A Real-Time Opponent Modeling System for Rush Football

Kennard Laviers
Department of EECS
University of Central Florida
Orlando, FL
klaviers@knights.ucf.edu

Gita Sukthankar
Department of EECS
University of Central Florida
Orlando, FL
gitars@eecs.ucf.edu

Abstract

One drawback with using plan recognition in adversarial games is that often players must commit to a plan before it is possible to infer the opponent's intentions. In such cases, it is valuable to couple plan recognition with plan repair, particularly in multi-agent domains where complete replanning is not computationally feasible. This paper presents a method for learning plan repair policies in real-time using Upper Confidence Bounds applied to Trees (UCT). We demonstrate how these policies can be coupled with plan recognition in an American football game (Rush 2008) to create an autonomous offensive team capable of responding to unexpected changes in defensive strategy. Our real-time version of UCT learns play modifications that result in a significantly higher average yardage and fewer interceptions than either the baseline game or domain-specific heuristics. Although it is possible to use the actual game simulator to measure reward offline, to execute UCT in real-time demands a different approach; here we describe two modules for reusing data from offline UCT searches to learn accurate state and reward estimators.

1 Introduction

Although effective opponent modeling is often identified as an important prerequisite for building agents in adversarial domains [van den Herik *et al.*, 2005], research efforts have focused mainly on the problem of fast and accurate plan recognition [Avrahami-Zilberbrand and Kaminka, 2005; Kabanza *et al.*, 2010]. Often in continuous-action games the information from plan recognition is used in an ad-hoc way to modify the agent's response, particularly when the agent's best response is relatively obvious. In this paper, we propose that coupling *plan recognition* with *plan repair* can be a powerful combination, particularly in multi-agent domains where replanning from scratch is difficult to do in real-time.

To explore this problem, we chose a popular domain where planning is crucial and accurate plan recognition is possible—American football [Intille and Bobick, 1999; Hess *et al.*, 2007]. Our goal is to produce a challenging and fun computer player for the Rush 2008 football game developed by Knexus Research [Molineaux, 2008], capable of responding to a human player in novel and unexpected ways.[1] In Rush 2008, 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. Planning is accomplished before a play is enacted, and the best plays are cached in a playbook. Certain defensive plays can effectively counter specific offenses. Once the play commences, it is possible to recognize the defensive play and to anticipate the imminent failure of the offensive play.

In such situations, we propose that plan repair can be used to mitigate poor expected future performance. Paradoxically, plan repair can easily worsen overall play performance by causing miscoordinations between players; even minor timing errors can significantly compromise the efficacy of a play. Moreover, it is difficult to predict future play performance at intermediate stages of the play execution since effective and ineffective plays share many superficial similarities. In this paper, we introduce an approach for learning effective plan repairs using a real-time version of Upper Confidence Bounds applied to Trees (UCT) [Kocsis and Szepesvári, 2006]. Figure 1 provides an overview of the key elements of our implementation. Our system is the first autonomous game player capable of learning team plan repairs in real-time to counter predicted opponent actions.

Monte Carlo search algorithms have been successfully used in games that have large search spaces [Chung *et al.*, 2005; Cazenave and Helmstetter, 2005; Cazenave, 2009; Ward and Cowling, 2009]. Upper Confidence Bounds applied to Trees (UCT) is one such method that performs Monte Carlo rollouts of a complete game from the current state. Prior work on UCT for multi-agent games has either relied on hand-coded game simulations [Balla and Fern, 2009] or the use of the actual game to evaluate rollouts. In this paper, we illustrate how data from offline UCT searches can be used to learn state and reward estimators capable of making limited predictions of future actions and play outcomes. Our UCT search procedure uses these estimators to calculate successor states and rewards in real-time. Experiments show that

[1] Note that we are not attempting to realistically simulate the thought processes or actions of human players and coaches.

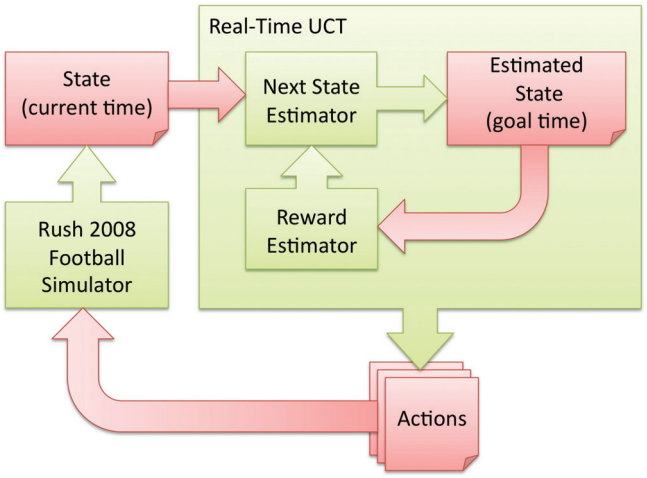

Figure 1: High-level diagram of our system. To run in real-time, our variant of UCT uses a successor state estimator learned from offline traces to calculate the effects of random rollouts. The reward is estimated from the projected terminal state (just before the quarterback is expected to throw the ball, designated in the diagram as the goal time.)

the plan repairs learned by our method offer significant improvements over the offensive plays executed by the baseline (non-AI system) and also a heuristic-based repair method.

2 Rush Football

American football is a contest of two teams played on a rectangular field. Games like soccer and football pose a different set of planning problems than are found in static, turn-based games such as chess or Go which only utilize one agent per color. Football requires real-time, continuous, multi-agent search techniques capable of handling a dynamic environment. Unlike standard American football, Rush teams only have 8 players simultaneously on the field out of a total roster of 18 players, and the field is 100×63 yards. The game's objective is to out-score the opponent, 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. Stochasticity exists in many aspects of the game including throwing, catching, fumbling, blocking, running, and tackling. Our work focuses on improving offensive team performance in executing passing plays.

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. Rush includes three offensive formations and four defensive ones. Each formation has eight different plays that can be executed from that

Figure 2: The Pro formation running play variant 4 against defensive formation 31 running variant 2.

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. Learning effective plays in Rush is hampered by: (1) a large and partly continuous search space, (2) lack of intermediate reward information, (3) difficulty in identifying action combinations that yield effective team coordination, and (4) constant threat of actions being thwarted by adversaries. Figure 2 shows an example play from the **Pro** formation:

1. the quarterback passes to an open receiver;
2. the running back and left wide receiver run hook routes;
3. the left and right guards pass block for the ball holder;
4. the other players wait.

3 Method

Our system for learning plan repairs in real-time relies on the following components.

Play Recognizer We treat the problem of intention recognition as a multi-class classification problem to identify the formation and play variant currently being executed by the defense. Although recognizing the static formation is straightforward, early recognition of play variants is challenging. We achieve 90% accuracy at time $t = 3$ using a multi-class support vector machine (SVM).

Next State Estimator To execute UCT rollouts in real-time our system must predict how defensive players will react as the offense adjusts its play. We train state/reward estimators using offline data from previous UCT searches and employ them in real-time.

Reward Estimator To calculate UCT Q-values in the predicted future state, the system estimates reward (yardage) based on relative positions of the players. Because of the inherent stochasticity of the domain, it is difficult to learn a reward estimator early in the play. We focus on estimating yardage at a later stage of the play—just before we expect the quarterback to throw the ball.

UCT Search Using the state and reward estimators, we use the UCT search algorithm to generate a sparse tree to

select actions for the key offensive players, a three player subset of the team automatically determined in advance. The search procedure is re-executed at every time step to account for unexpected actions taken by the defense.

Rush Simulator The selected player actions are issued to the Rush simulator via network sockets. The simulator returns the new locations of all offensive and defensive players to be used by the estimators.

3.1 UCT Search

After recognizing the play, we use UCT to search for the best action available to each of the key players. Key players are a subset of three offensive players identified offline for a specific formation. UCT maximizes the upper-confidence bound by preferentially exploring regions with the best probability of producing a high reward. Search continues for a predetermined number of rollouts; our implementation uses $N = 500$, which produced the best results while still allowing real-time execution. As suggested by the original UCT algorithm, unexplored nodes are always sampled first and at random. When no nodes from the current node are left unexplored, action selection is determined using a variant of the UCT formulation, $\pi(s) = \text{argmax}_a(Q^+(s,a))$, where π is the policy. The state s includes the locations of all players as well as the location of the ball. The action a contains the combination of actions for key players, $a = \{a_1, a_2, a_3\}$, where $a_{1,2,3} \in \{\text{Left}, \text{upLeft}, \ldots, \text{downLeft}\}$. Sampling continues until the predetermined number of samples N is reached. Finally, the action leading to the most frequently sampled child of the root node is selected as the next action.

Using a similar modification to that suggested in [Balla and Fern, 2009], we adjust the upper confidence calculation $Q^+(s,a) = Q(s,a) + c\sqrt{\frac{log\ n(s)}{n(s,a)}}$ to employ $c = Q(s,a)$. Typically c is a constant used to tune the biasing sequence to adjust exploration/exploitation of the search space. We used a modification proposed by [Balla and Fern, 2009] which allows the amount of exploration to scale proportionally to the quality function $Q(s,a)$, which ranges from 0 to 1. After a node is sampled, both $Q(s,a)$, the mean observed reward, and $n(s,a)$, the number of times the node is sampled are updated. This update occurs recursively from the leaf node to the root node:

$$n(s,a) \leftarrow n(s,a) + 1,$$

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

where R' is the reward given by the reward estimator.

For this spatial search problem, if actions are explored randomly, players will remain within a small radius of their starting positions. Even in conjunction with UCT, it is unlikely to find a good path. To eliminate circular travel, the system uses an attractive potential field [Arkin, 1989] in the direction of the goal that guides exploration toward the correct end zone.

To improve the efficiency of the search process, we constrain the Monte Carlo rollouts in the following ways. First, we only consider movement actions in the general direction of the player's potential field. Also, for every offensive formation, plan repairs are limited to a small subgroup of key players; the remaining players continue executing the original offensive play. The initial configuration of the players governs the players that are most likely to have a decisive impact on the play's success; by focusing search on a key subgroup of these three players (out of the total team of eight) we speed the search process significantly and concentrate the rollouts on higher expected reward regions. In the results section, we separately evaluate the contribution of these heuristics toward selecting effective plan repairs.

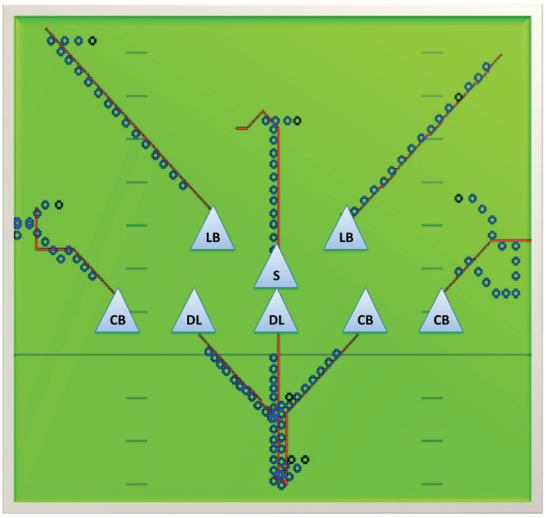

Figure 3: A comparison between the actual and predicted paths. Actual paths made by the defensive players are shown as red lines. Blue circles show the estimated path of the defensive players. The estimated motions are very close to the actual paths except in the case of the right CB, where the estimated path crosses but does not follow the true path.

3.2 Successor State Estimation

To predict successor states in real-time, we perform an incremental determination of where each player on the field could be at the next time-step. To accomplish this update, players are split into three groups: (1) defensive players, (2) offensive key players, and (3) offensive non-key players. The real-time UCT algorithm explores actions by the key players, and the successor state estimator seeks to predict how the defensive players will react to potential plan repairs. Locations of non-key offensive players are determined using the historical observation database to determine the most likely position each non key offensive player will occupy, given the play variant and time-step. Rather than executing individual movement stride commands, these players are actually performing high-level behaviors built into the Rush simulator; thus even though these players are technically under our control, we cannot predict with absolute certainty where they will be in the future.

Formally, the game state at time t can be expressed as the vector $\mathbf{s}(t) = (\mathbf{x}_{o1}, \ldots, \mathbf{x}_{o8}, \mathbf{x}_{d1}, \ldots, \mathbf{x}_{d8}, \mathbf{x}_b)$, where $\mathbf{x}_{oi}$, $\mathbf{x}_{dj}$, and $\mathbf{x}_b$ denote the (x,y) positions of the offensive players, defensive players, and the ball, respectively. Similarly, we denote by a_{oi} and a_{dj} the actions taken by the offensive

player oi and defensive player dj, respectively and a_b denotes the movement of the ball.

We predict the actions for the non-key offensive players from the historical archive of previously observed games; we simply advance the play according to the most likely action for each player and adjust the ball state accordingly. However, to determine the actions for the key offensive players (those whose actions will dramatically alter the current play), we identify promising actions from the UCT tree using the Monte-Carlo rollout. The goal is to alter the current play in a way that improves the expected yardage.

Predicting the opponent's response to the altered play is more difficult. For this, we train a classifier to predict the next action of each defensive player dj based on its position and that of its closest offensive player,

$$o\varphi = \arg\min_{oi} ||\mathbf{x}_{dj} - \mathbf{x}_{oi}||_2.$$

In other words, the classifier learns the mapping:

$$(\mathbf{x}_{dj}(t), \mathbf{x}_{o\varphi}(t)) \mapsto a_{dj}(t+1),$$

where $a \in \mathcal{A}$ is selected from the discrete set of actions described above. We employ the K^* classifier [Wang *et al.*, 2006] for this purpose. Applying a_{dj} to the defensive player's position enables us to predict its future position. The classifier is trained off-line using a set of observed plays and is executed on-line in real-time to predict actions of defensive players.

We predict the play state forward up to the time τ where we expect the quarterback to throw the ball. If by $t = \tau$, the quarterback has not thrown the ball, we continue predicting for five more time steps.

We evaluated the successor state estimator using 1177 test instances and found that it accurately predicts the next state for each defensive player 88.63% of the time. Note that this is an example of an unbalanced dataset in which certain actions are extremely rare. Figure 3 shows the trajectories generated using the successor state estimator compared to the actual positions of the defensive players.

3.3 Reward Estimation

The reward estimator is trained using examples of player configurations immediately preceding the quarterback throw. At this stage of the play, there is significantly less variability in the outcome than if we attempted to train a reward estimator based on earlier points in the play execution.

The reward estimator uses an input vector derived from the game state at the end of the prediction $\mathbf{s}(\tau)$ consisting of a concatenation of the following three attributes: 1) distance of the ball to each key offensive player; 2) distance from each key offensive player to the end zone; 3) distance from each key offensive player to his closest opponent.

The output is the expected yardage, quantized into 6 equally-sized bins. Our preliminary evaluations indicated that learning a continuous regression model for the yardage was much slower and did not improve accuracy. Therefore, we use a K^* classifier with the expected yardage treated as a discrete class (1–6).

We performed a 10-fold cross validation to validate the effectiveness of the reward estimator. The estimator was correct in 54.7% of the instances. Since different executions from the same player positions can result in drastically different outcomes, accurately estimating reward is a non-trivial problem. Improving the classification accuracy could potentially improve the effectiveness of our system but even with our current reward estimator, the focused UCT search is able to identify promising plan repairs.

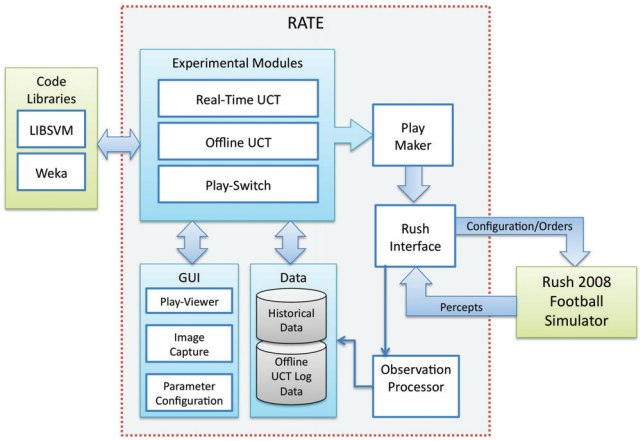

Figure 4: Our Rush Analyzer and Test Environment (RATE) is designed to facilitate reproducible research on planning and learning in the Rush 2008 football game. RATE consists of over 40,000 lines of code and has support for separate debugging of AI subsystems, parallel search, and point-and-click configuration.

4 Results

To demonstrate the effectiveness of the overall system, we compared the plans generated by the proposed method against the unmodified Rush 2008 engine (termed "baseline") and against a heuristic plan repair system that selects a legal repair action (with uniform probability) from the available set, using potential field and key player heuristics. Experiments were conducted using our Rush Analyzer and Test Environment (RATE) system, shown in Figure 4, which we constructed to support experimentation on planning and learning in Rush 2008. Because of the time requirements to connect sockets and perform file operations RATE operates as a multi-threaded application. Results in Figure 5 are shown for the fourth play variant of the Pro formation (a passing play) against 3 randomly selected defenses which gain fewer than 6.5 yards on average. Three defensive variants from different formations (31-2, 2222-2, and 2231-2) were arbitrarily chosen for testing. A two-tailed student t-test reveals that our approach (real-time UCT) outperforms both the baseline and heuristic approaches ($p < 0.01$) on total yardage gained.

In American football one of the worst outcomes is that the team in possession of the ball inadvertently passes it to the opposing team. These changes in ball possession have a significant impact on the future course of the game that is not reflected in single play matchups. Thus, we separately evaluated how frequently the defense succeeded at intercepting the ball independent of yardage gains (Figure 6). Our ap-

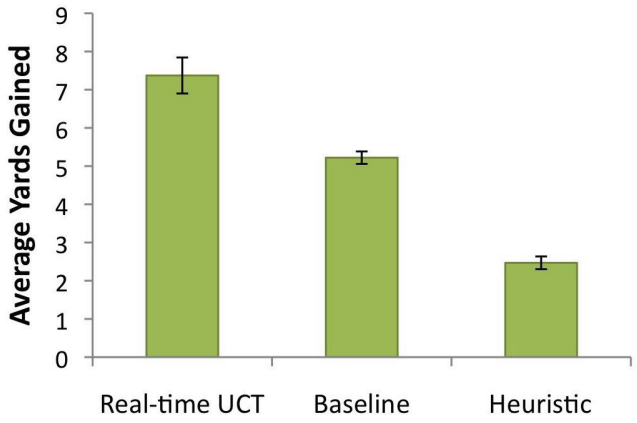

Figure 5: The proposed system with learned successor and reward state estimators results in a performance gain of almost 5 yards (67% improvement) over uniformly-drawn repair actions guided by the key player and potential field heuristics. Compared to the baseline (Rush 2008 using built-in play library) there is a 2 yard or 30% performance improvement. The error bars mark the 95% confidence interval.

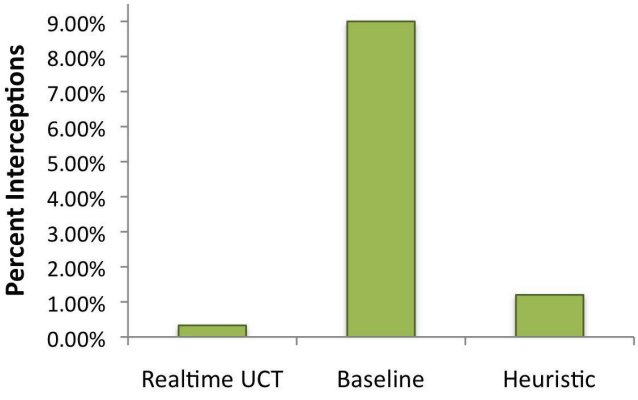

Figure 6: A key metric in American football is how frequently the opposing team is able to intercept the ball (lower is better). The proposed system improves significantly over both the baseline and heuristic approaches.

proach significantly reduces the number of interceptions over both the baseline Rush simulator and the uniformly-drawn plan repair actions, showing that coupling plan recognition with effective plan repair dramatically improves the offense's chance of maintaining possession of the ball. However, it appears that even the heuristically-guided plan repair reduces the number of interceptions, revealing that even imperfect plan repair can decisively impact the course of the game in a way not well-reflected by the yardage metric.

5 Related Work

Like Robocup, Rush 2008 was developed as a general platform for evaluating game-playing agents. Interestingly, one of the first mentions of opponent modeling in the AI literature pertains to predicting movements in football [McCarthy and Hayes, 1969]. In our work, we treat opponent modeling as a specialized version of online team behavior recognition in which our system solves a multi-class classification problem to identify the currently executing play. There has been prior work on offline play generation for Rush Football using techniques such as learning by demonstration (e.g., [Li *et al.*, 2009]). None of the previous approaches included a real-time planning component; plays were generated offline and loaded into the simulator. Accurate supervised and unsupervised plan recognition techniques have also been demonstrated within Rush 2008 [Molineaux *et al.*, 2009], where the authors showed the benefits of combining plan recognition with reinforcement learning. The defensive play information is directly included into the state space representation used by the reinforcement learner. Unlike our work, their method focuses exclusively on the quarterback, and all learning takes place offline. [Laviers *et al.*, 2009] present a non-learning plan adaptation method for Rush 2008, in which key players switch to a new play based on historical yardage gains. The effectiveness of this method is limited by the initial play library since there is no mechanism for discovering new repair sequences.

Within the Robocup domain, which shares some similarities with Rush 2008, the bulk of related work has been in the context of the Coach Agent competition (e.g., [Riley and Veloso, 2002]). Outside of the coach competition, most of the work on behavior recognition on Robocup has been theoretical in nature (e.g., [Avrahami-Zilberbrand and Kaminka, 2005]) and not usable in actual play; in contrast, our work conclusively demonstrates that play recognition can be used effectively in real-time.

Recently, UCT has enjoyed great success at learning policies in a wide-variety of games. Most relevant to our efforts, UCT has been demonstrated in WARGUS, a multi-agent real-time strategy game (RTS) by [Balla and Fern, 2009]. In general, our work differs from prior work using UCT in that it focuses on learning *plan repairs* rather than learning *plans*.

6 Conclusion

A missing ingredient in effective opponent modeling for games is the ability to couple plan recognition with plan repair. In this paper, we propose a real-time method for learning plan repair policies and show that it is possible to learn successor state and reward estimators from previous searches to do online multi-agent Monte Carlo rollouts. Simultaneously predicting the movement trajectories, future reward, and play strategies of multiple players in real-time is a daunting task but we illustrate how it is possible to divide and conquer this problem with an assortment of data-driven game models. Our learned plan repair policies outperform both the baseline system and a simple heuristics-based plan repair method at improving yardage gained on each play execution. More importantly, our method results in a dramatic drop in the number of interceptions, which is likely to result in significantly longer ball possession periods within the context of a full game. Although the details of the learning process may differ, we believe that our techniques will generalize to other real-time, continuous, multi-agent games that lack intermediate reward information such as squad-based shooter games.

In future work, we plan to incorporate information about classifier error rates directly into our plan repair to enable the calculation of "risk-sensitive" plan repair policies that consider the impact of prediction failures. In this particular domain, our play recognizer classifies plays with a $> 90\%$ accuracy so this feature was not a priority. An unanswered question is the long-term effect of plan repair in computer opponents on player enjoyability. Our hypothesis is that adding plan repair increases the variability of the game execution and results in an overall increase in player satisfaction based on the theory espoused by [Wray and Laird, 2003]. However, it is possible that the plan repair algorithm needs to be tuned to provide play at the correct difficulty level rather than simply optimized to be maximally effective; studying the question of adaptive difficulty is an area of future research.

7 Acknowledgments

This work was supported in part by DARPA award N10AP20027. We gratefully acknowledge Matt Molineaux and David Aha for their help and development efforts on Rush 2008.

References

[Arkin, 1989] R. Arkin. Motor schema – based mobile robot navigation. *The International Journal of Robotics Research*, 8(4):92–112, 1989.

[Avrahami-Zilberbrand and Kaminka, 2005] D. Avrahami-Zilberbrand and G. Kaminka. Fast and complete symbolic plan recognition. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2005.

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

[Cazenave and Helmstetter, 2005] T. Cazenave and B. Helmstetter. Combining tactical search and Monte-Carlo in the game of Go. In *Processings of Computational Intelligene in Games*, pages 171–175, 2005.

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

[Chung *et al.*, 2005] M. Chung, M. Buro, and J. Schaeffer. Monte Carlo planning in RTS games. In *CIG*, 2005.

[Hess *et al.*, 2007] R. Hess, A. Fern, and E. Mortensen. Mixture-of-parts pictorial structures for objects with variable part sets. In *Proceedings of International Conference on Computer Vision*, 2007.

[Intille and Bobick, 1999] S. Intille and A. Bobick. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of National Conference on Artificial Intelligence*, 1999.

[Kabanza *et al.*, 2010] F. Kabanza, P. Bellefeuille, F. Bisson, A. Benaskeur, and H. Irandoust. Opponent behaviour recognition for real-time strategy games. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition*, 2010.

[Kocsis and Szepesvári, 2006] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293. Springer, 2006.

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

[Li *et al.*, 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.

[McCarthy and Hayes, 1969] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, pages 463–502. Edinburgh University Press, 1969.

[Molineaux *et al.*, 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.

[Molineaux, 2008] M. Molineaux. Working Specification for Rush 2008 Interface. Technical report, Knexus Research Corp. May 2008, 2008.

[Riley and Veloso, 2002] P. Riley and M. Veloso. Recognizing probabilistic opponent movement models. In A. Birk, S. Coradeschi, and S. Tadorokoro, editors, *RoboCup-2001: Robot Soccer World Cup V*. Springer Verlag, 2002.

[van den Herik *et al.*, 2005] J. van den Herik, J. Donkers, and P. Spronck. Opponent modeling and commercial games. In *Procceedings of the Sympoisum on Computational Intelligence and Games*, 2005.

[Wang *et al.*, 2006] H. Wang, H. Zheng, D. Simpson, and F. Azuaje. Machine learning approaches to supporting the identification of photoreceptor-enriched genes based on expression data. *BMC Bioinformatics*, 7(1):116, 2006.

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

[Wray and Laird, 2003] R. Wray and J. Laird. Variability in human behavior modeling for military simulations. In *Proceedings of Behavior Representation in Modeling and Simulation Conference (BRIMS)*, 2003.