A Monte Carlo Approach for Football Play Generation

Kennard Laviers
School of EECS
U. of Central Florida
Orlando, FL
klaviers@eecs.ucf.edu

Gita Sukthankar
School of EECS
U. of Central Florida
Orlando, FL
gitars@eecs.ucf.edu

Abstract

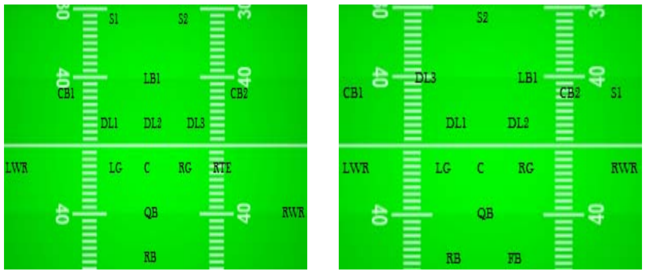
Learning effective policies in multi-agent adversarial games is a significant challenge since the search space can be prohibitively large when the actions of all the agents are considered simultaneously. Recent advances in Monte Carlo search methods have produced good results in single-agent games like Go with very large search spaces. In this paper, we propose a variation on the Monte Carlo method, UCT (Upper Confidence Bound Trees), for multi-agent, continuous-valued, adversarial games and demonstrate its utility at generating American football plays for Rush Football 2008. In football, like in many other multi-agent games, the actions of all of the agents are not equally crucial to gameplay success. By automatically identifying key players from historical game play, we can focus the UCT search on player groupings that have the largest impact on yardage gains in a particular formation.

Introduction

One issue with learning effective policies in multi-agent adversarial games is that the size of the search space can be prohibitively large when the actions of all players are considered simultaneously. Hence, single-agent reinforcement learning systems are relatively common, whereas multi-agent learning systems are less prevalent. In this paper, we demonstrate a multi-agent learning approach for generating offensive plays in the Rush 2008 football simulator. Rush 2008 simulates a modified version of American football and was developed from the open source Rush 2005 game.[1]

To succeed at American football, a team must be able to successfully execute closely-coordinated physical behavior. However, certain players, like the quarterback, clearly have a greater impact on the success of a given play. Yet changing the policy of the quarterback, without making corresponding changes to other teammates, can lead to a catastrophic failure of the play, whereas by simultaneously making changes in a subset of the players (e.g., the quarterback and a receiver) can amplify yardage gains.

Within the Rush football simulator, we observe that each play relies on the efforts of different subgroups within the main team to score team touchdowns. We devised a method to automatically identify these subgroups from historical play data based on: 1) mutual information between the offensive player, defensive blocker, and ball location 2) the observed ball work flow. After extracting these subgroups, we demonstrate how to determine which groups are the most important and use those groups to focus the search for new plays.



Power vs. 31 Split vs. 2231

Figure 1: Two match-ups between offensive (bottom) and defensive (top) formations. These lead to a variety of plays, in which different subsets of players can affect the play outcome. We identify frequently-observed coordination patterns from observed traces to guide multi-agent policy search.

Recent advances in Monte Carlo search methods have produced learning agents which effectively play games with very large search spaces where evaluating the value of intermediate states is difficult. In particular, the UCT (Kocsis and Szepesvri 2006) algorithm has produced promising results in a variety of games, ranging from Go (Gelly and Wang 2006) to the real-time strategy game WARGUS (Balla and Fern 2009). Using knowledge of key groups to bound search, we illustrate how UCT can be used to create new playbooks with user-defined levels of difficulty from novice to expert.

Rush Football

Football is a contest of two teams played on a rectangular field bordered on lengthwise sides by an end zone. Unlike American football, Rush teams only have 8 players on the field at a time out of a roster of 18 players, and the field is 100×63 yards. The game's objective is to out-score the op-

[1] http://sourceforge.net/projects/rush2005

ponent, where the offense (i.e., the team with possession of the ball), attempts to advance the ball from the line of scrimmage into their opponent's end zone. In a full game, the offensive team has four attempts to get a *first down* by moving the ball 10 yards down the field. If the ball is intercepted or fumbled and claimed by the defense, ball possession transfers to the defensive team. The Rush 2008 simulator only runs one play with the line of scrimmage set to the center of the field. Stochasticity exists in catching (i.e., whether a catch is successful), fumbling, tackles, distance/location of a thrown ball, and the selection of who to throw to if no receiver is "open" when the QB is forced to throw the ball.

The offensive lineup contains the following positions:

Quarterback (QB): given the ball at the start of each play. The QB hands the ball off or passes it to another player.

Running back (RB): begins in the backfield, behind the line of scrimmage where the ball is placed, with the quarterback and fullback.

Full back (FB): serves largely the same function as the RB.

Wide receiver (WR): primary receiver for pass plays.

Tight end (TE): begins on the line of scrimmage immediately to the outside of the offensive lineman and can receive passes.

Offensive linemen (OL): begin on the line of scrimmage and are primarily responsible for preventing the defense from reaching the ball carrier.

A Rush play is composed of (1) a starting formation and (2) instructions for each player in that formation. A formation is a set of (x,y) offsets from the center of the line of scrimmage. By default, instructions for each player consist of (a) an offset/destination point on the field to run to, and (b) a behavior to execute when they get there. Play instructions are similar to a conditional plan and include choice points where the players can make individual decisions as well as pre-defined behaviors that the player executes to the best of their physical capability. Rush includes three offensive formations (**power**, **pro**, and **split**) and four defensive ones (**23**, **31**, **2222**, **2231**). Each formation has eight different plays (numbered 1-8) that can be executed from that formation. Offensive plays typically include a handoff to the running back/fullback or a pass executed by the quarterback to one of the receivers, along with instructions for a running pattern to be followed by all the receivers. An example play from the **split** formation is given below:

- the quarterback will pass to an open receiver;
- the running back and fullback will run hook routes;
- the left wide receiver will run a corner right route;
- the right wide receiver will run a hook route;
- the other players will block for the ball holder.

Related Work

Rush 2008 was originally developed as a platform for evaluating game-playing agents and has been used to study the problem of learning strategies by observation of real-world football games (Li *et al.* 2009). A case-based reinforcement learning approach for modifying the quarterback's policy was demonstrated within Rush 2008 (Molineaux *et al.* 2009); note that all of the other players, other than the quarterback, always played the same strategy. (Laviers *et al.* 2009) created an online multi-agent play adaptation system that modified the team's policies in response to the opponents' play. However, the play adaptation system always utilized plays from the existing playbook, rather than creating new plays.

Monte Carlo rollout search algorithms have been used successfully in a number of games (Chung *et al.* 2005; Cazenave and Paris 2005; Cazenave 2009; Ward and Cowling 2009). The Upper Confidence Bound Tree (UCT) was introduced in (Kocsis and Szepesvri 2006) and spawned a host of research efforts (Gelly *et al.* 2006; Gelly and Wang 2006; Gelly and Silver 2007). Our work differs from previous UCT work in its use of key player subgroups to focus search in a multi-agent, continuous-value domain.

A key element of our approach is the use of automatically extracted coordination patterns from historical play data. In the Robocup soccer domain, (Iravani 2009) performed an in-depth analysis of interactions among players to identify multi-level networks. This is conceptually very closely related to the idea of grouping players by movement and workflow patterns. His system constructed agent interactions between players based on closest teammate, Voronoi regions and distance-based clusters. The identified networks were used to model the teams and identify how interactions affect team performance; however these networks were not incorporated into a play generation system.

Method

The basic idea behind our approach is to identify subgroups of coordinated players by observing a large number of football plays. Earlier work, such as (Laviers *et al.* 2009), has shown that appropriately changing the behavior of a critical subgroup (e.g., QB, RB, FB) during an offensive play, in response to a recognized defensive strategy, significantly improves yardage. Our work automatically determines the critical subgroups of players (for each play) by an analysis of spatio-temporal observations to determine all sub-groups, and supervised learning to learn which ones will garner the best results.

Identifying Key Players

In order to determine which players should be grouped together we first must understand dependencies among the eight players for each formation. All players coordinate to some extent but some players' actions are so tightly coupled that they form a *subgroup* during the given play. Changing the command for one athlete in a subgroup without adjusting the others causes the play to lose cohesion, potentially resulting in a yardage loss rather than a gain. We identify subgroups using a combination of two methods, the first based on a statistical analysis of player trajectories and the second on workflow.

The mutual information between two random variables measures their statistical dependence. Inspired by this, our method for identifying subgroups attempts to quantify the degree to which the trajectories of players are coupled, based on a set of observed instances of the given play. However, the naive instantiation of this idea, which simply computes the dependence between player trajectories without considering the game state is doomed to failure. This is because offensive players' motions are dominated by three factors: 1) its plan as specified by the playbook, 2) the current position of the ball, and 3) the current position of the defensive player assigned to block him.

So, if we want to calculate the relationships between the offensive players, we need to place their trajectories in a context that considers these factors. Our method for doing this is straightforward. Rather than computing statistics on raw player trajectories, we derive a feature that includes these factors and compute statistics between the feature vectors as follows.

First, for each player on the offense, we determine the trajectory of the defensive player assigned to block him. Since this assigned defensive player is typically the opponent that remains closest to the player during the course of the play, we determine the assigned defender to be the one whose average distance to the given player is the least. More formally, for a given offensive player, $o \in \{o_1, \dots, o_8\}$, the assigned defender, $d \in \{d_1, \dots, d_8\}$ is:

$$d = \underset{d_i}{\operatorname{argmin}} \sum_{t=1}^{T} |o(t) - d_i(t)|_2,$$

where $o(t)$ and $d_i(t)$ denote the 2D positions of the given players at time t. Our feature $f(t)$ is simply the centroid (average) of $o(t)$, $d(t)$ and the ball position $b(t)$. We can now compute sets of features $\{f_i\}$ and $\{f_j\}$ from the collection of observed plays for a given pair of offensive players o_i and o_j, treating observations through time simply as independent measurements. We model the distributions F_i and F_j of each of these features as 2D Gaussian distributions with diagonal covariance.

We then quantify the independence between these feature distributions using the symmetricized Kullback-Leibler divergence (Kullback and Leibler 1951):

$$S(o_i, o_j) = D_{KL}(F_i || F_j) + D_{KL}(F_j || F_i),$$

where

$$D_{KL}(F_i || F_j) = \sum_k F_i(k) \log \left(\frac{F_i(k)}{F_j(k)} \right).$$

Pairs of athletes with low $S(.)$ are those whose movements during a given play are closely coupled. We compute the average $S(.)$ score over all pairs (o_i, o_j) in the team and identify as candidate subgroups those pairs whose score falls in the lowest quartile.

The grouping process involves more than just finding the mutual information between players. We must also determine relationships formed based on possession of the football. When the quarterback hands the ball off to the running back or fullback their movements are coordinated for only a brief span of time before the ball is transferred to the next player. Because of this, the mutual information (MI) algorithm described above does not adequately capture this relationship. We developed another mechanism to identify such *workflows* and add them to the list of MI-based groups.

Our characterization of the workflow during a play is based on ball transitions. Given our dataset, we count transitions from one player to another. The historical data indicates that, in almost all offensive formations, the RB receives the ball the majority of the time, except when the FB is in play and in which case we see the ball typically passed from the QB to either the RB or the FB. Consequently, the $\{$QB, RB, and FB$\}$ naturally forms as a group for *running plays* which was identified in (Laviers *et al.* 2009) as a "key group". The same happens between the QB and the receiving player in *passing plays*, which forms another workflow group $\{$QB, LWR, RWR, and RTE$\}$. The final list of candidates is therefore simply the union of the MI candidates and the workflow candidates.

To use these subgroups we learn a prediction for the yardage impact of changing different extracted subgroups. For these studies, we compared the performance of several supervised classifiers and selected the K* instance-based algorithm, which is similar to Knn, but uses an entropy-based distance measure. To generate a training set, we ran the Rush 2008 football simulator on 450 randomly selected play variations. As the input features, we use the presence of possible observable offensive player actions (runningTo, carryingBall, waiting, charging, blocking, receivingPass, and sweeping); training is performed using 10-fold cross-validation.

Play Generation using UCT

We use Monte Carlo UCT (Kocsis and Szepesvri 2006) to generate new football plays. Using the top-ranked extracted subgroups to focus action investigations yields significant run-time reduction over a standard Monte-Carlo UCT implementation. To search the complete tree without using our subgroup selection method would require an estimated 50 days of processing time as opposed to the 4 days required by our method.

Offensive plays in the Rush 2008 football simulator share the same structure across all formations. Plays start with a **runTo** command which places a player at a strategic location to execute another play command. After the player arrives at this location, there is a decision point in the play structure where an offensive action can be executed. To effectively use a UCT style exploration we needed to devise a mechanism for combining these actions into a hierarchical tree structure where the most important choices are decided first.

Because of the potentially prohibitive number of possible location points, we have UCT initially search through the possible combinations of offensive high-level commands for the key players, even though chronologically the commands occur later in the play sequence. Once the commands are picked for the players, the system employs binary search to search the **runTo** area for each of the players (Figure 2). The system creates a bounding box around each players' historical **runTo** locations, and at level 2 (immediately after the high-level command is selected), the bounding box is split

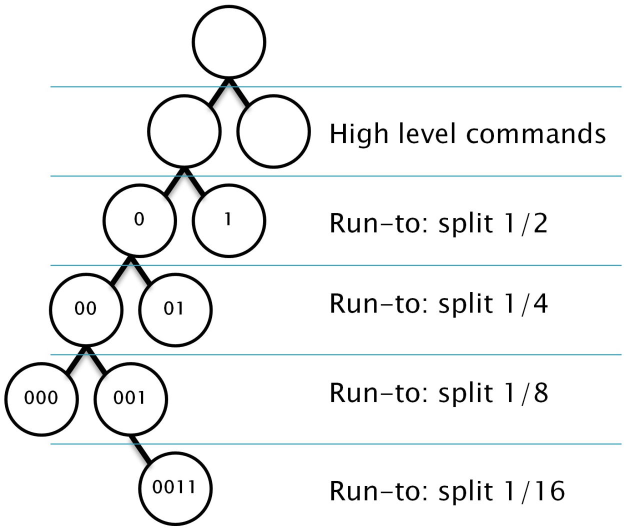
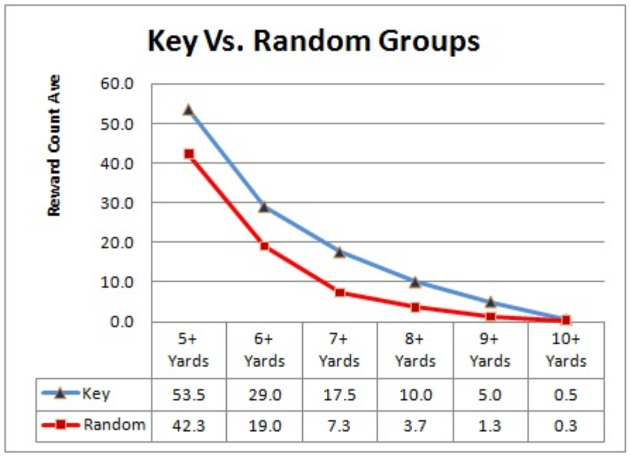


Figure 2: A representation of the UCT sparse tree.



	5+ Yards	6+ Yards	7+ Yards	8+ Yards	9+ Yards	10+ Yards
Key	53.5	29.0	17.5	10.0	5.0	0.5
Random	42.3	19.0	7.3	3.7	1.3	0.3

Figure 4: A detailed breakdown of the reward values above 5 yards. This chart clearly indicates that at the higher yardage spectrum the key groups provide a significantly greater number of solutions than the random groups.

in half. Following Monte Carlo expansion the location is initially randomly selected (Figure 5). At level 3 the space is again divided in half and the process continues until level 5 where the player is provided a **runTo** location which represents 1/16 of the bounding box area. The system takes a sample only at the leaf.

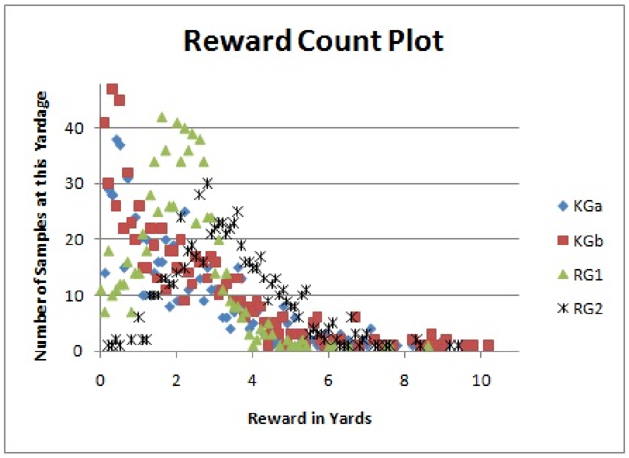


Figure 3: Comparison of randomly selecting players for action modifications vs. using the top-ranked subgroup. We see in the case of the random groups (R1 and R2) most of the reward values are close to the baseline of 2.8 yards while the key groups (K1 and K2) are dispersed more evenly across the yardage spectrum. This indicates that changing those players has a greater impact on the play.

This two dimensional search was designed to maintain as small a sampling as possible without harming the system's chance of finding solutions which produce large yardage gains. To focus the search, the locations each player can move to are bounded to be close (within 1 yard) to the region covered by the specific player in the training data. At the leaf node the centroid of the square is calculated and the player uses that location to execute the **runTo** command. Our method effectively allows the most important features to be searched first and the least important, last.

As mentioned, action modifications are limited to the players in the top ranked subgroup identified using K*; the other players execute commands from the original play. Our system needs to determine the best plan over a wide range of opponent defensive configurations. To do this, for each rollout the system randomly samples 50% of all possible defenses (evens or odds, one for testing and the other for training) and returns the average yardage gained in the sampling. Since UCT method provides a ranked search with the most likely solutions grouped near the start of the search, we limit the search algorithm to 1000 nodes with the expectation that a good solution will be found in this search space.

We perform action selection using a variant of the UCT formulation, $\pi(s,a) = \text{argmax}_a(Q^+(s,a))$, where π is the policy used to choose the best action a from state s. Before revisiting a node, each unexplored node from the same branch must have already been explored; selection of unexplored nodes is accomplished randomly. We demonstrate that it is important to correctly identify key players for the formation by examining the effect of randomly selecting players for action modification on the value of $Q(s,a)$ (Figure 3).

Using a similar modification to the bandit as suggested in (Balla and Fern 2009), we adjust the upper confidence calculation $Q^+(s,a) = Q(s,a) + c \times \sqrt{\frac{log\ n(s)}{n(s,a)}}$ to employ $c = Q(s,a) + .\varsigma$ where $\varsigma = .0001$ for our domain. The ς causes the system to consider nodes explored less often with zero reward more than nodes with a greater number of repeated zero rewards. Ultimately this allows us to account for the number of times a node is visited and prevent the search to remaining stuck on zero reward nodes, a problem

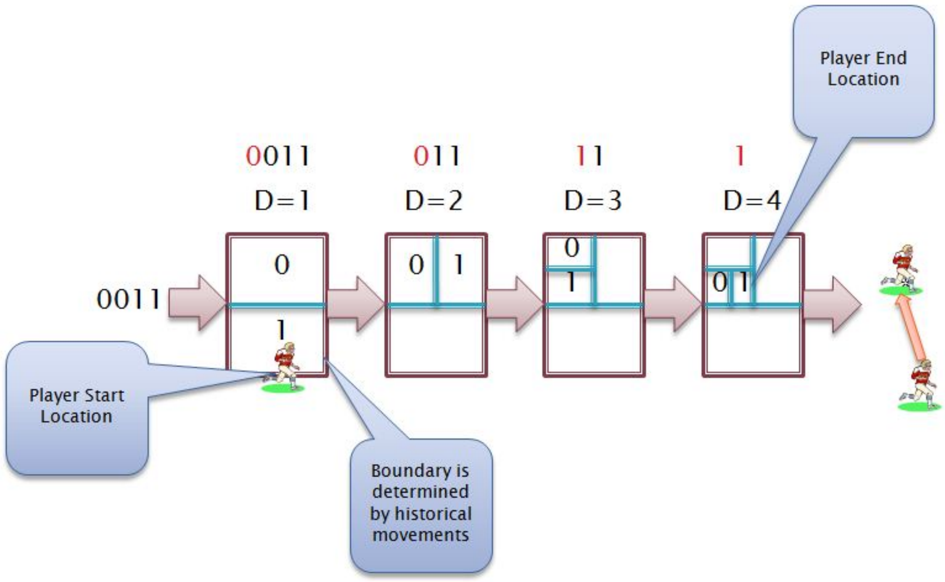


Figure 5: This graph shows how the binary string generated in the search tree creates a location for a player to move to in the **runTo** portion of the play.

we identified in early testing.

We implemented UCT in a distributed system constructed in such a way to prevent multiple threads from sampling the same node. The update function for $n(s, a)$ was modified to increment the counter after the node is visited, but before the leaf is sampled. Since sampling takes close to one second it's imperative to other threads exploring the tree to know when a node is touched to avoid infinite looping at a node.

After a node is sampled the update function is called to update $Q(s, a)$.

$$Q(s,a) \leftarrow Q(s,a) + \frac{1}{n(s,a)}\left(R - Q(s,a)\right)$$

and

$$R = \frac{\sum_{i=0}^{I} \gamma_i}{I} / 15$$

where R is the reward, I is the total number of iterations times the number of defenses sampled, and γ is the list of yards gained in each sample. We normalize R by dividing by 15 which is 30% more than the maximum unnormalized reward.

Results

We evaluated the efficacy of this approach at generating passing plays, which require tightly coupled coordination between multiple players to succeed. Our version of UCT was seeded with Pro formation variants (4–8). Figure 7 summarizes experiments comparing UCT (limited by subgroup) against the baseline Rush playbook and play adaptation. Overall, the UCT plays consistently outperform the baseline Rush system and play adaptation using domain knowledge. Viewing the output trace of one UCT search reveals some characteristics of our algorithm. First the system randomly explores, then as promising nodes are found they are exploited until UCT is confident it has a correct value for that branch before it moves on to other parts of the tree and repeats the process. Developers interested in automatically generating a list of plays can easily pick football plans which produce the level of difficulty the developer is interested in.

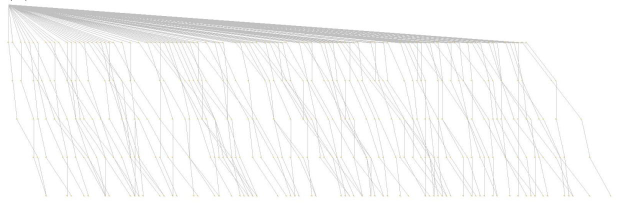
Conclusion

In this work, we presented a method for extracting subgroup coordination patterns from historical play data. We demonstrate our method in the Rush 2008 football simulator and believe that it can generalize to other team adversarial games, such as basketball and soccer. Although we have access to the ground truth playbook information executed by the simulator, the Rush plays do not explicitly specify subgroup coordination patterns between players. The subgroup patterns seem to emerge from the interactions between offensive and defensive formations creating different player openings which in turn affect play workflows when the quarterback chooses to pass or hand off to different players. We demonstrate that we can identify and reuse coordination patterns to focus our search over the space of multi-agent policies, without exhaustively searching the set partition of player subgroups. By sorting our candidate subgroups using a ranking learned by K*, we can reliably identify effective subgroups and improve the performance and speed of the UCT Monte Carlo search.

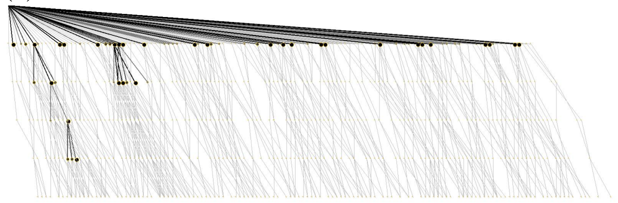
Acknowledgments

This work was supported in part by DARPA grant HR0011-09-1-0023.

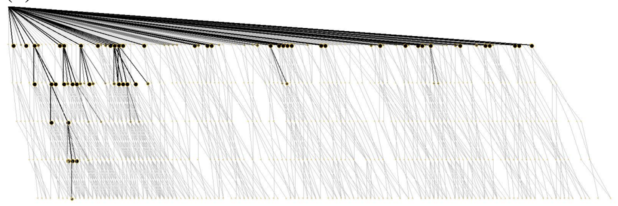
(a) At 100 iterations



(b) At 300 iterations



(c) At 500 iterations



(d) At 700 iterations

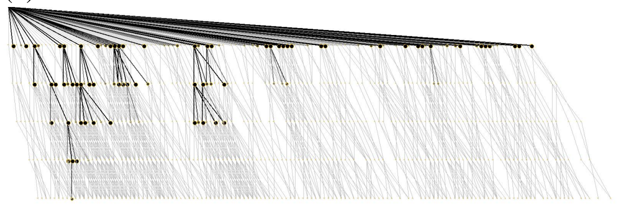


Figure 6: Our UCT variant expands in a very focused direction and quickly identifies a high yardage area of the tree.

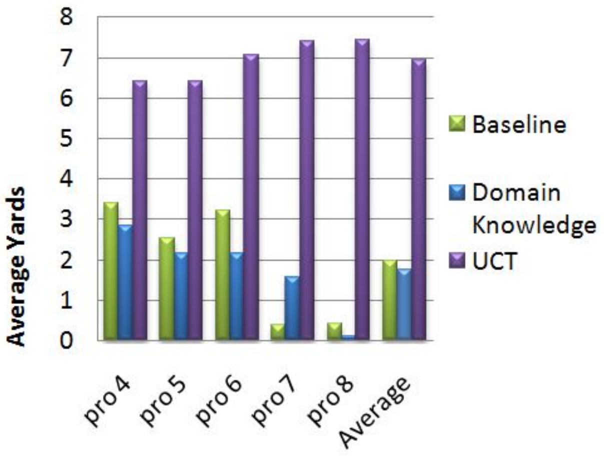


Figure 7: Comparison of multi-agent policy generation methods starting in the Pro formation. Our variant of UCT outperforms the baseline playbook and the domain knowledge play adaptation method.

References

R. Balla and A. Fern. UCT for tactical assault planning in real-time strategy games. In *Proceedings of International Joint Conference on Artficial Intelligence*, 2009.

Tristan Cazenave and Labo Ia Universit Paris. Combining tactical search and Monte-Carlo in the game of Go. In *Proceedings of IEEE Symposium on Computational Intelligence and Games*, pages 171–175, 2005.

Tristan Cazenave. Nested Monte-Carlo search. In *Proceedings of the International Joint Conference on Artifical Intelligence*, pages 456–461, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

Michael Chung, Michael Buro, and Jonathan Schaeffer. Monte Carlo planning in RTS games. In *Proceedings of IEEE Symposium on Computational Intelligence and Games*, 2005.

Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *In Zoubin Ghahramani, editor, Proceedings of the International Conference of Machine Learning (ICML 2007*, pages 273–280, 2007.

Sylvain Gelly and Yizao Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go, December 2006.

Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with Patterns in Monte-Carlo Go. Research Report RR-6062, INRIA, 2006.

P. Iravani. Multi-level network analysis of multi-agent systems. *RoboCup 2008: Robot Soccer World Cup XII*, pages 495–506, 2009.

Levente Kocsis and Csaba Szepesvri. Bandit based Monte-Carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

S. Kullback and R. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

K. Laviers, G. Sukthankar, M. Molineaux, and D. Aha. Improving offensive performance through opponent modeling. In *Procceedings of Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE)*, 2009.

N. Li, D. Stracuzzi, G. Cleveland, P. Langley, T. Konik, D. Shapiro, K. Ali, M. Molineaux, and D. Aha. Constructing game agents from video of human behavior. In *Proceedings of Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2009.

M. Molineaux, D. Aha, and G. Sukthankar. Beating the defense: Using plan recognition to inform learning agents. In *Proceedings of Florida Artifical Intelligence Research Society*, 2009.

C. D. Ward and P. I. Cowling. Monte Carlo search applied to card selection in Magic:The Gathering. In *Proceedings of IEEE Symposium on Computational Intelligence and Games*, pages 9–16, Piscataway, NJ, USA, 2009. IEEE Press.