Optimal and Approximate Policies for Winning Timed, Zero-Sum Games

Colin McMillen
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
mcmillen@cs.cmu.edu

February 27, 2007

Thesis Proposal

Committee:
Manuela Veloso (chair)
Drew Bagnell
Stephen Smith
Michael Littman (Rutgers University)

Abstract

Adversarial multiagent systems offer a wide variety of challenges. In this thesis, I consider *timed, zero-sum games*, which have a time limit and a win/loss outcome. In these games, each agent needs to decide how *to win*; i.e., how to be ahead of the opponent in score when the time of the game runs out. This work is applicable to adversarial domains with dynamic environments and uncertainty in the effects of actions, such as robot soccer, iterated gambling games, computer and video games, and other similar scenarios.

I intend to address a challenging online learning problem: the agent plays an unknown opponent only once in a game with fixed duration and wants to win. The outcome of the game develops incrementally as the game progresses; i.e., as goals or points are scored. The agent must learn how to win during the course of that single game using reward signals it can measure online during the game. I am particularly interested in using the amount of time remaining, the current score of the game, and knowledge of the opponent's performance as such reward signals. I propose to investigate how an agent can learn to win against an unknown opponent in this online fashion.

In a timed, zero-sum game, our agent's goal is to maximize the probability of winning. In general, the optimal zero-sum solution requires a policy that depends on the score, the time remaining, and the opponent's strategy. In this thesis, I consider two cases of opponent dynamics: *static opponents*, whose policies do not depend on the score or time remaining; and *dynamic opponents*, whose policies depend on the complete game state, including score and time remaining.

My previous work has addressed static opponents. The optimal policy for static opponents ("SO-optimal") can be computed by treating the domain as a Markov decision process (MDP) whose states encode all possible values of score difference and time remaining. However, the computation of the SO-optimal policy can be computationally expensive, since the number of states grows as a function of score and the degree of time discretization. An approach adopted in practice is to simplify the domain by omitting time and score from the model and attempting to maximize expected rewards in this simplified model. This stationary, score-independent ("SSI") solution is suboptimal because it does not explicitly reason about *winning*—being ahead of the opponent, with high probability, when time runs out.

In between the extremes of SO-optimal and SSI, there are many interesting approximations which trade off solution time with solution quality. In my initial thesis research, I have developed heuristic solution techniques that aggregate states based on the time remaining. These heuristics allow us to efficiently find approximate solutions for timed, zero-sum games with large state spaces or long time horizons. In addition to these time-based heuristics, there are many other techniques that can potentially allow for efficient approximate solutions to timed, zero-sum games. I investigate the losses incurred by using the SSI approximation and contrast this with the SO-optimal solution and the other approximations developed in this thesis.

For my thesis, I propose to continue researching efficient solution techniques for timed, zero-sum games and to apply these techniques to challenging adversarial domains such as robot soccer. I also intend to address the problem of dynamic opponents. Specifically, I intend to contribute:

- Application of efficient MDP solution techniques to timed, zero-sum MDPs, including approximation algorithms.
- Heuristic approximation techniques for solving timed, zero-sum MDPs as a function of state space, time horizon length, and score.
- Investigation of the effect of time horizon length and time discretization on zero-sum performance.
- Algorithms for online reasoning about dynamic opponent behavior in a finite-horizon environment.
- Application of all these techniques to a complex adversarial domain such as robot soccer.

Contents

1 Introduction

1.1 Motivation

I propose to address the problem of agents acting effectively in finite-horizon, stochastic, adversarial environments. My work is motivated by zero-sum games with score and limited time; in particular, robot soccer. In timed, zero-sum games, winning against the opponent is more important than the final score. Intuitively, it seems that a team which is losing near the end of the game should play aggressively, trying to even the score before time runs out. Such a team can be said to be *risk-taking*: willing to trade expected reward for a higher probability of winning (or at least tying) the game. Conversely, a team which is winning near the end of the game may want to play a *risk-averse* strategy, playing defensively to prevent the opponent from scoring. These sorts of strategies are used widely in human sports. In previous work, I have discussed how a team of soccer-playing robots can change *plays* (high-level team strategies) based on factors such as the time remaining in a game and the score difference [McMillen et al., 2005, McMillen and Veloso, 2006b]. However, this strategy selection was hand-tuned, using simple rules such as, "If our team is losing and there is less than one minute remaining, play aggressively." [Stone, 1998] The work proposed for this thesis will allow the derivation of optimal and approximately-optimal strategy selections for timed, zero-sum games such as robot soccer.

1.2 Dimensions of Reasoning in Uncertain Environments

Robot soccer has a number of features that make it a truly challenging testbed for artificial intelligence and robotics research. However, I am also interested in the more general problem of reasoning in other finite-horizon, uncertain environments. Many interesting domains preserve the finite-horizon aspect of robot soccer while differing in other areas. In this section, I consider several dimensions of problem difficulty that are commonly studied in the AI literature. Through this analysis, the difficulties of the robot soccer domain will be made clear. I propose to tackle the full robot soccer problem by first studying relatively simple domains. From this starting point, I intend to build domains of increasing complexity, using the results of previous steps to gain the insight required to solve complex, uncertain domains.

Uncertainty can enter a domain in many different ways. We concern ourselves here with four major areas of uncertainty: action noise, observations, unknown environments, and multiple agents.

- **Action noise.** Agents' actions can be *deterministic* or *stochastic* (also called *noisy*). In a deterministic environment, given the current state and the action chosen by the agent, the successor state is known with certainty. In a stochastic environment, action effects are uncertain, leading to a probability distribution over successor states.

- **Observations.** In a *fully observable* environment, the agent always knows the exact state of the world. In a *partially observable* environment, the agent is not given the state directly, but receives observations that may allow it to infer the state of the world. Some environments may be *unobservable*, in which case the agent has no way of inferring anything about the world state.

- **Unknown environments.** In a known environment, we assume that the agent is given an accurate model of the environment dynamics *a priori* (usually specified by the system designer). In unknown environments, the agent is not provided with such a model, and therefore needs to learn something about the environment to act in a reasonable manner.

- **Multiple agents.** Is a single agent operating in the environment, or multiple agents? Optimal reasoning is much more difficult in a multi-agent environment, as the effects of the other agents need to be

considered. In particular, in an *adversarial* domain, the other agents are actively working against our agent's interests.

Figure 1 shows some common AI problems that are related to reasoning under uncertainty, and how they fit in to these dimensions. (The figure is meant to be illustrative, not exhaustive, so not all combinations are shown.)

# Agents	Environment	Actions	Observations	Problem Type
single	known	deterministic	full	classical planning
single	known	stochastic	full	MDPs
single	known	stochastic	partial	POMDPs
single	unknown	stochastic	full/partial	reinforcement learning
multi	known	deterministic	full	matrix games
multi	known	stochastic	full	stochastic games
multi	known	stochastic	partial	POSGs
multi	unknown	stochastic	full/partial	multiagent RL

Figure 1: Dimensions of reasoning in uncertain environments.

For my thesis, I am interested in exploring domains with *stochastic actions* and *full observability*. Robots in domains such as RoboCup experience significant actuator noise, so the ability to account for stochastic actions is extremely important. Robots also experience sensor noise (partial observability), but for the high-level strategic problems I am interested in, I make the assumption that the error of low-level sensors will not significantly impact our action choices. I believe this is justified because previous experience with the RoboCup domain indicates that the most useful features for strategy selection are at a high level, such as the score of the game, the time remaining, the global position of the ball, the position of our teammates, and so on. These features are not often noisy at a global level. I am therefore primarily interested in domains which involve stochastic actions and full observability.

1.3 Optimal and Approximate Zero-Sum Solutions

In MDPs, stochastic games, and reinforcement learning, agents take actions in an environment and receive *reward*—a scalar signal that the agent is acting to maximize, according to some optimality criterion. Optimality criteria from the literature are discussed in detail in Section 2.1.2; the most commonly used optimality criteria are *expected cumulative reward* and *expected cumulative discounted reward*.

I am primarily interested in finite-horizon, zero-sum games, in which every point of reward scored by one agent is a point against the other agent. The goal in such a game is to maximize the probability of winning. In these domains, let *intermediate reward* be the difference between our agent's score and our opponent's score. At the end of the time horizon h, we say that our agent *wins* if it has achieved positive intermediate reward, *loses* if it has achieved negative intermediate reward, and *ties* if its intermediate reward is exactly zero. This leads naturally to a reward function like the following:

$$r_h = \begin{cases} 1 & \text{if } r_{intermediate} > 0 \\ 0 & \text{if } r_{intermediate} = 0 \\ -1 & \text{if } r_{intermediate} < 0. \end{cases} \quad (1)$$

The optimal solution for such a problem maximizes the expected value of the *true reward* r_h. This optimal policy will generally be nonstationary: the optimal action from a given state depends on the number of timesteps remaining and the current score difference. If we have a *dynamic opponent* (which changes

strategy based on the score and/or time remaining), the optimal policy also needs to take the opponent's dynamics into account. We call such a policy DO-optimal (dynamic-opponent-optimal). An SO-optimal (static-opponent-optimal) policy is a policy which performs optimally against a *static opponent*: an opponent whose strategy does not depend on the score or time remaining.

Although the optimal policy depends on score and time remaining, the underlying dynamics of the domain do not. Adding score and time to the problem significantly increases computational complexity, which is problematic in challenging domains such as robot soccer. In practice, researchers in such domains often approximate the domain by omitting time and score from the model and attempting to maximize expected intermediate reward in this simplified model. This approximation sacrifices optimality for a significant reduction in computational complexity. Since the resulting policy ignores time and score, we call this approximation the stationary, score-independent (SSI) solution. In previous work, I have tested the SO-optimal and SSI solutions for a simple domain inspired by robot soccer. These results are discussed in Section 3.2.

In between the extremes of the SO-optimal and SSI solutions, there are many interesting approximations which trade off computation time with solution quality. In Section 3.2.7, I discuss heuristic solution techniques that aggregate states based on the time remaining. For this thesis, I intend to investigate additional approximate solutions and to quantify the tradeoffs involved in the application of these approximation techniques.

One objection that can be raised here is: why not simply use the true reward to guide the agent's behavior? Intermediate rewards are needed because I am interested in learning how to achieve the desired reward *online*—we want to win against an unknown opponent *during the course of a single game*. A reward signal that is only provided at the end of the game is not useful for online learning. By considering the score and time remaining, our agent can utilize the intermediate rewards, seen during play, to determine how to maximize the probability of winning.

1.4 Expected Contributions

The ultimate goal of my thesis research is to empirically demonstrate that explicit reasoning about score, time remaining, and opponent dynamics can lead to an improvement in team performance in a challenging, dynamic, adversarial domain, such as robot soccer. For my thesis, I propose to continue researching efficient solution techniques for timed, zero-sum games and to apply these techniques to challenging adversarial domains such as robot soccer. I also intend to address the problem of dynamic opponents. Specifically, I intend to contribute:

- Application of efficient MDP solution techniques to timed, zero-sum MDPs, including approximation algorithms.
- Heuristic approximation techniques for solving timed, zero-sum MDPs as a function of state space, time horizon length, and score.
- Investigation of the effect of time horizon length and time discretization on zero-sum performance.
- Algorithms for online reasoning about opponent behavior in a finite-horizon environment.
- Application of all these techniques to a complex adversarial domain such as robot soccer.

For more details on my proposed work, see Section 4.

2 Background and Related Work

There is a large body of work on reasoning optimally (and near-optimally) in stochastic and adversarial environments. For the work proposed in my thesis, I intend to make extensive use of Markov decision processes (MDPs), including factored MDPs and other efficient solution techniques. I discuss these techniques in this section. I also give brief summaries of zero-sum games that are often played by humans, and discuss which of these I expect my thesis work will be applicable to. Since this thesis was originally motivated by the domain of robot soccer, I also present a description of the robot soccer domain and present some related work from that field.

2.1 Markov Decision Processes

Markov Decision Processes (MDPs) are a powerful tool for planning in the presence of uncertainty [Puterman, 1994]. MDPs provide a theoretically sound means of achieving optimal rewards in uncertain domains. Since I am interested in domains with stochastic actions, I expect to make significant use of MDPs in my thesis work.

2.1.1 Definitions

An MDP is represented as a tuple (S, A, T, R, s_0). S is a set of states and A is a set of actions. s_0 is the initial state. T is the transition function; $T(s, a, s') \rightarrow [0, 1]$ denotes the probability of transitioning to state s' when action a is executed in state s. R is the reward function; $R(s, a)$ denotes the reward received upon taking action a when in state s. Alternatively, rewards can be found on the states, in which case the agent receives reward $R(s)$ upon entering state s. A *policy* is a mapping $\pi : S \times \mathcal{N} \rightarrow A$ from $\langle$state, time$\rangle$ pairs to actions. A *stationary policy* is a policy which does not depend on time; a policy that does depend on time is called *nonstationary* or *time-dependent* [Mundhenk et al., 2000].

2.1.2 Optimality Criteria

The typical goal of decision making in an MDP is to find an optimal policy. In order to do this, an *optimality criterion* (also known as an *objective function*) needs to be defined. Many optimality criteria for MDPs have been proposed in the literature (see [Puterman, 1994, Mahadevan, 1996] for extensive discussion); we concentrate here on those that are most commonly used:

- Expected cumulative reward
- Expected cumulative discounted reward
- Average reward rate (expected reward per time step).

In the *expected cumulative reward* and *expected cumulative discounted reward* frameworks, the *value* of state s under policy π is denoted as:

$$V_\pi^k(s) = E\Big[\sum_{t=0}^{k} \gamma^t r_t\Big], \tag{2}$$

where r_t is the reward received at time t, $\gamma \in (0, 1]$ is a discount factor applied to future rewards, and k is the *horizon*: the number of time steps until completion. When $k = \infty$, we have an *infinite-horizon problem*; when k is finite, we have a *finite-horizon problem*. The discount factor γ is generally used for the formulation of infinite-horizon problems; otherwise, the cumulative reward is likely to grow without bound. For finite-horizon problems, infinite rewards are not possible, so $\gamma = 1$ is typically used.

A common goal of decision making in an MDP is to find a policy π that maximizes $V_\pi^k(s)$ for every $s \in S$. Such a policy is said to be *optimal*. The optimal value $V^k(s)$ of s is the expected discounted future reward received when we start in state s and follow an optimal policy. V^k can be computed exactly using dynamic programming techniques; this process is known as *value iteration* [Bellman, 1957]. Value iteration uses the Bellman equation, which is:

$$V^{k+1}(s) = \max_{a \in A} \Big\{ R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^k(s') \Big\}, \tag{3}$$

where $V^0(s) = 0$. For the finite-horizon case with k timesteps remaining, V^k allows us to easily find the optimal next action from any state. However, the optimal action from a given state may depend on the number of time steps remaining; such a policy is said to be *nonstationary*. For the infinite-horizon case, V^k converges to some V^* as $k \to \infty$ (for $\gamma < 1$). A related method of computing the optimal policy is *policy iteration* [Howard, 1960], in which an initial policy π is iteratively improved using the value function V_π until it converges to the optimal policy π^* (with value V^*).

Another well-studied MDP optimality criterion is *average reward rate*, introduced by Howard [Howard, 1960]. Average reward rate is an undiscounted optimality framework which can be applied to infinite-horizon problems with a cyclic nature. Classical techniques such as value iteration and policy iteration can also be applied to average reward MDPs.

2.1.3 Efficient MDP Solutions

Solving MDPs directly can be computationally intractable when the state space is very large. Therefore, researchers have introduced a variety of techniques that allow for efficient solutions to MDPs representing large, real-world problems. These techniques will be valuable for my thesis work because the domains I am interested in investigating have complex, rich dynamics with many states and transitions. We will see in Section 3.2 that the SO-optimal solution to a timed, zero-sum game involves the creation of an MDP that is polynomially larger than the SSI approximation. Therefore, to find good policies in real-world domains, I will need to make use of efficient MDP solution techniques. In this section, I focus on the techniques of factoring, abstraction, and approximation.

A common technique for efficient MDP solution is *state abstraction* (also known as *state aggregation*), in which the original *ground* MDP is transformed into an *abstract* representation. Abstraction allows the decision maker to ignore irrelevant information, reducing the size of the state space and allowing for a more tractable solution. Li et al. provide a recent discussion of abstraction techniques, unifying several previously-proposed techniques into a single theoretical framework [Li et al., 2006]. In bisimulation [Givan et al., 2003], states with identical transition and reward functions are aggregated. This notion of state equivalence guarantees that the optimal policy for the abstract MDP corresponds to the optimal policy for the ground MDP. Dean et al. propose approximate bisimulation, in which states that behave approximately the same are grouped together, leading to a "bounded parameter MDP" (BMDP) [Dean et al., 1997]. The authors give an *interval value iteration* algorithm that is guaranteed to result in a policy that is within some bound of the optimal policy. A similar technique is proposed by Ferns et al., who have developed bisimulation metrics (distance functions) on the states of an MDP [Ferns et al., 2004]. Intuitively, states which are "close together" can be aggregated together. Other criteria that have been used for abstraction include equivalence of rewards and Q-values [Chapman and Kaelbling, 1991], equivalence of best actions [Jong and Stone, 2005], similar Bellman residuals [Bertsekas and Castanon, 1989], and many others.

In many real-world domains, it is advantageous to describe sets of states rather than enumerating them explicitly. In particular, the state of a system can often be fully specified by a cross-product of individual *features*. *Factored MDPs* (also known as *structured MDPs*) are MDPs in which the dynamics are represented symbolically [Boutilier et al., 1999]. This can be accomplished with probabilis-

tic STRIPS operators [Hanks and McDermott, 1994], dynamic Bayesian networks [Boutilier et al., 1995, Dean and Kanazawa, 1989], or algebraic decision diagrams (ADDs) [Hoey et al., 1999]. SPUDD is an algorithm that uses ADDs to represent value functions and policies [Hoey et al., 1999]. This allows SPUDD to find optimal policies in factored MDPs without explicitly enumerating the state space. Hoey et al. show that SPUDD can be used to efficiently compute optimal policies for factored MDPs containing tens of millions of states. SPUDD outperforms flat value iteration by multiple orders of magnitude for complex real-world planning problems.

Unfortunately, factored MDPs are not always immediately amenable to efficient solutions, as there is no guarantee that the value function can be represented compactly [Koller and Parr, 1999]. Therefore, approximation techniques have been explored for factored MDPs. An approximate version of SPUDD called APRICODD computes near-optimal value functions and policies, successfully finding near-optimal policies for MDPs with up to 34 billion states [St-Aubin et al., 2000, Hoey et al., 2000]. Guestrin et al. present two approximate solution algorithms that approximate the value function using a linear combination of basis functions [Guestrin et al., 2003]. Their algorithm compares favorably to APRICODD in performance, successfully finding approximately-optimal policies for MDPs with over 10^{40} states.

Sutton et al. discuss policy gradient techniques, which attempt to find a good policy by computing a gradient of the future reward and following the gradient to improve the policy [Sutton et al., 1999]. Kakade and Langford present a *conservative policy iteration* (CPI) algorithm [Kakade and Langford, 2002]. In traditional policy iteration, we replace the old policy π with a new policy π' after every policy improvement. In CPI, the new policy π_{new} is stochastic, where π and π' are weighted by a parameter α: $\pi_{new}(a, s) = (1 - \alpha)\pi(a, s) + \alpha\pi'(a, s)$. If $\alpha = 1$, the CPI algorithm is equivalent to traditional policy iteration. Under some assumptions, CPI is guaranteed to terminate is a small number of timesteps and returns an approximately optimal policy. With similar assumptions, Bagnell et al. present a dynamic programming algorithm (PSDP) in which policies are "backed up" instead of values [Bagnell et al., 2003].

2.1.4 Non-Markovian Rewards

Bacchus et al. discuss *non-Markovian rewards* as a way to assign rewards to behaviors that extend over time [Bacchus et al., 1996]. They define non-Markovian reward decision processes (NMRDPs) as a generalization of MDPs in which the reward function R takes in *histories*, of the form $\langle s_1, s_2, \ldots s_n \rangle$. Since the explicit specification of such a reward function is impossible (there are an infinite number of possible histories), the authors use a temporal logic (PLTL) to specify non-Markovian rewards. Since the value of an action depends on history, policies in an NMRDP are a mapping from histories to actions. The authors present an algorithm for converting an NMRDP into a standard MDP by "expanding" the MDP: annotating each state with the additional history information needed to ascribe the rewards. There is a tradeoff between the effort spent translating the MDP—producing a *small* equivalent MDP without storage of irrelevant history—and the effort required to solve the result (since an MDP with many unneeded states will take longer to solve using standard MDP solution methods). Some solution approaches specifically target factored MDP representations [Bacchus et al., 1997] or anytime heuristic search algorithms [Thiebaux et al., 2002].

The SO-optimal solution to a timed, zero-sum game is similar to the NMRDP solution algorithm, because the SO-optimal solution also requires that states be annotated with additional information about the time remaining and cumulative intermediate reward. However, the size of an SO-optimal MDP is only polynomially larger than the simplified stationary, score-independent MDP; with NMRDPs the transformed MDP can be exponentially larger. Section 3.2 discusses the worst-case computational complexity of SO-optimal MDPs in detail. It is likely that efficient approximation algorithms explored by the NMRDP community may also be viable appromixation algorithms for solving timed, zero-sum games; it is also likely that the approximation algorithms I explore in this thesis could also be used to find efficient approximate solutions for NMRDPs.

2.2 Domains

In this section, I summarize a variety zero-sum games that are often played by humans. I compare some of the important features of these domains, and consider which domains I expect my thesis work will be applicable to.

In the introduction, I classified several common AI problems according to the number of agents involved, whether the environment is known, whether the actions are stochastic or deterministic, and whether the observations are full or partial. In this section, I only consider zero-sum games, so we know that the number of agents is always two.[1] Since the rules of the game are known *a priori*, I assume that a model of the environment is known in all these cases, though for the more complicated games, the model is likely to be imperfect. These games will differ in action determinism, observability, and two other criteria: *timed ending* and *simultaneous play*. I consider a game *timed* if it ends after a certain amount of time has elapsed. Timed games are a subset of finite-horizon games in which the horizon is known exactly when play begins. In games with simultaneous play, players choose actions at the same time (or in a continuous fashion), instead of taking turns making moves.

The first game I consider is rock-paper-scissors (RPS). RPS is a matrix game in which each player simultaneously chooses an action; the joint action chosen deterministically specifies the rewards of each player. Unlike the other games we consider, there isn't really a concept of *states* or *observations* in rock-paper-scissors, since the game is just a one-shot choice of actions. Therefore, RPS has deterministic actions, no observations, and is not timed, but does have simultaneous play. All matrix games share these features.

In chess and go, players take turns making moves on a board. Actions are deterministic and the state is always fully observable to both players. Plays are not made simultaneously. In tournament play, these games often feature a per-player time limit, but that is not a main feature of the game: the time limit is simply in place to ensure that players do not take inordinately long to make moves. There is also no notion of *being ahead* when time runs out—any player whose time runs out is considered to have lost. Therefore, I do not consider these games to be timed.

There are many different variants of poker, but they mostly share the same high-level features. Each player receives some cards at random, and players take turns betting on their hands. In addition to betting, players may also have actions that allow an exchange of some cards for new cards. The results of betting are deterministic, but the cards received in an exchange are not. Therefore, actions may be deterministic or stochastic, depending on the specific variant of poker that is being played. What makes poker interesting is that the state of the game is not fully observable. Like chess and go, poker is not timed.

Soccer and American football are sports games in which a team of players attempts to manuever a ball into a goal. Each features stochastic actions (as even professional-level players cannot consistently reproduce the exact same results) and partial observability (because no team member can perfectly see the entire field of play at once). Play is simultaneous and timed, since the game ends when a specific time limit is reached. American football has a richer notion of *time management*: the game is split up into *plays*, and the results of certain plays stop the time clock until the next play has begun. The play of the final minutes of an American football game is typically heavily influenced by time management decisions: the team that is ahead will attempt to take as much time as possible between plays, while the team that is behind will attempt to execute as many plays as possible in order to score before time runs out.

Next, I turn to discussion of various computer and video games. There are many computer games that are interesting from an AI standpoint. New games are constantly being released by different developers, so the most popular games at any point in time are likely to change rapidly as the state of the art progresses. Therefore we concern ourselves with the characteristics of specific *genres* of computer games, without

[1] For some of these games, there are actually many more than two players involved; for instance, in soccer there are typically 22 players present on the field. However, I assume that all the players on a single team are fully cooperative. I therefore treat each team as a single agent playing against its opponent.

Game	Timed	Simultaneous Play	Actions	Observations
Rock-paper-scissors	no	yes	deterministic	N/A
Chess/Go	no	no	deterministic	full
Poker	no	no	determ./stocha.	partial
Soccer/American football	yes	yes	stochastic	partial
Turn-based strategy	maybe	no	stochastic	full/partial
Real-time strategy	maybe	yes	stochastic	full/partial
Tactical shooter	maybe	yes	stochastic	partial

Figure 2: Summary of common zero-sum games.

focusing too much on the details of any specific game.

Turn-based strategy games are conceptually similar to board games such as chess and backgammon, though the mechanics of gameplay are often orders of magnitude more complex. One of the most well-known turn-based strategy series is Sid Meier's *Civilization* series, in which players build rival civilizations. A popular open-source turn-based strategy game is *The Battle for Wesnoth*, in which players create armies and send them into battle. Turn-based strategy games are often timed. In *The Battle for Wesnoth*, each battle lasts for a fixed number of turns; if the player does not complete the battle objective before the turns run out, the result is a loss. Play is not simultaneous, but actions are stochastic. Observations may be full or partial, depending on the game.

Real-time strategy games are similar to turn-based strategy games, with the difference that players make moves simultaneously. Popular real-time strategy games include Blizzard's *Warcraft* and *Starcraft* series and Microsoft's *Age of Empires* series. Buro's ORTS project [Buro, 2006] is an open-source real-time strategy engine that is specifically intended for real-time AI research. Some real-time strategy games are timed, while some are not.

A first-person tactical shooter is a game in which two teams compete to achieve some goal. Each player controls a single in-world avatar, who is equipped with a gun and can shoot at the other players. Popular tactical shooters include Valve's *Counterstrike* and Epic's *Unreal Tournament*. In tactical shooters, play is simultaneous, actions are stochastic, and observations are partial (each player can only see what their avatar sees). Tactical shooters often have timed scenarios; examples include escorting a specific player to a certain location before time runs out and defusing a bomb before time runs out.

Figure 2 summarizes the characteristics of these games. All the domains which are considered *timed* are likely to directly benefit from the work proposed for this thesis. We can see that all of these timed domains have stochastic actions and often are only partially observable. Timed domains usually also feature simultaneous play. I would like to apply the work presented in this paper to at least one of these timed domains. In addition to the computer games, there are at least two robot soccer-based simulation environments that could be used: the RoboCup simulation league simulator and our own simulator for the four-legged league: PuppySim. PuppySim allows us to run our actual robots' behavior code (written in the Python programming language) in an off-board simulation environment.

2.3 Robot Soccer

In the RoboCup four-legged league [Kitano et al., 1997], two teams of four Sony AIBO robots play each other in a twenty-minute game of soccer on a field approximately 6m$\times$4m in size. Figure 3 shows a snapshot of a recent AIBO game. The settings and the rules of the game change every year to create new research challenges. The current complete rules of the domain are available online [RoboCup Technical Committee, 2007]. I focus here on the general features of the game that are of relevance to high-level strategy and team coordination. These features include:

- **Full autonomy:** each team of robots operates completely without human supervision. However, teams are allowed to change the robots' programming at halftime or during a timeout. Each team is granted one timeout per game. Timeouts may only be called during a global stoppage of play, such as after a goal is scored.
- **Distributed teams:** all perception, computation, and action are done on-board the robots. The robots are equipped with 802.11b wireless networking, which enables communication among team members; however, the robots are not allowed to communicate with any off-board computers.
- **Limited perception:** each robot's primary sensor is a low-resolution camera with a very narrow field of view (under 60 degrees). A single robot therefore has a very limited view of the world, so teams can benefit greatly from communication strategies that build a shared world model.
- **Dynamic, adversarial environment:** the presence of adversaries in the environment is a significant challenge. Opponents ensure that the environment is extremely dynamic: within a few seconds, the state of the world may change significantly. A team that takes too long to coordinate will have robots that display hesitation in carrying out their tasks, which gives the opponents a significant advantage.
- **Finite-horizon, zero-sum game:** Most importantly for this thesis, soccer is a finite-horizon, zero-sum game. A game of soccer has a winning team, a losing team, and a defined ending point. Playing a conservative strategy—which might work well over a long period of time—is of no use to a team that is losing and only has a few seconds remaining in the game. A team in this situation must choose a strategy that can score a goal quickly, even if such a strategy has other weaknesses. The four-legged league therefore presents teams with the difficult task of trying to learn how to win against an unknown opponent in a zero-sum game with a fixed time limit.

Figure 3: A RoboCup four-legged league soccer match.

In the RoboCup four-legged league, teamwork is typically accomplished by allocating *roles* to robots. Gerkey and Mataric [Gerkey and Mataric, 2004] discuss role assignment in RoboCup, giving an overview of the strategies used by teams in each of the RoboCup leagues. In the four-legged league, the predominant approach involves allocating three roles: an *attacker*, a *defender*, and some sort of *supporter*. According to Gerkey's survey, nearly all RoboCup teams assign roles to robots in a greedy fashion. For example, in

previous years, our own team (CMPack) assigned the roles in a fixed order using a well-defined objective function, namely first the *attacker* role to the robot that could reach the ball most quickly, then the *defender* role to whichever of the other two robots was closest to our own goal, and finally the *supporter* role to the remaining robot [Vail and Veloso, 2003]. CMPack's explicit use of teamwork and coordination was a very effective strategy that contributed strongly to our world championship in 2002. Other details about our robot soccer team can be found in our annual team report [Veloso et al., 2005] and Section 3.1 of this document.

Bowling et al. introduced the concept of *plays* as a method for team coordination in the RoboCup small-size league [Bowling et al., 2004a]. A play is a team plan that provides a set of roles, which are assigned to the robots upon initiation of the play. Each play also has a set of *applicability conditions*, which determine when a play is available for execution; a *weight*, which is used to decide which play to run when multiple plays are applicable; and *done conditions*, which specify when the play is complete and whether the play is considered to have completed successfully. Multiple plays can capture different teamwork strategies, as explicit responses to different types of opponents. Bowling et al. also show that play selection weights could be adjusted online to match an opponent—a technique known as *playbook adaptation* [Bowling et al., 2004b].

Dylla et al. propose a soccer strategy language that formalizes the strategies and tactics used by human soccer teams [Dylla et al., 2005]. Their goal is to be able to specify soccer strategies in an abstract way that does not depend strongly on the specific robot hardware used in the competition. However, the authors have not used this language in the implementation of any real robot soccer team. In Section 3.1, I present a strategy language that could be used in any multi-robot system and that has been implemented in the four-legged league of the 2005 RoboCup competition.

3 Previous Work

The previous work presented in this proposal has been strongly inspired by the domain of robot soccer, specifically the RoboCup four-legged league. In Section 3.1, I discuss my experience developing team strategies for robot soccer, introducing the idea of *plays* (alternative teamwork strategies) for distributed, multi-robot coordination. The desire for optimal play selection—choosing plays such that the probability of winning is maximized—is the motivation for this thesis. In Section 3.2, I discuss solution techniques for timed, zero-sum games in detail. I present an efficient algorithm for finding the SO-optimal policy, which is the optimal policy against a static opponent. I also present a variety of approximate solution methods that trade off solution quality with computation time. These algorithms and heuristics have been empirically validated in simple domains.

3.1 Distributed, Play-Based Role Assignment for Robot Teams

For two years, I have worked on multi-robot coordination as part of the CMPack'04 and CMDash'05 entries to the RoboCup four-legged league [Veloso et al., 2005]. For the 2005 international competition, I developed several teamwork innovations for the four-legged league. The most notable of these innovations is the introduction of *plays*—alternative teamwork strategies—in the context of a distributed team of robots. Plays allow a team of robots to discretely change their strategy depending on the state of the game. The inspiration for this thesis is the desire for *optimal play selection* in a robot soccer team, where the optimal play choice (at any point in time) is that which is most likely to lead to a victory.

In [McMillen et al., 2005], we empirically show the need for switching between two different ways of handling a simple robot soccer scenario, depending on the behavior of an opponent robot. We argue that the use of high-level coordination strategies is most appropriate for the RoboCup four-legged league, due to the distributed, dynamic, adversarial environment and the presence of high network latency.

Encouraged by this initial result, we decided to pursue a *play-based* coordination strategy for the 2005 competition [McMillen and Veloso, 2006b]. A *play* is a team plan that provides a set of roles, which are assigned to the robots upon initiation of the play. Bowling et al. introduced a play-based method for team coordination in the RoboCup small-size league [Bowling et al., 2004a]. However, the small-size league has centralized control of the robots. One of the significant contributions of our work is the development of a play system that works in a distributed team. Our play system is the first known implementation of play-based teamwork in a distributed team of robots.

The play language described by Bowling assumes that the number of robots is fixed, and therefore always provides exactly four different roles for the robots. In another extension to Bowling's work, our plays also specify which roles are to be used if the team loses some number of robots due to penalties or crashes. This extension to Bowling's play language allows the team to robustly adapt to the loss of team members without the need for additional communication. This is an important extension for domains with limited or high-latency communication.

3.1.1 Play Language

Our play language is strongly inspired by the work of Bowling et al. Our language allows us to define *applicability conditions*, which denote when a play is suitable for execution; what *roles* should be assigned when we have a specific number of active robots on the team; and a *weight*, which is used to decide which play to run when multiple plays are applicable.

Applicability. An applicability condition denotes when a play is suitable for execution. Each applicability condition is a conjunction of binary predicates. A play may specify multiple applicability condi-

```
PLAY StrongDefense
APPLICABLE fewerPlayers
APPLICABLE secondHalf winningBy2OrMoreGoals
ROLES 1 Goalkeeper
ROLES 2 Goalkeeper Defender
ROLES 3 Goalkeeper Defender Independent
ROLES 4 Goalkeeper Defender Midfielder Independent
WEIGHT 3
```

Figure 4: An example play with multiple applicability conditions.

tions; in this case, the play is considered executable if any of the separate applicability conditions are satisfied.

Roles. Each play specifies which roles should be assigned to a team with a variable number of robots by defining different `ROLES` directives. A directive applies when a team has k active robots, and specifies the corresponding k roles to be assigned. If a robot team has n members, each play has a maximum of n `ROLES` directives. Since our AIBO teams are composed of four robots, our plays have four `ROLES` directives.

Weight. Weight is used to decide which play to run when multiple plays are applicable. In our current implementation, the play selector always chooses the applicable play with greatest weight. The manual selection of weights is used to accomplish desired behavior. Ideally, the team would instead make optimal play choices autonomously, by solving an appropriate decision problem.

The *play selector* is a software module that runs continuously on one robot that is arbitrarily chosen to be the leader. The play selector chooses which play the team should be running. The leader periodically broadcasts the current play (and role assignments) to its teammates. A play is considered to be complete as soon as the play selector chooses a different play, which may happen because the current play is no longer applicable or because another play with greater weight has recently become applicable.

Figure 4 shows an example of a defensive play. Its applicability conditions specify that this play is applicable 1) when our team has fewer active players than the opponents or 2) when the game is in the second half and our team is winning by at least two points. If we have only one active robot on our team, we will assign it the Goalkeeper role; if we have two robots, one is assigned the Goalkeeper role and the other is assigned the Defender role; and so on.

The *role* assigned to each robot determines what behaviors the robot actually runs. Our approach is unique in that it is *region-based*: each robot is assigned to a region of the field. A robot is primarily responsible for going after the ball whenever the ball is in that robot's region. When the ball is not in its region, the robot will position itself at a good position within its region. To ensure that one or more robots are always chasing after the ball, these regions typically overlap significantly. We have developed algorithms that prevent the robots from interfering with one another even when they are playing in overlapping regions.

3.1.2 Summary

We have developed a distributed play-based role assignment algorithm for a distributed team of soccer-playing robots. The algorithm aims to solve several important multi-robot challenges, including the presence of adversaries, resistance to play and role oscillation, and robustness to network failure. In addition to

plays, we made several other enhancements to teamwork and coordination for the 2005 competition. These enhancements are described in [McMillen and Veloso, 2006a] but not discussed in this document.

This play-based system was tested in the RoboCup 2005 competition. Our team came in fourth place in a challenging competition of twenty-four teams. Our team typically rotated through three well-balanced plays in the first minutes of each game, which allowed us to see the performance of each play against the specific opponent. As a form of adjustable autonomy, we could manually change the team's strategy at halftime or during a timeout. Plays also allowed us to have the team autonomously change strategy based on the time left in the game and the current score. For instance, if there were fewer than two minutes remaining and our team was losing, the team would autonomously decide to play very aggressively, "pulling" the goalkeeper out of the goal box and providing us with another field player that could potentially score a goal. In fact, in the 3rd–4th place game of the RoboCup 2005 competition, our goalkeeper robot nearly scored a goal in the final seconds of the game.

The major limitation of our current play system is that all the reasoning about score, time remaining, and play utility (weights) is manually hand-coded into the plays. Ideally, we would like the team to reason optimally, using some model of the environment and opponents, to select the plays that are most likely to lead to a win. The work proposed for my thesis aims to effectively address these limitations.

3.2 Timed, Zero-Sum MDPs

In this section, I consider timed, zero-sum MDPs in detail, including a detailed example which assumes that the opponent is static. I present the SO-optimal algorithm, which efficiently finds the optimal policy for the static-opponent case. However, for large state spaces and long time horizons, the SO-optimal algorithm is computationally intractable. I therefore present the stationary, score-independent (SSI) approximation, which ignores time remaining and score; I also present the stationary, score-dependent (SSD) approximation, which considers score but ignores time remaining. I consider three heuristics that allow for an approximate solution to timed, zero-sum MDPs with large state spaces or long time horizons. In preliminary results, I show the performance of the optimal solution and these approximations on a variety of simple MDPs.

3.2.1 Definitions

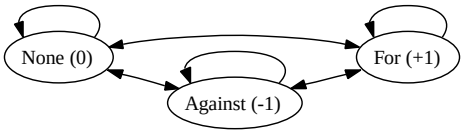
In a *timed, zero-sum MDP*, there is a fixed time horizon h; after h steps have occurred, the agent with the higher score wins. Let *intermediate reward* be the difference between our agent's score and the opponent's score. We can then consider "state" to be a cross-product of three entities: the base state of the system, s; the number of time steps remaining, t; and the current intermediate reward, ir.[2] Each final state (in which $t = 0$) is assigned reward according to Equation 4: $+1$ if $ir > 0$, -1 if $ir < 0$, 0 otherwise. The objective is to maximize expected reward, which is equivalent to maximizing $\Pr[win] - \Pr[loss]$.

3.2.2 Example

In a timed, zero-sum MDP, the optimal policy will generally be nonstationary. To illustrate this, we present an example—inspired by robot soccer—that will be used in the following sections. We simplify the robot soccer domain significantly by modeling it as an MDP M with three states, as shown in Figure 5:

1. FOR: our team scores a goal (cumulative intermediate reward $+1$)
2. AGAINST: the opponents score a goal (cumulative intermediate reward -1)

[2]The base state is the physical state of the system; for example, in robot soccer, the base state might include the position of the ball and the poses of the eight robots.



a	$T(*, a, \text{FOR})$	$T(*, a, \text{AGAINST})$	$T(*, a, \text{NONE})$
balanced	0.05	0.05	0.9
offensive	0.25	0.5	0.25
defensive	0.01	0.02	0.97

Figure 5: Example MDP M, inspired by robot soccer.

3. NONE: no score occurs

Our agent is a team of robots, and each action choice corresponds to a strategy (play) the team can adopt. In this example, we consider three different plays: *balanced*, *offensive*, and *defensive*. We treat the opponent team as static, using the transition probabilities to model the opponent as part of the environment. The transition probabilities of these actions are shown in Figure 5. When the *balanced* play is chosen, our agent has a 5% chance of scoring and the opponent has a 5% chance of scoring. The *offensive* play is risky: it increases our team's chance of scoring but gives the opponent an even greater chance of scoring. Inversely, the *defensive* play is conservative. For simplicity, these transition probabilities do not depend on the current state.

3.2.3 Optimal Solution

The MDP M shown in Figure 5 is a *base MDP*: M gives the transition dynamics of a domain, but it does not include any concept of cumulative score or time remaining. To solve the problem optimally, we need to include all possible combinations of base states, score, and time remaining. Figure 6 shows an MDP M' that is the proper representation of a timed, zero-sum MDP with base MDP M and time horizon $h = 3$.

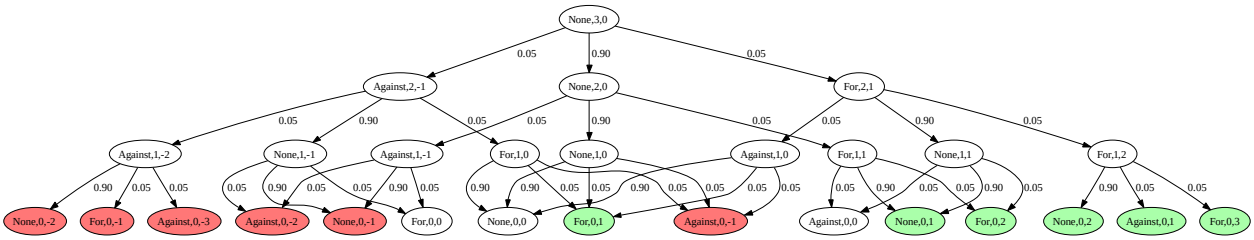


Figure 6: M', the timed, zero-sum MDP with base state dynamics given by M (presented in Figure 5) and time horizon $h = 3$. Lightly-shaded states have reward 1; darkly-shaded states have reward -1; unshaded states have reward 0. Transition probabilities for the *balanced* action are shown.

Each state s' in M' is a tuple (s, t, ir), where s is a base state from M, t is the number of time steps remaining, and ir is the cumulative intermediate reward received by the agent within the first $h - t$ time steps. The agent starts in state $(\text{NONE}, 3, 0)$, indicating that the agent is in base state NONE with 3 time steps remaining and no cumulative intermediate reward. At every time step, t decreases by 1; ir increases by 1 when our team scores a goal (when we enter the FOR state) and decreases by 1 when our opponent scores a goal (when we enter the AGAINST state). The final states are the only states with any reward; our agent receives reward $+1$ for a win and -1 for a loss.

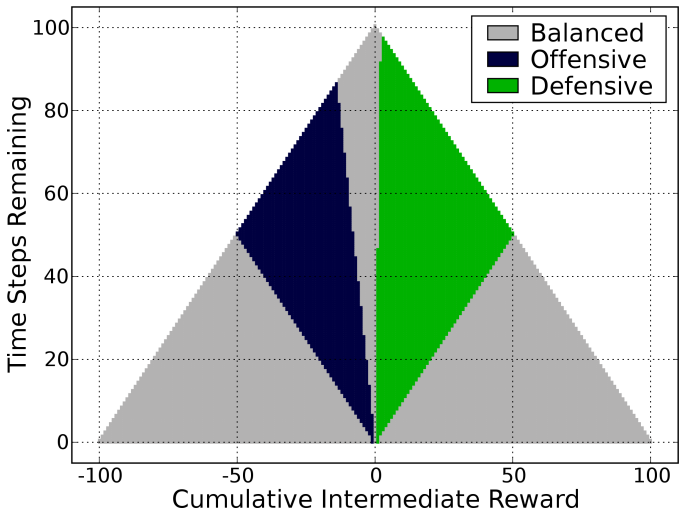


Figure 7: The optimal policy for M (shown in Figure 5), with time horizon $h = 100$ steps. The y-axis shows the number of time steps remaining; the x-axis shows the cumulative intermediate reward (score difference). The shaded areas show the optimal action for every possible combination of time remaining and intermediate reward.

The policy that maximizes rewards in this MDP is the SO-optimal (static-opponent-optimal) policy.[3] This policy can be found using any MDP solution technique. Since the states of M' are arranged into layers and all rewards are found in the bottom ($t = 0$) layer, value iteration can be performed efficiently: we start at the $t = 0$ layer and apply Bellman backups until the rewards "bubble up" to the top. Using this algorithm, each state in M' is backed up exactly once.

We now compute the running time of this algorithm. Let S be the set of states in the base MDP M and let IR be the set of all possible cumulative intermediate reward values. Since each state in M' is a tuple (s, t, ir), the number of states in M' is upper bounded by $|S| \times |IR| \times (h + 1)$. Assume that intermediate rewards are drawn from a set N of small integers (which is typically the case for timed, zero-sum games) and let m be the element of N with highest magnitude. Then the elements of IR are bounded by $[-mh, mh]$, so $|IR| = 2mh + 1$. Then the number of states in M' is upper bounded by $|S| \times (2mh + 1) \times (h + 1) = O(|S|h^2m)$. Each Bellman update requires a maximization over $|A|$ actions of a sum of $\leq |S|$ possible successor states. Since each state is updated exactly once, the worst-case running time of value iteration on M' is $O(|A||S|^2h^2m)$.

Figure 7 shows the SO-optimal policy for the domain M (Figure 5) with time horizon $h = 100$. The y-axis shows the number of time steps remaining; the x-axis shows the cumulative intermediate reward (score difference). The shaded areas show the optimal action for every possible combination of time remaining and intermediate reward. (Since M's transition probabilities are the same from every state, the policy does not depend on the current state.) This policy has an expected reward of 0.151. By following this policy, our agent will win approximately 50% of the time, lose 35% of the time, and tie 15% of the time.

Figure 7 also shows that the optimal policy for M is nonstationary: the policy depends on the number of time steps remaining and the cumulative intermediate reward. Qualitatively, the optimal policy is to choose the *defensive* play when winning by a significant number of points and to choose the *offensive* play when losing by a significant number of points. When the score is close to a tie, the best play is *balanced*. As the time remaining decreases, the point difference needed to choose *offensive* or *defensive* decreases—the

[3] Remember that we are assuming that the opponent team is static, which allows us to use the transition probabilities to model the opponent as part of the environment.

Algorithm 1 Creates a minimal MDP M' suitable for finding the SO-optimal policy given a base MDP M and time horizon h.

1: **Given:** MDP $M = (S, A, T, R, s_0)$, time horizon h
2: $s'_0 \leftarrow (s_0, h, 0)$
3: $S' \leftarrow \{s'_0\}$
4: **for** $i \leftarrow h$ to 1 **do**
5: **for all** states $s'_1 = (s_1, t, ir) \in S'$ such that $t = i$ **do**
6: **for all** transitions $T(s_1, a, s_2)$ in M **do**
7: $s'_2 \leftarrow (s_2, t - 1, ir + R(s_2))$
8: $S' \leftarrow S' \cup \{s'_2\}$
9: $T'(s'_1, a, s'_2) = T(s_1, a, s_2)$
10: **for all** states $s' = (s, t, ir)$ in M' **do**
11: **if** $t = 0$ and $ir > 0$ **then**
12: $R'(s') \leftarrow 1$
13: **else if** $t = 0$ and $ir < 0$ **then**
14: $R'(s') \leftarrow -1$
15: **else**
16: $R'(s') \leftarrow 0$
17: **return** $M' = (S', A, T', R', s'_0)$

agent acts more "urgently." The *balanced* regions in the lower-left and lower-right of the figure are states from which the actions of the agent no longer have any effect on the outcome (because the number of steps remaining is greater than the score difference). In these regions, all actions have equal expected reward and the agent chooses *balanced* by default.

3.2.4 Constant-Factor Speedup

The discussion in the preceding section did not address the issue of how to generate the SO-optimal MDP M'. The naïve approach is to simply generate all combinations of s, t, and ir. Then the number of states in the resulting MDP is exactly $|S| \times (2mh + 1) \times (h + 1)$. For M (presented in Figure 5) and $h = 100$, this results in an MDP with 61,206 states. However, many of these states are not reachable. In practice, we can gain a constant-factor speedup by only generating the reachable states. Algorithm 1 creates a minimal MDP M' that only contains the reachable states. For M and $h = 100$, Algorithm 1 generates an MDP with 30,000 states; this is approximately a factor-of-2 speedup.

We now explain Algorithm 1 in detail. The set of states S' is initialized to the initial state $s'_0 = (s_0, h, 0)$ (lines 2–3). The first loop (lines 4–9) generates all the remaining states in S' and the transition function T'. This loop iterates from h steps remaining down to 1; at iteration i it generates all states that have $i - 1$ time steps remaining, by finding all the states in S' with i time steps remaining (line 5) and generating all possible successors of these states. For each such state $s'_1 = (s_1, t, ir)$, the algorithm finds all transitions in M from s_1 to some other state s_2 on action a (line 6). A new state s'_2 is created for each such s_2. Each s'_2 has base state s_2, $i - 1$ time steps remaining, and cumulative intermediate reward $ir + R(s_2)$ (line 7). s'_2 is added to S' if it doesn't already exist (line 8). The transition probability $T'(s'_1, a, s'_2)$ is equal to the transition probability $T(s_1, a, s_2)$ of the base MDP M (line 9). The second loop (lines 10–16) assigns rewards to each state in S'. The algorithm has now defined S', T', R', and s'_0; the action space A is left unchanged. The converted MDP M' is then (S', A, T', R', s'_0) (line 17).

3.2.5 Stationary, Score-Independent Approximation

The SO-optimal solution is the optimal solution for a timed, zero-sum game (under the assumption that the opponent is static). However, the SO-optimal MDP is larger than the base MDP by a factor of approximately h^2m. For domains with large state spaces or long time horizons, the SO-optimal solution becomes computationally intractable. An approach often adopted in practice is to simplify the domain by omitting time and score from the model. This is equivalent to maximizing expected rewards in the base MDP M. The resulting policy is stationary and does not depend on score; we therefore call this policy the stationary, score-independent ("SSI") solution.

In MDP M, the expected one-step reward of action a from state s is equal to $\sum_{s'} R(s') \times T(s, a, s')$. Using this, we can compute the expected one-step reward of each action:

- *balanced*: $0 = 0.05 \times 1 + 0.05 \times -1 + 0.9 \times 0$
- *offensive*: $-0.25 = 0.25 \times 1 + 0.5 \times -1 + 0.25 \times 0$
- *defensive*: $-0.01 = 0.01 \times 1 + 0.02 \times -1 + 0.97 \times 0$

The optimal SSI policy executes the *balanced* action at every time step. The *balanced* play has the highest expected one-step reward, and (for this MDP) also has the optimal expected long-term reward for any choice of γ in the range $[0, 1]$.

This policy has an expected true reward of 0, with the probability of winning being equal to the probability of losing. The exact probability of each result depends on the time horizon h and can be determined by the multinomial probability distribution. For $h = 100$, the probability of winning is 43.63%, the probability of losing is 43.63%, and the probability of tying is 12.74%. Compared to the SO-optimal policy, the SSI policy is more likely to lose and less likely to win or tie.

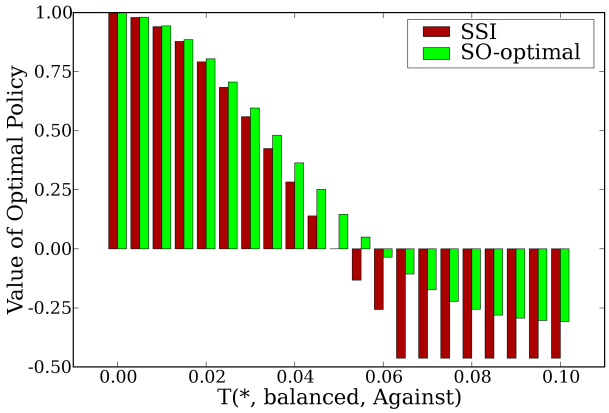


Figure 8: Effect of changing the opponent's capabilities. The x-axis shows the probability $T(*, \textit{balanced}, \textsc{Against})$ of our opponent scoring when we choose the *balanced* action. The y-axis shows the expected true reward of following the SO-optimal and SSI policies.

In Figure 8, we show the effect of changing the opponent's capabilities. Specifically, we vary the probability $T(*, \textit{balanced}, \textsc{Against})$ of our opponent scoring when we choose the *balanced* action. The y-axis shows the expected true reward of following the SO-optimal and SSI policies. In all cases, SO-optimal performs better than SSI. It is interesting to note that the difference between the two policies is greatest when the capabilities of each team are similar—that is, where $T(*, \textit{balanced}, \textsc{Against})$ is close to 0.5.

We also consider the performance of SO-optimal and SSI on 5000 randomly generated MDPs. Each of these MDPs has the same structure as M (shown in Figure 5), but the transition probabilities of each

state/action pair are chosen as follows: $T(s, a, \text{AGAINST})$ is chosen uniformly from $[0.0, 0.5)$; $T(s, a, \text{FOR})$ is chosen uniformly from $[0.9, 1.0) \times T(s, a, \text{AGAINST})$; and $T(s, a, \text{NONE})$ is set such that the three probabilities sum to 1. Note that our team is less likely to score than the opponents at every timestep, no matter which action is chosen. Therefore, the expected true reward of SSI is negative for every MDP. Figure 9 is a histogram depicting the distribution of true rewards for SO-optimal and SSI on these 5000 MDPs. Each bar shows the number of MDPs that have optimal policies within a given range of true rewards. The mean true reward for SO-optimal is 0.1971; the mean true reward for SSI is -0.0659. These results show that explicitly reasoning about score and time remaining allows our team to win with high probability, even against an opponent that is otherwise superior.

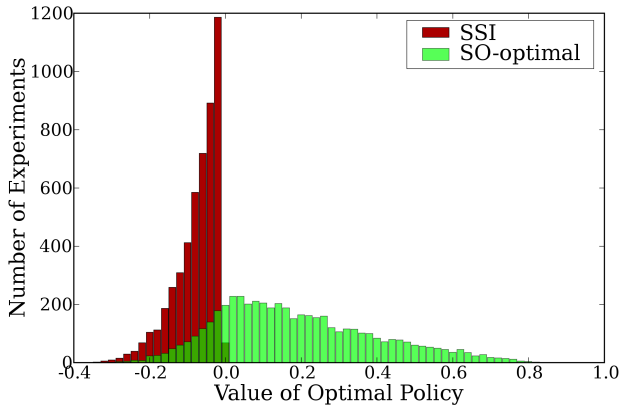


Figure 9: Performance of SO-optimal and SSI on 5000 randomly generated MDPs. Each bar shows the number of MDPs that have optimal policies within a given range of true rewards. The mean true reward for SO-optimal is 0.1971; the mean true reward for SSI is -0.0659.

3.2.6 Stationary, Score-Dependent Approximation

In this section, I briefly consider the class of stationary, score-dependent (SSD) policies, which depend on score but not on time. SSD policies are a richer class of policies than SSI; the optimal SSD policy for any domain has expected true reward at least at high as the optimal SSI policy for that domain. In preliminary results, I have found the optimal SSD policy for M. The optimal policy for this domain is as follows:

$$\pi(s, t, ir) = \begin{cases} \textit{defensive} & \text{if } ir \geq 1 \\ \textit{offensive} & \text{if } ir \leq -4 \\ \textit{balanced} & \text{otherwise.} \end{cases} \tag{4}$$

This policy has expected true reward of 0.0827. By following this policy, our agent will win approximately 48.0% of the time, lose 39.8% of the time, and tie 12.2% of the time. This is a significant performance improvement over SSI, even though this policy does not incorporate time.

3.2.7 Heuristic Approximation Techniques

In this section, I present three different heuristic techniques that take both score and time remaining into account and allow us to find high-quality approximate solutions. Each of these heuristics can be seen as an informed version of state aggregation in which states are aggregated based on the time remaining.

The *uniform-k* heuristic. With this heuristic, our agent adopts a nonstationary policy but only considers changing its policy every k time steps. The net effect of this change is to "compress" the time horizon

uniformly by a factor of k. This change directly leads to a decrease in the state space of the expanded MDP M', allowing for a more efficient solution. However, this solution will be suboptimal because the agent does not consider switching policies at every time step.

The *lazy-k* **heuristic.** With this heuristic, our agent ignores time and score until there are k steps remaining. For the first $h - k$ time steps, the agent acts in accordance with the optimal SSI policy for the base MDP M. Once there are k steps remaining, the agent uses Algorithm 1 to create an SO-optimal MDP M' with a time horizon k and an initial state chosen to reflect the actual current state of the system (including the cumulative intermediate reward). The agent solves M' and uses the SO-optimal solution to adopt a nonstationary policy for the remaining k time steps. The main idea of this technique is to concentrate computational effort near the end of the run, when the agent's actions may have a greater effect on the overall outcome.

The *logarithmic-k-m* **heuristic.** With this heuristic, the agent makes a number of decisions that is logarithmic in the time horizon. The *lazy-k* heuristic saves computation by only considering the end of the time horizon; the *uniform-k* heuristic saves computation by compressing the entire time horizon. The *logarithmic* heuristic is a hybrid approach in which the time resolution becomes finer as we approach the time horizon. For instance, we might allow the agent to switch policies at every step during the final 10 steps of the run, every two steps for the previous 20 steps of the run, every four steps for the previous 40 steps of the run, and so on. The *logarithmic* heuristic depends on two parameters: k, the number of decisions the agent makes before the time resolution is increased, and m, the multiple by which the resolution is increased. For the example given above, $k = 10$ because the agent needs to take 10 actions before each increase, and $m = 2$ because the time resolution doubles on each increase.

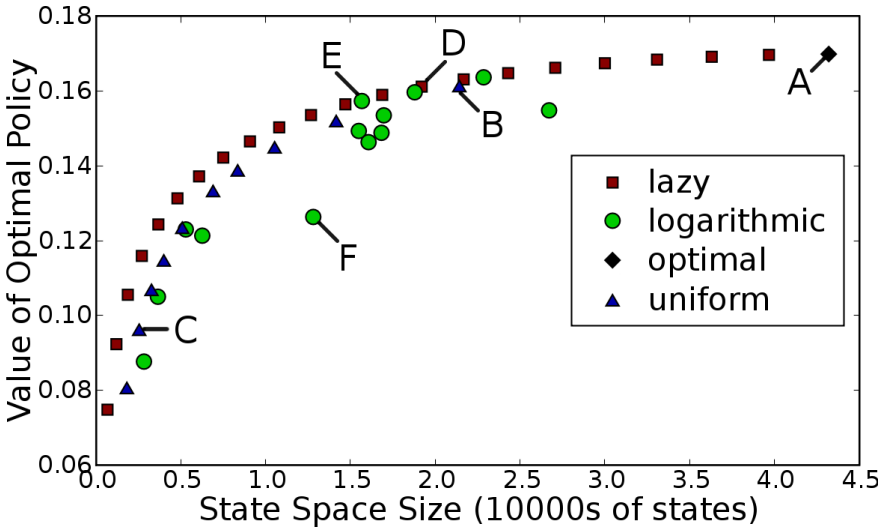


Figure 10: Performance of heuristic techniques on 60 randomly generated MDPs.

I tested the performance of these heuristic techniques, for a variety of parameter settings, on 60 different MDPs. These MDPs were chosen randomly from the 5000 MDPs that were used in Section 3.2.5.[4] Figure 10 summarizes the results. Each point on the graph corresponds to a heuristic technique with some parameter setting. The x-axis shows the number of states in the expanded MDP (averaged over the 60 MDPs); the y-axis shows the mean expected true reward of that heuristic technique. Ideally, we would like a technique that provides a high true reward with a low number of states. Points in the upper-left frontier of the graph represent Pareto-efficient tradeoffs between state space size and expected true reward.

[4]For the results presented in this section, h was set to 120 instead of 100; this was convenient for the *uniform-k* heuristic because many integers evenly divide 120.

The SO-optimal algorithm, labeled "A" in the graph, has a mean reward of 0.1699 and requires 43,200 states to compute. For small values of k, the *uniform* heuristic closely approximates the optimal solution while significantly reducing the size of the state space. *Uniform*-2, labeled "B", has mean reward 0.1608 and requires 21,420 states. The selection of k is a tradeoff between solution time and quality; *uniform*-15 (labeled "C") uses only 2,544 states, but the reward drops to 0.0957. *Lazy*-80 (labeled "D") has mean reward 0.1612 and uses only 19,200 states; this is fewer states than *uniform*-2 and a higher mean reward. In general, the *lazy* heuristic consistently has a higher reward than *uniform* at a given state space size. *Logarithmic*-8-2, labeled "E", closely matches the performance of *lazy*-k; however, *logarithmic*-2-4 (labeled "F") performs much worse than both *lazy* and *uniform*. In general, the performance of *logarithmic* seems highly parameter-dependent, and in no case does *logarithmic* significantly outperform *lazy* at a given state space size. Of the heuristics considered in this section, *lazy* consistently offers the best tradeoff in terms of solution time and quality, which indicates that acting optimally is most important near the end of the time horizon.

3.2.8 Summary

In this section, I discussed a variety of solution methods for timed, zero-sum games. The SO-optimal solution can be found by value iteration in $O(|A||S|^2h^2m)$ time. I have presented an algorithm that allows for a constant-factor speedup by generating the minimal MDP M' that properly represents a timed, zero-sum problem. I have shown experimentally that the SO-optimal solution can allow our team to win with high probability, even against a wide variety of otherwise superior opponents. Since the exact value iteration algorithm may be computationally intractable for MDPs with large state spaces or long time horizons, we are interested in approximations to the SO-optimal solution. In practice, the SSI solution is often used. The SSI solution ignores time and score; SSI offers a significant reduction in computational complexity but produces a poor-quality solution. The SSD solution only ignores time; SSD is a richer class of policies that generally outperforms SSI. I have also developed three heuristic techniques that can be used to find approximately-optimal policies for timed, zero-sum domains. The *lazy*-k heuristic ignores time and score until there are k steps remaining; in preliminary experimental results, *lazy*-k consistently has the highest performance at a given state space size. The *lazy*-k heuristic concentrates computational effort on the end of the time horizon, when the agent's actions have a greater effect on the final outcome.

The solution techniques presented above all assume that the opponent is static and can be treated as part of the environment. If the opponent is unknown, or changes its strategy during the course of the game, we cannot model the opponent as a static part of the environment. In the future, I plan on considering the problem of dynamic opponents. Further details of my proposed work can be found in Section 4.

4 Proposed Work

The ultimate goal of my thesis research is to empirically demonstrate that explicit reasoning about score, time remaining, and opponent capabilities can lead to an improvement in team performance in a challenging, dynamic, adversarial domain, such as robot soccer.

In my previous research, I have presented a principled, exact algorithm for finding optimal policies in timed, zero-sum MDPs (see Section 3.2). However, there are two unsolved problems that need to be addressed in order to achieve positive results in a challenging domain:

1. **Scaling.** The exact value iteration algorithm for timed, zero-sum MDPs allows us to find SO-optimal solutions in $O(|A||S|^2h^2m)$ time. Because of the quadratic dependence on time horizon length and state space size, the exact algorithm is impractical for domains with long time horizons or large state spaces.

2. **Presence of opponents.** By modeling the environment as an MDP, we assume that the opponent is static and can be treated as part of the environment. If the opponent is unknown, or changes its strategy during the course of the game, we cannot model the opponent as a static part of the environment.

The main contribution of my thesis will be to address these two challenges. I intend to investigate solution methods that scale well to challenging domains, and to investigate approaches that take the abilities of unknown, dynamic opponents into account. The specific techniques that I intend to use are described in the following sections.

For the final thesis, I plan to present results for a robot soccer simulation domain (PuppySim or the RoboCup simulation server) or one of the timed computer-game domains discussed in Section 2.2. I would also like to implement these algorithms for a real team of soccer-playing robots (in the RoboCup four-legged league or small-size league). I intend for the real-robot implementation to be a proof of concept: due to the significant difficulty in running large numbers of real-robot experiments with many robots, I am unsure whether I will be able to achieve statistically significant results on the real robots.

4.1 Expected Contributions

4.1.1 Efficient Solution Techniques

The SO-optimal solution does not scale well with respect to the time horizon length or state space size. However, the structure of the SO-optimal MDP is very regular, which suggests that factored solutions will work well for these problems. I therefore intend to apply some of the factored techniques, such as SPUDD [Hoey et al., 1999], to solving timed, zero-sum problems.

There are a variety of approximately-optimal MDP solution techniques, such as [St-Aubin et al., 2000, Guestrin et al., 2003], that could be applied to timed, zero-sum domains. These techniques should scale even better than the factored approaches at the cost of sacrificing some amount of optimality.

I also intend to explore policy-iteration solution techniques. The application of classical policy iteration [Howard, 1960] to zero-sum MDPs is trivial but will suffer from the same scaling problems as the value iteration algorithm. However, there are several efficient approximation algorithms that use a policy iteration approach, such as policy gradient methods [Sutton et al., 1999] and PSDP [Bagnell et al., 2003].

4.1.2 Heuristic Solution Techniques

In addition to these general techniques, there are a number of approximation techniques that are specific to timed, zero-sum problems. In Section 3.2.7, I introduced the *uniform-k*, *lazy-k*, and *logarithmic-k-m* heuristics. Each of these heuristics can be seen as an informed version of state aggregation in which

states are aggregated based on the time remaining. I am interested in investigating additional heuristics that aggregate states based on time.

I am also interested in pursuing approaches that aggregate states based on intermediate rewards. Each layer of the MDP created by Algorithm 1 grows wider by a constant factor m because we assume that, in the worst case, the highest-magnitude score m occurs on every time step. In real-world domains, many time steps occur between each score, so many of the states are extremely improbable. Aggregating states with high-magnitude cumulative intermediate rewards into a single "win" or "lose" state may also allow for near-optimal solutions at a reduced computational cost.

I also believe that these heuristics need to be tested on a wider variety of more complex domains. The 3-state MDP presented in Section 3.2 is quite simple; it is not yet clear how these heuristics will perform in more complex domains.

4.1.3 Reasoning about Opponent Behavior

The most challenging aspect of my thesis is the problem of online learning in an adversarial, finite-horizon environment. I would like to show that a team of agents can improve its performance against an unknown opponent during the course of a single game. This problem presents several challenges, including:

1. **Large state space.** The features of interest in a robot soccer game include the position (x, y) of the ball and poses (x, y, θ) of the eight robots. This means that the state space consists of at least 26 continuous dimensions. In addition to these dimensions, we also care about the score, time remaining, penalized robots, etc.

2. **Limited number of training examples.** A full game of robot soccer is 20 minutes long. If we assume that the team selects a play once every second, this corresponds to only 1200 learning examples over the course of a game. Even if the state space is reduced significantly by discretization and domain engineering, many legal states will not be visited over the course of a single game.

3. **Sparse rewards.** The only true reward in timed, zero-sum domains comes at the end of the game; in robot soccer even the intermediate rewards are sparse (the average game has about 6 total goals scored).

4. **Highly stochastic actions.** In robot soccer, an action is a play selection. The outcome of each action is stochastic because of several factors:

   - The robots' actuators are very noisy: kicks fail, robots slip as they move on the carpet, etc.
   - The presence of adversaries in the environment: opponent robots manipulate the ball, push our robots, obstruct kicks, etc.
   - Plays control the robots only at a high level; at a lower level the robots are autonomously deciding where to look, where to walk, when to kick, etc.

These challenges mean that the application of typical online reinforcement learning algorithms will not be feasible. Instead, I propose a hybrid offline/online approach, in which the team learns extensively against several different opponents offline, then uses the learned knowledge online in order to adapt to an unknown opponent. I will assume that the opponent's action granularity is the same as ours: that is, they choose from a set of ($\sim$5-20) high-level strategies (plays) at every time step.

In the offline phase of learning, we would run each of our team's plays against each of the opponent's plays. Each play combination will be tested for a large number of time steps (probably in the millions). The offline learning component does not directly reason about score and time remaining, because the final

outcome of each game does not matter (since we are practicing offline against a fictional opponent.) Rather, we are interested in collecting statistics that will allow us to make proper decisions against a real opponent in the online scenario. The offline learning algorithm will collect the following statistics:

- The distribution of scores and time-to-score for every play combination. Specifically, let $D^+(s, p, q, t)$ be the probability of our team scoring in exactly t timesteps, given that the game is currently in state s, our team chooses play p, and our opponent chooses play q throughout the entire time interval. Similarly, let $D^-(s, p, q, t)$ be the probability of our opponent scoring in exactly t timesteps, given that the game is currently in state s, our team chooses play p, and our opponent chooses play q. Through offline play, we can learn estimates $\hat{D}^+$ and $\hat{D}^-$ of D^+ and D^-. The online component can then use these estimates to make proper decisions during game play.

- Estimates $\hat{T}$ of the domain's transition probabilities. The transition probabilities depend on the play selections of both teams, so I am interested in estimating $T(s, p, q, s')$, which is the probability of transitioning from s to s' when our team chooses play p and the opponent team chooses play q. These estimates will be used in the online learning component to determine which play the opponent team is selecting.

- The play p that performs best against a wide variety of the opponent's plays (on average). This play will be used as the default play during the beginning of the online component, when we don't yet have any knowledge of what play the other team is selecting.

Note that the description above does not specify how the state will be represented as input to the functions D^+, D^-, and T. I expect that the state input to the transition function T will need to contain, at a minimum, the (discretized) positions of our team's robots and the position of the ball. It is not necessary that D^+ and D^- use the same state representation as T; in fact, I expect that D^+ and D^- will use a more abstract state representation (for instance, only the current position of the ball).

In the online phase of learning, we have two goals: recognize which play the opponent team is playing, and choose our own plays in response such that our probability of winning is maximized.

With the statistics that were computed offline, the opponent's plays can be recognized using an activity recognition technique such as hidden Markov models (HMMs). As the game progresses, our team gets observations $(s_1, s_2, s_3, \dots)$ of the state transitions and we know which plays $(p_1, p_2, p_3, \dots)$ our own team selected at each time step. The hidden variables are the play selections $(q_1, q_2, q_3, \dots)$ of the other team. We want to find the sequence of opponent play selections that maximizes the likelihood of the observed transitions, which is:

$$T(s_1, p_1, q_1, s_2) \times T(s_2, p_2, q_2, s_3) \times \cdots = \Pi_{i=1}^{n} T(s_i, p_i, q_i, s_{i+1}) \tag{5}$$

If we assume that the opponent team is static (always chooses the same play), we can trivially find the most likely opponent play as follows:

$$\arg\max_{q} \Pi_{i=1}^{n} T(s_i, p_i, q, s_{i+1}) \tag{6}$$

If we don't assume that the opponent team is static, we want to find the most likely sequence $\langle q_1, q_2, \dots, q_n \rangle$ of opponent plays. This is a straightforward HMM inference problem and can be solved with the Viterbi algorithm.

Given knowledge of the opponent team's play choice, we would like to choose our own plays such that the probability of winning is maximized. Here, we want to find a nonstationary policy π that maps from state, score difference, and time remaining to a play choice. To do this, we will use our estimates $\hat{D}^+$ and $\hat{D}^-$ of the time-to-score distributions, which were computed offline. Therefore, the state passed into π must be the same state representation that is used in the domain of D^+ and D^-. To compute π, I will have to make

some assumptions about the behavior of the opponent team—for instance, that they will continue selecting the most recent play for a period of time (perhaps until the next score). At every time step, we will find the most likely opponent play and then recompute π based on our current state. This will enable the team to automatically adapt to unforeseen circumstances (for instance, if the opponents change plays unexpectedly, or if it takes significantly longer to score than D^+ predicted).

In summary, I plan on improving performance against an unknown team through a hybrid offline/online learning approach. This approach assumes that the opponents choose from a fixed set of plays and that the opponent's play selections stay constant for a reasonable period of time. In the offline component, our team will try out all combinations of plays, and learn estimates of D^+ and D^- (the distributions of time-to-score) and T (the transition dynamics of the domain). Online, our team uses $\hat{T}$ to classify which play the opponent team is playing and uses $\hat{D}^+$ and $\hat{D}^-$ to find a nonstationary policy that maximizes the probability of winning.

4.2 Schedule

A tentative timeline for the completion of my thesis research is as follows:

- **Spring 2007**: Implement additional heuristics for timed, zero-sum domains and test these heuristics on some simple domains. Also investigate the use of existing MDP approximation techniques in these domains. Start developing the offline portion of the learning algorithm.
- **Summer 2007**: Develop the online portion of the learning algorithm and show that the hybrid offline/online approach works for a simple domain. Show that the heuristics and approximation techniques generalize to more complex domains.
- **Fall 2007**: Demonstrate that team performance can be improved with the hybrid offline/online approach in a full (but simulated) robot soccer scenario.
- **Spring/Summer 2008**: Implement and evaluate some of the above algorithms on a real robot soccer team. I may also work on any other interesting problems/results that have arisen from the work done so far.
- **Summer/Fall 2008**: Write and defend thesis.

References

[Bacchus et al., 1996] Bacchus, F., Boutilier, C., and Grove, A. (1996). Rewarding behaviors. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1160–1167, Portland, Oregon, USA. AAAI Press / The MIT Press.

[Bacchus et al., 1997] Bacchus, F., Boutilier, C., and Grove, A. (1997). Structured solution methods for non-Markovian decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*.

[Bagnell et al., 2003] Bagnell, J., Kakade, S., Ng, A., and Schneider, J. (2003). Policy search by dynamic programming. In *Neural Information Processing Systems*.

[Bellman, 1957] Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

[Bertsekas and Castanon, 1989] Bertsekas, D. P. and Castanon, D. A. (1989). Adaptive aggregation methos for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*.

[Boutilier et al., 1999] Boutilier, C., Dean, T., and Hanks, S. (1999). Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*.

[Boutilier et al., 1995] Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[Bowling et al., 2004a] Bowling, M., Browning, B., Chang, A., and Veloso, M. (2004a). Plays as team plans for coordination and adaptation. In Polani, D., Browning, B., Bonarini, A., and Yoshida, K., editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Computer Science*, pages 686–693. Springer Verlag, Berlin, Germany.

[Bowling et al., 2004b] Bowling, M., Browning, B., and Veloso, M. (2004b). Plays as team plans for coordination and adaptation. In *Proc. 14th Intl. Conf. on Automated Planning and Scheduling (ICAPS-04)*.

[Buro, 2006] Buro, M. (2006). Orts homepage. available online at http://www.cs.ualberta.ca/∼mburo/orts/.

[Chapman and Kaelbling, 1991] Chapman, D. and Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: an algorithm and performance comparisons. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[Dean et al., 1997] Dean, T., Givan, R., and Leach, S. M. (1997). Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings of Uncertainty in Artificial Intelligence*.

[Dean and Kanazawa, 1989] Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*.

[Dylla et al., 2005] Dylla, F., Ferrein, A., Lakemeyer, G., Murray, J., Obst, O., Röfer, T., Stolzenburg, F., Visser, U., and Wagner, T. (2005). Towards a league-independent qualitative soccer theory for RoboCup. In *RoboCup 2004: Robot Soccer World Cup VIII*, Lecture Notes in Computer Science. Springer Verlag, Berlin, Germany.

[Ferns et al., 2004] Ferns, N., Panangden, P., and Precup, D. (2004). Metrics for finite Markov decision processes. In *Proceedings of Uncertainty in Artificial Intelligence*.

[Gerkey and Mataric, 2004] Gerkey, B. P. and Mataric, M. J. (2004). On role allocation in RoboCup. In Polani, D., Browning, B., Bonarini, A., and Yoshida, K., editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Computer Science*, pages 43–53. Springer Verlag, Berlin, Germany.

[Givan et al., 2003] Givan, R., Dean, T., and Grieg, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*.

[Guestrin et al., 2003] Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*.

[Hanks and McDermott, 1994] Hanks, S. and McDermott, D. V. (1994). Modeling a dynamic and uncertain world I: Symbolic and probabilistic reasoning about change. *Artificial Intelligence*.

[Hoey et al., 1999] Hoey, J., St-Aubin, R., Hu, A., and Boutilier, C. (1999). SPUDD: Stochastic planning using decision diagrams. In *Proceedings of Uncertainty in Artificial Intelligence*.

[Hoey et al., 2000] Hoey, J., St-Aubin, R., Hu, A., and Boutilier, C. (2000). Optimal and approximate stochastic planning using decision diagrams. University of British Columbia Technical Report TR-00-05.

[Howard, 1960] Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press.

[Jong and Stone, 2005] Jong, N. K. and Stone, P. (2005). State abstraction discovery from irrelevant state variables. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[Kakade and Langford, 2002] Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *Proceedings of the International Conference on Machine Learning*.

[Kitano et al., 1997] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1997). RoboCup: The robot World Cup initiative. In *Proc. of the First Intl. Conf. on Autonomous Agents (Agents '97)*.

[Koller and Parr, 1999] Koller, D. and Parr, R. (1999). Computing factored value functions for policies in structured MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[Li et al., 2006] Li, L., Walsh, T. J., and Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *Ninth International Symposium on Artificial Intelligence and Mathematics*.

[Mahadevan, 1996] Mahadevan, S. (1996). Optimality criteria in reinforcement learning. In *Proceedings of the AAAI Fall Symposium on Learning Complex Behaviors in Adaptive Intelligent Systems*.

[McMillen et al., 2005] McMillen, C., Rybski, P., and Veloso, M. (2005). Levels of multi-robot coordination for dynamic environments. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Volume III*, pages 53–64. Kluwer Academic Publishers.

[McMillen and Veloso, 2006a] McMillen, C. and Veloso, M. (2006a). Distributed, play-based coordination for robot teams in dynamic environments. In *RoboCup 2006: Robot Soccer World Cup X*.

[McMillen and Veloso, 2006b] McMillen, C. and Veloso, M. (2006b). Distributed, play-based role assignment for robot teams in dynamic environments. In *Proceedings of Distributed Autonomous Robotic Systems*.

[Mundhenk et al., 2000] Mundhenk, M., Goldsmith, J., Lusena, C., and Allender, E. (2000). Complexity of finite-horizon Markov decision process problems. *Journal of the ACM*, 47(4):681–720.

[Puterman, 1994] Puterman, R. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

[RoboCup Technical Committee, 2007] RoboCup Technical Committee (2007). Sony four legged robot football league rule book Available online at: http://www.tzi.de/4legged/pub/Website/Downloads/Rules2007.pdf.

[St-Aubin et al., 2000] St-Aubin, R., Hoey, J., and Boutilier, C. (2000). APRICODD: Approximate policy construction using decision diagrams. In *Neural Information Processing Systems*, pages 1089–1095.

[Stone, 1998] Stone, P. (1998). *Layered Learning in Multi-Agent Systems*. PhD thesis, Carnegie Mellon University.

[Sutton et al., 1999] Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation.

[Thiebaux et al., 2002] Thiebaux, S., Kabanza, F., and Slaney, J. (2002). Anytime state-based solution methods for decision processes with non-Markovian rewards. In *Proceedings of Uncertainty in Artificial Intelligence*.

[Vail and Veloso, 2003] Vail, D. and Veloso, M. (2003). Dynamic multi-robot coordination. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Volume II*, pages 87–100. Kluwer Academic Publishers.

[Veloso et al., 2005] Veloso, M., Rybski, P. E., Chernova, S., McMillen, C., Fasola, J., von Hundelshausen, F., Vail, D., Trevor, A., Hauert, S., and Espinoza, R. R. (2005). CMDash'05: Team report. Available online at http://www.cs.cmu.edu/∼robosoccer/legged/reports/CMDash05-report.pdf.