UCT will choose perhaps a less extreme bad move (a "soft trap") during the full length of the game. This, we believe, is a major reason why minimax based strategies have been much more successful than UCT in games like Chess, even though UCT currently clearly dominates computer Go.

Traps in Adversarial Search Spaces

The question we seek to address is, *how often do adversarial search spaces exhibit traps and how likely is a player to fall into such a trap?* The notion of search traps and states at risk is illustrated in Figure 1 and a formal definition is given below. In the rest of this paper, a *k-move winning strategy* for a player p means that there exists an action a_1 for p such that for every possible counter-action of the opponent, there exists an action a_2 for p such that for every counter-action of the opponent, and so on, there exists an action a_k for p that results in a win for p.

Definition 1. In a 2-player game G, the current player p at state s is said to be **at risk** if there exists a move m from state s such that after executing m, the opponent of p has a k-move winning strategy. The state of the game after executing m is referred to as a **level-k search trap** for p.

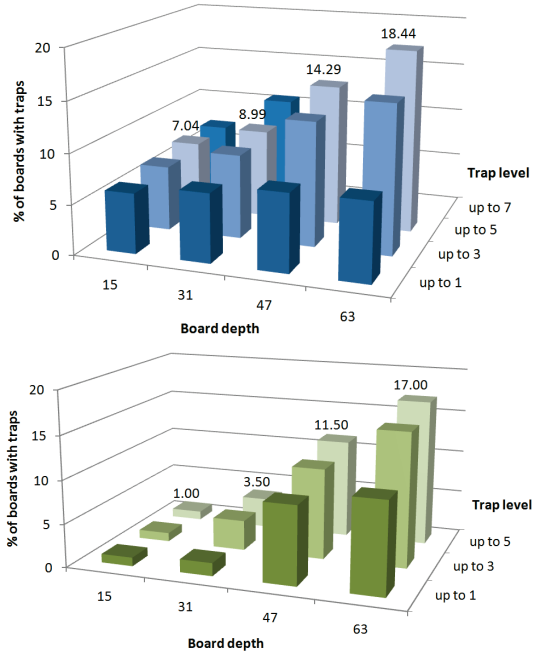

Figure 2: Percentage of Chess boards at various plys that have a shallow trap at level $\leq k$. Top: 200 semi-randomly generated boards. Bottom: 200 actual grandmaster games.

Clearly, a search trap is of concern only when the trap is actually computationally detectable by the opponent. In the extreme case, *every* move will lead to a position where, in principle, either the current player has a winning strategy, the opponent has a winning strategy, or there is a draw (if the game allows draws). However, if the winning strategy for the opponent is so complex or so deep that he cannot reasonably compute it, then making a move to get to that state shouldn't be viewed as falling into a dangerous trap. Therefore, we are interested in *traps that are identifiable with a reasonable amount of computation by the adversary*. This is quantified by the *level* associated with each trap, indicating the depth of the opponent's winning strategy.

We will be interested in relatively shallow search traps, typically those at levels 3, ..., 7. Further, a trap will be of even more interest when avoiding the unfortunate move associated with the shallow trap can actually lead to a much deeper game play and perhaps even a winning strategy for the current player. If the underlying adversarial search space does have traps of this nature, then it becomes critical for the player to identify and avoid such traps. This is where the difference between exhaustive algorithms such as minimax and sampling-based algorithms such as UCT comes in—we demonstrate that UCT-style algorithms have a good chance of missing even very shallow traps and thus losing the game.

Existence of Search Traps

At first, it appears that such shallow traps may be quite rare in interesting games. This seems to indeed be the case for games such as Go where it is highly unlikely or even im-

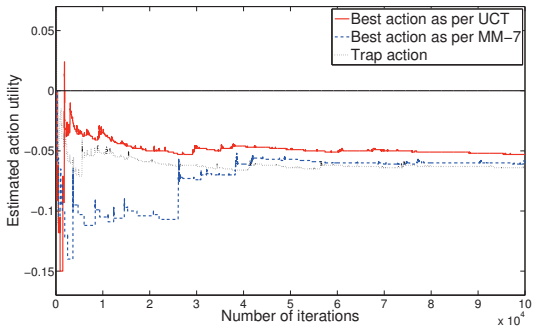

Figure 3: Utility estimates of UCT for the most promising actions (solid red at top and dashed blue) compared with that of a level-5 trap action (dotted black initially in the middle).

possible to find abrupt endings of games along one line of play and dozens of moves along a different line of play. This is often true in games where wins and losses are defined through the notion of overall territory, etc. On the other hand, in games such as Chess where wins and losses are defined through the capture of a certain key piece (e.g., the King) or, in general, through a property of a very small part of the whole state, we found that surprisingly shallow traps exist—not only for artificially created boards but even in games played by grandmasters.

Figure 2 shows the percentage of Chess boards that were found to have traps at various levels. For the top plot, we used 200 games where each move was played randomly with probability $1/3$ and based on the GNU Chess[1] heuristic with probability $2/3$. For the bottom plot, we took 200 games that were actually played by grandmasters.[2] In each case, we played the game to 15-ply, 31-ply, 47-ply, and 63-ply deep, and then computed whether or not the resulting board had a trap at levels 1, 3, or 5 (or 7 as well, in the case of semi-randomly generated 15-ply or 31-ply boards). The height of the bars in the plots indicate the percentage of the resulting boards that were found to have fairly shallow traps. For example, for semi-randomly generated boards that are 31-ply deep, roughly 7-10% of the games have a trap at levels 1 through 7. For games that are 47-ply deep, as many as 15% of the games have shallow traps. In games played by grandmasters, the percentage of traps is lower but still quite significant—roughly 12% for 47-ply games. This shows that *in games such as Chess, surprisingly shallow traps are sprinkled throughout the search space, at essentially every depth of the game.* Hence, a good game playing strategy for such a search space must incorporate the capability to recognize and avoid such traps.

Identifying and Avoiding Traps

This section describes our empirical investigation of the question, *can UCT-style algorithms successfully identify and avoid shallow traps?* Clearly, a k-level trap can be identified by a minimax search of depth $k+1$ starting from the current state. We therefore give UCT an equivalent search time (measured in terms of the number of nodes processed) and study its behavior. In all these experiments, the exploration bias parameter for UCT is fixed at a value of 0.4, for this was empirically observed to produce the best performance.

[1]Source: http://www.gnu.org/software/chess

[2]Source: http://www.chessgames.com

Table 1: UCT utility estimates for best action as per UCT, best action as per minimax depth-7, and the trap action. Shown for varying trap depths.

	UCT-best	Minimax-best	Trap move
level-1 trap	-0.083	-0.092	-0.250
level-3 trap	$+0.020$	$+0.013$	-0.012
level-5 trap	-0.056	-0.063	-0.066
level-7 trap	$+0.011$	$+0.009$	$+0.004$

For the first experiment, we consider the utility assigned by UCT to the action leading to the trap state, and compare it with the utility assigned by it to both the action UCT considers the best and the action minimax search (of depth 7, with the GNU Chess heuristic applied at non-terminal leaf nodes) considers the best. For Chess boards with White as the current player, the utilities are real numbers in the range $[-1,1]$, with $+1$ indicating a guaranteed win for White (if played according to a winning strategy) and -1 indicating a guaranteed loss. In particular, the action a leading to the trap for Black should have a utility close to -1.

Figure 3 shows the evolution of the utility of these three actions as a function of the number of iterations of UCT, up to the first 100,000 iterations. The board under consideration has a level-5 trap. Table 1 gives UCT utility values for three other boards as well, with traps at levels 1, 3, and 7, respectively. We make two observations from this experiment. First, the dotted black line, corresponding to the utility assigned to the trap action, is typically far from being assigned the ideal utility of -1, which is also seen in the table except for the board with a level-1 trap. Second, for level-5 and level-7 traps, the UCT utilities for the trap action are very close to the utilities assigned to the *best* action (best as seen by either UCT or minimax), to a point that it is not easy for the algorithm to distinguish the best action from an action resulting in a clear loss. As mentioned earlier, this suggests that UCT will have even more trouble distinguishing "soft traps" from good moves, and is likely to make moves that result, for example, in a significant material disadvantage.

One reason that UCT may have assigned a relatively high utility to the trap state for White (over -0.07 rather than the ideal -1) is that it simply didn't visit the trap state enough, presumably because it did not find the action leading to it to be a promising enough action. *Could the estimate of UCT get better if it were allowed to have more samples from the trap state?* We again found that this is not the case—even if UCT is allowed to sample as many as 10 times the number of nodes minimax needs to visit to identify the trap, the utility assigned by UCT to the trap state remains high, over -0.15,

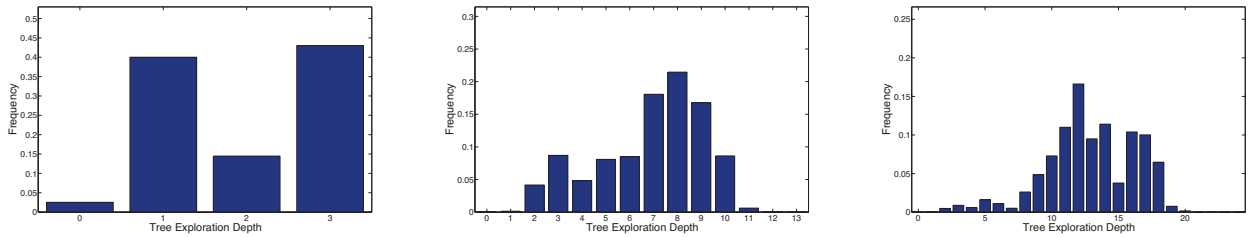

Figure 4: Histogram of search depths that iterations of UCT explored when in a trap state of depth 3, 5, and 7, respectively.

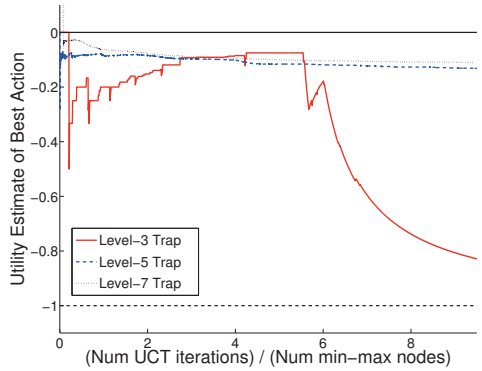

Figure 5: UCT estimate of the trap state stays incorrectly high, except for the level-3 trap (solid red curve that drops), even if the trap state is visited 10x times the number of nodes visited by the minimax search identifying the trap.

as long as the trap is at level 5 or higher.[3] This is depicted in Figure 5, which shows the evolution of the utility assigned by UCT to the trap state if UCT is started at the trap state itself and thus forced to visit this state in every iteration. For the level-3 trap (the solid red curve), UCT does identify the winning strategy and subsequently quickly converges to -1 as in the ideal case. However, for level-5 and level-7 traps, its utility remains high.

Finally, in order to better understand why UCT does not assign a utility even remotely close to -1 to trap states at levels 5 and higher even with 10 times the effort of minimax search (in some cases 50 times the effort; data not presented here), we took a deeper look at the main feature of UCT, namely, that it searches relatively much deeper into regions that it thinks are promising. Of course, for a level-k trap, minimax search would identify the level-k winning strategy and never explore nodes deeper than depth k. On the other hand, as the center and right-hand-side histograms in Figure 4 show, UCT spends nearly 70% of the time exploring nodes deeper than level 5 in the presence of a level-5 trap, and nearly 95% of the time exploring nodes deeper than level 7 (as high as level 20) in the presence of a level-7 trap. This is in stark contrast to the left hand side plot, which shows that UCT explores nodes only up to depth 3 in the presence of a level-3 trap, which it correctly identifies. In general, the amount of effort UCT spends beyond what it needs to had it identified the winning strategy, appears to increase exponentially with trap depth. This demonstrates that UCT pretty much fails to identify the (shallow) winning strategy during these search experiments.

[3]For level-7 traps, UCT did find the trap in some runs after around 5 times the effort of minimax search; nevertheless, it failed on several runs and risked the chance of falling into a trap.

Conclusion

This work provides new insights into the nature of adversarial search spaces, specifically in terms of the existence (or non-existence) of shallow traps, which UCT is not good at identifying. This helps explain the success of UCT in domains such as Go and its limitations in domains such as Chess. This study also indicates that UCT-style methods have the potential of enhancing adversarial reasoning systems such as QBF solvers, as long as the domain they operate on does not have traps in the underlying search space.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2-3):235–256.

Chang, H. S.; Fu, M. C.; Hu, J.; and Marcus, S. I. 2005. An adaptive sampling algorithm for solving Markov decision processes. *Operations Research* 53(1):126–139.

Ciancarini, P., and Favini, G. P. 2009. Monte Carlo tree search techniques in the game of Kriegspiel. In *IJCAI-09*.

Coulom, R. 2006. Efficient selection and backup operators in Monte-Carlo tree search. In *5th Intl. Conf. on Computer and Games*, volume 4360 of *LNCS*, 72–83.

Finnsson, H., and Björnsson, Y. 2008. Simulation-based approach to general game playing. In *AAAI-08*, 259–264. AAAI Press.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In *24th ICML*, 273–280.

Gelly, S., and Silver, D. 2008. Achieving master level play in 9×9 computer Go. In *23rd AAAI*, 1537–1540.

Ginsberg, M. L. 1999. GIB: Steps toward an expert-level bridge-playing program. In *IJCAI-99*, 584–589.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *17th ECML*, volume 4212 of *LNCS*, 282–293.

Nau, D. S. 1983. Pathology on game trees revisited, and an alternative to minimaxing. *Artif. Intell.* 21(1-2):221–244.

Pearl, J. 1983. On the nature of pathology in game searching. *Artif. Intell.* 20(4):427–453.

Sheppard, B. 2002. World-championship-caliber Scrabble. *Artif. Intell.* 134(1-2):241–275.