Learning and Exploiting Relative Weaknesses of Opponent Agents

Shaul Markovitch (shaulm@cs.technion.ac.il) and Ronit Reger (ronitr@cs.technion.ac.il)
Computer Science Department,
Technion, Israel Institute of Technology

Abstract.

Agents in a competitive interaction can greatly benefit from adapting to a particular adversary, rather than using the same general strategy against all opponents. One method of such adaptation is *Opponent Modelling*, in which a model of an opponent is acquired and utilized as part of the agent's decision procedure in future interactions with this opponent. However, acquiring an accurate model of a complex opponent strategy may be computationally infeasible. In addition, if the learned model is not accurate, then using it to predict the opponent's actions may potentially harm the agent's strategy rather than improving it. We thus define the concept of *opponent weakness*, and present a method for learning a model of this simpler concept. We analyze examples of past behavior of an opponent in a particular domain, judging its actions using a trusted judge. We then infer a *weakness model* based on the opponent's actions relative to the domain state, and incorporate this model into our agent's decision procedure. We also make use of a similar *self weakness model*, allowing the agent to prefer states in which the opponent is weak and our agent strong; where we have a *relative advantage* over the opponent. Experimental results spanning two different test domains demonstrate the agents' improved performance when making use of the weakness models.

Keywords: Opponent Modelling, Multi-Agent Systems, Machine Learning

1. Introduction

> "Lasker sought to explore and make use of his opponent's weaknesses. Often he would choose the second or third strongest move, because this was the most unpleasant one for that particular opponent..."[1]

Consider a *multi-agent system* (MAS) in which several autonomous agents interact with each other in a competitive environment. Examples of such an interaction may be a bargaining environment of competing e-commerce agents, a Robocup robotic soccer game, or 2-player games such as checkers or chess. Each competing rational agent seeks

[1] A description of the style of the great chess master Emanuel Lasker. The quote is taken from a page about masters of chess (http://www.rgs.edu.sg/student/cca/chess/greatmasters.html).

to maximize its own utility, thereby defeating its opponents. However, in a realistic setting, agents normally possess only *bounded rationality*[2] (Russell and Wefald, 1991; Simon, 1982), and so the decisions they make are not always optimal.

The more knowledge an agent has about its environment, the better the strategy it is able to develop. In a multi-agent system, this should include knowledge about other agents. In a cooperative environment, the agent may receive this knowledge directly from its fellow agents. In a competitive environment, however, the agent is not likely to be given this information directly, since the other agents' strategies are private. One possible approach for dealing with this problem is to endow the agent with the capability of *adapting* to other agents in the system (Weiss and Sen, 1996).

The usage of learning techniques for adapting to other agents has received attention recently in the MAS research community (for a survey, see (Sen and Weiss, 1999)). The research in this field focuses on two central approaches: *model-based learning* (also known as Opponent Modelling), where an explicit model of the opponent strategy is generated and exploited (Carmel and Markovitch, 1998; Carmel and Markovitch, 1996c; Freund et al., 1995; Mor et al., 1996; Stone et al., 2000), and *model-free learning*, where the agent strategy is directly adapted based on observed behavior of opponent agents (Sandholm and Crites, 1995; Littman, 1994; Uther and Veloso, 1997). This distinction is also applicable to the more general reinforcement learning problem (Kaelbling et al., 1996), where both model-based (Sutton, 1990; Moore and Atkeson, 1993) and model-free approaches (Watkins, 1989; Watkins and Dayan, 1992) exist.

While the model-based approach is considered to be more data-efficient than the model-free approach (Atkeson and Santamaria, 1997; Carmel and Markovitch, 1999), it also suffers from two primary problems:

- *Complexity:* It is difficult to learn a complex strategy from examples. Even if the strategy is assumed to be relatively simple, e.g. modelled by a deterministic finite-state automata (DFA), it is still hard to learn (Angluin, 1978). Most agent strategies in a real settings are more complex than a DFA, and so learning a model of such a strategy becomes computationally infeasible.

- *Risk:* The model is ultimately used to anticipate the opponent agent's decisions, or to simulate its actions. If, however, the model

[2] A boundedly rational agent tries to get the maximal utility achievable within its resource constraints.

is not entirely accurate, then relying on its predictions may harm the agent's performance rather than improving it (Iida et al., 1994). Note that even in the unlikely event that the agent possesses an exact model of its opponent, utilizing it will not guarantee an exact prediction due to the limited simulation resources available during a real interaction.

This paper presents a new methodology for Opponent Modelling, that addresses the problematic issues of both complexity and risk. To contend with the complexity of learning a full opponent model, we learn only a certain *aspect* of the opponent's strategy: the opponent *weakness*. Assuming our opponent is a *boundedly rational agent* whose quality of decision is not uniform over all domain states, we attempt to characterize the set of states in which the opponent's performance is relatively inferior. In order to reduce the risk of using a faulty model, we use the model only to *bias* the agent's action decisions, in a minimally risky way. Thus, even if the model is not accurate with respect to the opponent's behavior, its use cannot harm the agent's performance significantly. In addition, we consider states in which the opponent suffers from weakness, but our own agent's strategy is expected to fair well. In other words, we take advantage of points at which our agent exhibits a *relative advantage* over the opponent. We show that this adaptation improves the modelling agent's overall performance in a competitive interaction.

The rest of this paper is organized as follows: In Section 2, we explain our approach and the algorithm that we have developed, including a description of the system that implements the algorithm. In Section 3 we present experimental results. In Section 4, we present some related work in the field of *Opponent Modelling*, and discuss our contribution with respect to the field. Finally, in the Discussion section, we present our conclusions, and some directions for further work.

2. Learning and Exploiting a Model of Opponent Weakness

Given the assumption that an opponent agent is *fallible*, and furthermore does not act with an evenly distributed level of strength over all domain states, our aim is to learn a model of the opponent's weaknesses, and exploit this knowledge in a competitive interaction. The strategy is adapted to the opponent based on observed past behavior. We are working within the framework of the *model-based approach*, so that we acquire a model of the opponent's strategy, and adapt our strategy to make use of this model.

The approach we propose is as follows. We start by defining the concept of *opponent weakness*, or the set of states in which the opponent is relatively fallible. We then tag the example states we have, by evaluating the opponent actions using a *teacher*. The states are then converted to feature vectors, and we use a standard induction algorithm to learn the *weakness* concept. Finally, we incorporate the inferred model into our agent's decision procedure, by using it to bias its action selection towards states where the opponent exhibits weakness.

2.1. The Concept of Opponent Weakness

Let S be the set of all possible states. Let A be the set of all possible actions, where for each $a \in A$, $a : S \to S$. Let $\varphi : S \to A$ be an agent decision function. Let $T : S \to A$ be a *teacher* decision function. We say that the agent strategy φ is *weak* at state s with respect to teacher T if and only if their decisions at state s are different, i.e.

$$Weakness_{\varphi,T}(s) \Leftrightarrow \varphi(s) \neq T(s) \tag{1}$$

Applying the definition of this concept, then, reduces the problem of Opponent Modelling to that of concept learning[3].

This definition, however, may not be adequate when there are several best or almost-best actions, since the agent may select a strong action that is different than the teacher's action choice.

If we are given, in addition to T, a teacher utility function $U_T : S \to \Re$, then we can define the *weakness function* as

$$Weakness_{\varphi,T,U_T}(s) \stackrel{def}{=} U_T(a_{T,s}(s)) - U_T(a_{\varphi,s}(s)) \tag{2}$$

(where, for readability, we denote $T(s)$ by $a_{T,s}$ and $\varphi(s)$ by $a_{\varphi,s}$).

We have now expressed the problem of Opponent Modelling as a problem of learning a real-valued function. If, however, we are interested in remaining within the *concept learning* paradigm, we can modify this definition as follows. Let $0 \leq \epsilon \leq 1$. Let $b_\epsilon(S)$ be the maximal b such that $\frac{\left|\left\{s \in S | Weakness_{\varphi,T,U_T}(s) \geq b\right\}\right|}{|S|} \geq \epsilon$. Now we can define weakness as a binary concept:

$$Weakness_{\varphi,T,U_T}(s) \Leftrightarrow U_T(a_{T,s}(s)) - U_T(a_{\varphi,s}(s)) \geq b_\epsilon(S) \tag{3}$$

Thus, if we order the states in S according to the utility difference function, then the upper $\epsilon \cdot 100$ percentile is considered weak.

[3] Note that this definition implies an assumption that *weakness* is Markovian, whether the agent strategy is Markovian or not.

The given definition of *weakness* leads to a learning problem that is intuitively simpler than the general *Opponent Modelling* approach of learning an arbitrary decision procedure. Ideally, the concept should be defined with respect to a teacher that has an *optimal* decision procedure, according to the global utility which the agent tries to maximize. In practice, the availability of such an oracle is unlikely, and so we must rely on a teacher that is better (stronger) than the agent in the sense that it yields better global utility.

2.2. Learning a Model

In the previous sub-section, we reduced the problem of opponent modelling to the problem of concept learning.

In this sub-section, we will discuss the issues associated with applying the inductive concept learning paradigm to the problem of learning weaknesses.

Inductive concept learning consists of the following stages:

1. Collecting a set of examples
2. Tagging the examples
3. Converting the examples to feature vector representation
4. Using an induction algorithm to infer a classifier

The rest of this sub-section will discuss each of these stages with respect to the particular problem of weakness induction.

2.2.1. *Where the examples come from*

The examples of opponent behavior are pairs, each consisting of a domain state and an action that the opponent was observed to take at that state.

These can be acquired either from a history of past interactions of the opponent with the agent, or from observing the opponent in interactions with other agents. Consider, for example, the domain of chess, in which there are large databases available of recorded games for a given opponent. Alternatively, the agent can refer to records of all previous chess games that it has played against the particular opponent. Another example is an agent participating in electronic commerce. In this case, the agent may spend some time observing the actions of a foreseeable opponent, and attempt to accumulate examples of behavior in a wide range of situations.

2.2.2. *The Teacher*
The function of the *teacher* is to give us an evaluation of opponent actions, according to Equations 1, 2, or 3. The teacher should be an agent whose decision function is more accurate than the one used by the opponent. We assume an *offline* learning framework, where resources are not as limited as in a real interaction setting. In many cases, our own agent's decision procedure is a *contract algorithm* (Zilberstein, 1995) that improves with a larger allocation of resources, (e.g. any form of lookahead search). Such a decision procedure may be used as a teacher, by allocating it resources that are significantly greater than those assumed to be available to the opponent agent when choosing the example action.

The framework we employ depends on the monotonicity of the performance profile of the teacher, i.e. that the quality of decision improves with greater resource allocation. The question of the asymptotic behavior of the performance profile has received much attention in the game-playing research community. Nau (1980) originally presented a game searching pathology as it relates to the Minimax search strategy. In practice, however, we have found, as have many other researchers, (Nau, 1982; Pearl, 1983) that this "pathology" does not appear to provide a performance obstacle in most cases. Thus, at least in the game-playing context, it is justifiable to use deep search as a teacher. Note that recent studies have shown diminishing returns for increasing search depth (Junghanns and Schaeffer, 1997). In such cases, the extra resources may be better invested in the knowledge aspect (e.g. utility function) rather than increased search depth.

2.2.3. *Characterizing Domain States*
The features used to characterize a domain state clearly play an important role in the quality of the classifier being learned. Ideally, the features should have some predictive power regarding the class and should be easy and efficient to compute, in order to reduce strategy execution overhead. They can either be manually constructed and supplied as external knowledge to the system, or can be automatically generated using one of the known feature-generation techniques (Markovitch and Rosenstein, 2001; Hu and Kibler, 1996; Matheus and Rendell, 1989; Pagallo and Haussler, 1990).

2.2.4. *Induction Algorithms*
Any of the available induction algorithms may be applied to this problem. Our primary concern with regard to the learned model is its classification efficiency, since it will ultimately be used by the decision procedure of a bounded rational agent. In our experiments (described

in the next section), we therefore use decision trees (Quinlan, 1986), which are known to be efficient classifiers.

If we decide to represent *weakness* as a real-valued function (Equation 2) then we can use regression trees instead of decision trees.

2.3. Usage of the Weakness Model

Once a classifier has been learned, our agent possesses a model of opponent weaknesses, and may use this model in competitive interaction with the opponent. There are several options for incorporation of the learned model into the agent's decision strategy. In the remainder of this subsection, we will describe three such alternatives.

2.3.1. *Window-Based Usage Method*

One option for usage is to consult the weakness model to decide among a *window* of two or more immediate actions that are considered to be "best" or "almost best" by the agent. Under the binary classification scheme we choose an action that leads to a state that is classified as belonging to the opponent *weakness*. This usage scheme is illustrated in Figure 1. Note the similarity of this approach to the strategy employed by Lasker, the chess master, as described in the quotation at the beginning of the paper. If the weakness is modeled by a continuous function, then we choose the action leading to a state with the highest weakness value.

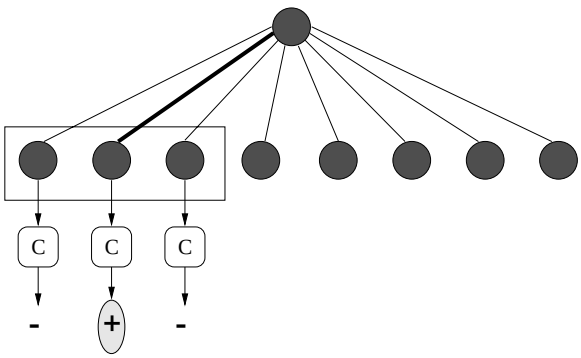

Figure 1. Using the acquired model to select among a window of best, or "almost-best" actions. Each next state is passed to the learned model, and the agent prefers states in which the opponent is classified as weak.

If the agent's decision procedure can be modified to return the set of best actions (before tie-breaking is performed) instead of a single action,

then we can use this set as the *window*. If the decision procedure also associates a utility value with each action, then the window may consist of the set of immediate actions that have maximal, or near-maximal utility. The window-based selection algorithm is shown in Figure 2. If we decide to represent *weakness* as a real-valued function (Equation 2) then we select an action in the window with maximal weakness value.

Algorithm $Select\_Action(state)$
  $\overline{A} \leftarrow legal\_actions(state)$
  $\overline{S} \leftarrow \{a(state) | a \in \overline{A}\}$
  $M \leftarrow \max_{s' \in \overline{S}}(u(s'))$
  $m \leftarrow \min_{s' \in \overline{S}}(u(s'))$
  if $M = m$
    return $Random\_Select(\overline{A})$
  $Window \leftarrow \{s \in \overline{S} | \frac{M-u(s)}{M-m} \leq \delta_w\}$
  $W_{weak} \leftarrow \{s \in Window | Weakness(s) = T\}$
  if $W_{weak} \neq \{\}$
    $A' \leftarrow \{a \in \overline{A} | a(state) \in W_{weak}\}$
    return $(Random\_Select(A')$
  else
    return $(Random\_Select(\overline{A}))$

Figure 2. The modelling agent's algorithm for determining and selecting from the window of "almost best" actions.

There exists a trade-off in the size of the window, in which a smaller window means less risk to the agent's original strategy, along with less potential improvement in using the weakness model. This tradeoff is determined by the δ_w parameter, which determines the maximal deviation from the best utility allowed in the window.

2.3.2. *Utility-Function Method*
A possible drawback of the above approach is that it only tests the weakness of states that are in the immediate neighborhood of the current state. Often, the agent decision procedure performs some type of lookahead, which evaluates states that are several steps ahead in the interaction. In such a case, our strategy may be more effective if it is applied to states at the search horizon.

One way of achieving this goal is by incorporating the weakness model into the agent's utility function. We may do so, for example, by linear combination of the original utility evaluation and the model prediction term. Assume that $u : S \rightarrow [0, 1]$ is the agent's original

utility function, (where S is the set of states). Assume that c is the classification term returned by the model, which equals 1 for weak states and 0 otherwise (or any number in between, if real-valued weakness is learned). Then the agent's new utility function is: $u'(s) = (1-\delta_u)\cdot u(s)+\delta_u \cdot c(s)$, where $0 \leq \delta_u \leq 1$ is the weight of the modelling term. δ_u determines the risk associated with using the weakness model.

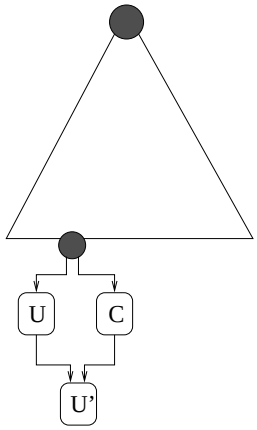

Figure 3. Using the acquired model to affect the value of the agent's utility function. The result of the classifier is combined with the original utility value to generate a new utility value.

The advantage of the "deep" method, as compared to the window-based approach, is that the prediction of the weakness model is likely to be more indicative, since it considers states that are at the boundary of the lookahead search. This allows the agent to systematically lead the interaction towards these "desirable" states. The disadvantage of the method is the high computational cost associated with it, due to the potentially exponential number of applications of the weakness classifier. This overhead is directly dependent on the ratio between the cost of applying the classifier and the cost of applying the utility function.

2.3.3. *Hybrid Usage Approach*
A desirable compromise to the two above approaches would keep the low computational cost of the window-based approach, while considering states from the search boundary. This may be done by selecting from a window of immediate actions, but determining the weakness value of each action differently. Instead of applying the weakness classifier to the immediate state resulting from the candidate action, we consult the model on the state at the leaf level that is responsible for the value associated with that action.

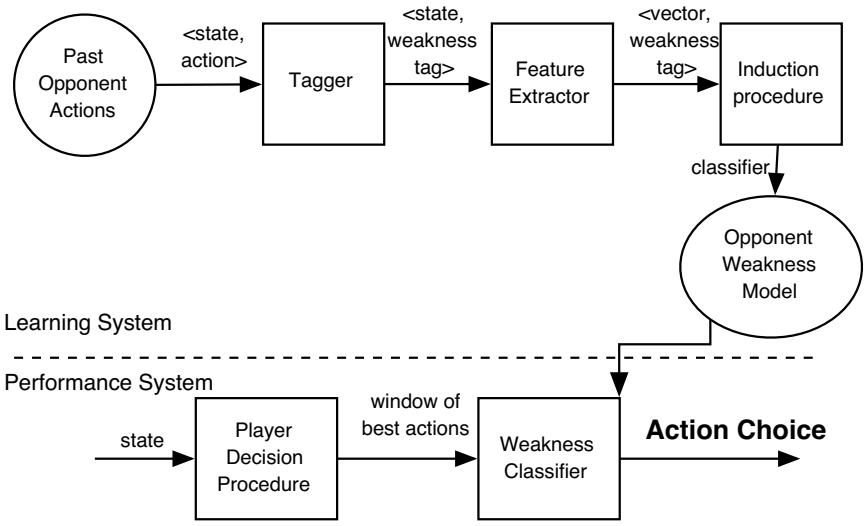

Figure 4. Data-Flow of the opponent weakness modelling algorithm, window-based usage method

This can be done by modifying the agent's lookahead procedure, such that, in addition to each backed-up value, it also propagates the corresponding state[4]. Thus, the model can be applied to these *propagated* states. In the proposed approach, we are consulting the model on states deep in the search tree (which carries its benefit, as described above), and yet applying it only at the top level of the search tree, thereby applying it few times.

A data-flow diagram of the entire algorithm described is shown in Figure 4.

2.4. Incorporation of a Self Model

The opponent weakness model, as learned and utilized above, fails to take into consideration the possibility that our agent may exhibit weakness in those same states where the opponent is fallible. A complicated domain state, for example, may make it difficult for our own agent to select a good action, as it does for the opponent.

We would like a method that allows the agent to recognize situations in which the opponent is weak, but our agent is not. While the agent has access to its own decision procedure, it cannot test for weakness during a real interaction due to resource limitation. Thus, we suggest learning

[4] A similar propagation procedure was used by Hsu et al. (1990) for learning an evaluation function from a grandmaster game database.

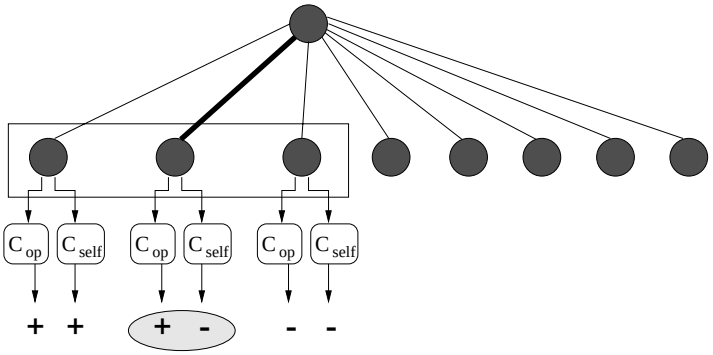

Figure 5. Using both the opponent model and the self model to select between a window of "almost best" actions. Both the self model and opponent model affect the agent's choice.

a model of our own agent's weaknesses – a *self model*, in addition to the opponent weakness model.

This model is learned in a similar manner to the learning of the opponent model, by observing examples of *self behavior* and inferring a classifier. We can use the same teacher that is used to judge opponent actions to tag our own agent's action choices. During offline learning, the teacher can be allocated more resources than are available to the agent in a realistic interaction, and therefore its evaluation is of higher quality than that of the agent.

The learned *self model* can be incorporated into our agent's decision procedure alongside the opponent model. Figure 5 illustrates this idea. In addition to consulting the opponent model to select among a window of actions, we now consult the self model as well, and prefer states in which the opponent is predicted to be weak and our agent is predicted to be strong. Figure 6 formalizes the agent's new action selection algorithm.

The *self model* can also be incorporated into the utility function method by adding it as an additional factor to the linear combination. A data flow diagram describing the revised algorithm is shown in Figure 7.

3. Empirical Study

In order to demonstrate the effectiveness of the described method, we have designed and executed a series of experiments, investigating two different domains. The domains used in our experiments have a fairly

Algorithm $Select\_Action(state)$
  $\overline{A} \leftarrow legal\_actions(state)$
  $\overline{S} \leftarrow \{a(state) | a \in \overline{A}\}$
  $M \leftarrow \max_{s' \in \overline{S}}(u(s'))$
  $m \leftarrow \min_{s' \in \overline{S}}(u(s'))$
  if $M = m$
    return $Random\_Select(\overline{A})$
  $Window \leftarrow \{s \in \overline{S} | \frac{M-u(s)}{M-m} \leq \delta_w\}$
  $Op_{weak} \leftarrow \{s \in Window | Weakness_{op}(s) = T\}$
  $Self_{weak} \leftarrow \{s \in Window | Weakness_{self}(s) = T\}$
  if $Op_{weak} \setminus Self_{weak} \neq \{\}$
    $A'_1 \leftarrow \{a \in \overline{A} | a(state) \in Op_{weak} \setminus Self_{weak}\}$
    return $Random\_Select(A'_1)$
  else $Equiv \leftarrow (Self_{weak} \cap Op_{weak}) \cup (\overline{A} \setminus Self_{weak} \cap \overline{A} \setminus Op_{weak})$
    if $Equiv \neq \{\}$
      $A'_2 \leftarrow \{a \in \overline{A} | a(state) \in Equiv\}$
      return $Random\_Select(A'_2)$
    else
      return $Random\_Select(\overline{A})$

Figure 6. The modelling agent's algorithm for determining and selecting from the window of "almost best" actions, when using both the opponent model and self model.

simple, discrete representation, thus allowing the characterization of domain states by a limited number of features.

3.1. Experimental Methodology

Each set of experiments consists of an offline learning phase, and a series of tournaments representing the performance phase. We gather domain state examples to build the weakness models of a particular agent in a given domain, and use the acquired models during the tournaments.

The teacher used is the modelling agent's original search procedure (and heuristic evaluation function) to a greater depth. We make no direct assumption about the search depth of the opponent action decisions being analyzed, nor do we assume that the opponent searches to a fixed depth. Furthermore, we need not even presume that the agent uses any kind of lookahead search in its decision procedure. The only assumption made is that the quality of the teacher's decision is higher than that of the opponent, due to its greater resource allocation. We

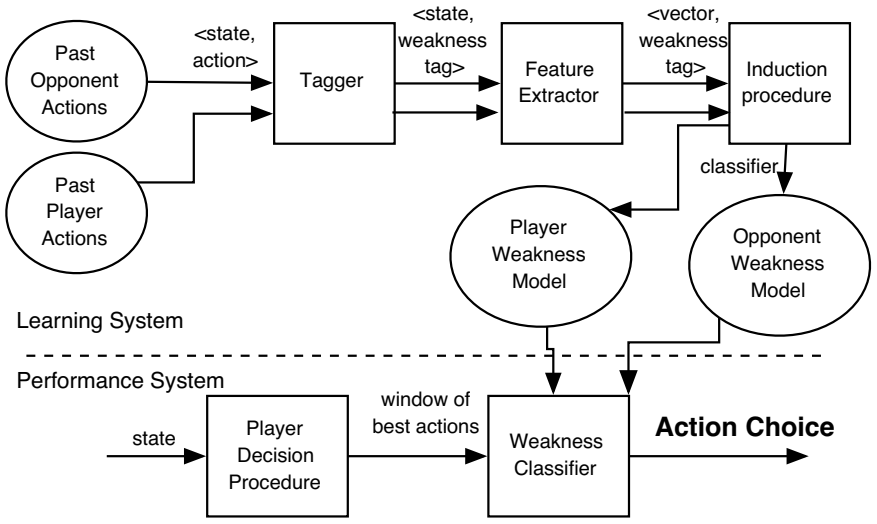

Figure 7. Data-Flow of the opponent weakness and self model algorithm (window-based usage method)

can thus depend on the teacher to effectively judge the quality of the opponent's action decisions.

The agent's performance in actual competition (a series of games) is the measure we use to compare the performance of the *modelling agent* with that of the original agent. The opponent they face uses a fixed strategy. Note that the tournament takes place under the same time constraint conditions as the examples used in the learning phase.

3.2. Outline of the Empirical Study

The experiments are intended to test the effect of learning a weakness model and incorporating it into the agent's strategy. The architecture of a learning agent that uses our methodology is affected by numerous parameters. The goal of the empirical study is to test the effect of these parameters on the agent's performance. We start by conducting a *basic experiment*, which applies 'default' values to each of the system parameters. We follow by performing a set of parametric experiments, where all of the parameters are fixed except for the one we are testing.

We first test various parameters that affect the learning process. These include the allocated learning resources, the quality of the teacher, and the feature set used for classification. We then examine parameters associated with the usage of the model, namely the model-usage strategy, the risk factor, and the incorporation of a self-model. We also test

parameters of the agent itself, including the heuristic function used in the agent strategy and the action time limit.

A detailed description of the various experiment parameters follows.

3.2.1. *Dependent Variables*

– *Performance Measure:* In order to estimate the performance of an agent in a real interaction, we perform a finite tournament (set of games) between the agent and a fixed opponent.

  - If W is the number of wins of the measured agent in a tournament of n games, and T is the number of ties, we define its performance in the tournament as: $\frac{W+\frac{T}{2}}{n}$. We consider each tournament as a sequence of binomial experiments, and use standard statistical methods to calculate the number of experiments needed to achieve accurate estimates with high confidence. We run 10,000 games in each tournament in order to achieve a confidence interval of ± 0.01 with a probability of 0.95.

– *Model accuracy:* While the ultimate measure of success of the system is performance in a real interaction, as described above, it is also interesting to monitor the accuracy of the learned classifiers. We do so using the standard 10-fold cross-validation test.

3.2.2. *Independent Variables*

– *Allocated Learning Resources:* The resources invested in learning the model of opponent weaknesses. This is controlled by two parameters:

  - *Number of examples:* The number of opponent action examples used to build the model.
  - *Teacher depth:* The search depth of the teacher determines the quality of the tagging.

– *Model-Usage Method:* The three methods of model-incorporation described in Section 2.3, i.e.

  - *Window-Based method*
  - *Utility Function method*
  - *Propagated-States method*

– *Risk Factor:* The magnitude of the effect of the model on the agent's decision procedure. For the window-based and propagated-states methods, it is determined by δ_w, and for the utility-function method by δ_u. Both values are between 0 and 1, where a larger value indicates more weight given to the weakness model. $\delta_w = 1$ and $\delta_u = 1$ imply decision making based only on the weakness model, while $\delta_w = 0$ and $\delta_u = \epsilon$ mean that the weakness model is used only for tie-breaking.

– *Domain:* We use two 2-player, zero-sum, perfect information games - *Connect-Four*[5] and *checkers*.

– *Feature Set:* The domain-specific features used to build and reference the weakness classifier.

  - *Simple:* Features based on a simple representation of the domain state, or elements computed by a typical evaluation function.
  - *Pattern-based:* Features based on spatial patterns of the domain state. All possible combinations of a certain geometric pattern may yield a very large number of features. Therefore, a limited number of such patterns may be "discovered" by pruning the larger set, using an automatic feature extraction system.

– *Agents' Search Strategies:* Both the agents and the opponents make use of a lookahead-based search strategy, including a heuristic evaluation function. Thus, the strategy is determined by two parameters:

  - *Search method:* We tested 2 types of search methods:
    * Fixed-depth $\alpha\beta$-search
    * Time-limited iterative-deepening $\alpha\beta$-search
  - *Heuristic function:* We used 3 different heuristic functions for each of the 2 domains (identified as H1, H2, and H3 in *Connect Four*, and C1, C2, and C3 in *checkers*) [6].

[5] Connect Four is played on a vertical 6X7 matrix. On each turn, a player drops a disc into one of the 7 columns. The first player that succeeds to form a sequence of 4 discs either vertically, horizontally, or diagonally wins the game. We make note that the game of Connect-Four is actually a solved game (Allis, 1988). However, it can still be used as a relevant test domain for *bounded-rational agents*.

[6] The *checkers* functions use a linear combination of features such as material-advantage, number-of-threats, center-control and king-pawn-distance with different weights. The functions for *connect four* use a linear combination of features that

– *Time Limit:* A time limit may be enforced for each action decision made by the agents in a tournament. This parameter tests the effect on performance caused by the computational overhead of consulting the weakness model.

3.2.3. *Default Parameter Values*

Unless otherwise specified, all experiments were performed with the following defaults:

– *Usage-Strategy:* Window-based

– *Risk Factor:* $\delta_w = 0$, $\delta_u = 0.1$

– *Agent Search Strategy:* Fixed-Depth $\alpha\beta$-Search, using H2 in *Connect Four* and C1 in *checkers*.

– *Opponent Search Strategy:* Fixed-Depth $\alpha\beta$-Search, using H3 in *Connect Four* and C3 in *checkers*.

– *Agent and Opponent Search Depth*[7] $= 3$

– *Teacher Search Depth* $= 6$

– *Feature Set:* Simple Features (non-pattern-based features)

– *Number of Examples to Learn Classifier:* 10000

– *Number of Games per Tournament:* 10000

3.3. Effectiveness of the Weakness Modelling Approach

The *basic experiment* demonstrates the performance improvement achieved by a modelling player, upon learning and utilizing a model of opponent weaknesses.

The result is shown in Table I. We can see that the use of the opponent weakness model clearly provides the agent with a performance benefit. Note also that both results are statistically significant.

Due to the low-risk attitude of our approach, we do not expect a huge leap in performance as a result of applying the modelling strategy.

count the number of four squares in a row (to any direction) that can still be filled to a one-color row. There is one feature for those filled with 3 pieces, one for 2 pieces etc.

[7] Search depth is limited to 3 in order to allow each tournament competition to consist of 10000 games (and have each tournament complete in a reasonable amount of time).

Table I. *Results comparing players' performances with and without modelling. The results are averaged over 10,000 games. We report confidence intervals for p=0.05.*

	Before Modelling	After Modelling
Connect-Four	0.556 $\pm$0.0097	0.629 $\pm$0.0095
Checkers	0.432 $\pm$0.0097	0.495 $\pm$0.0098

The benefit of modelling, however, is sufficient to shift the agent from a clearly inferior position to a fairly even one in the *checkers* domain, and from a fairly even position in the *Connect-Four* domain to one of significant superiority.

To test the sensitivity of our approach to the particular selection of heuristic functions, we repeated the above experiment, varying the heuristic functions for both the modelling agent and the opponent. Three different heuristic functions were used for each, thus yielding 9 combinations. Figure 8 summarizes the results of this experiment, for both domains.

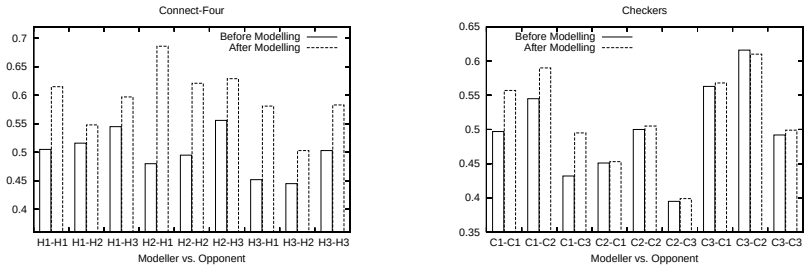

Figure 8. 3 different heuristic functions used by the modelling and modelled agents

We find that the benefit of the approach is not sensitive to the particular selection of heuristic functions, though the method appears to be more robust in the *Connect-Four* domain than in the *checkers* domain.

3.4. Empirical Study of the Learning Algorithm

In this section, we report the results of several experiments performed to gain an understanding of various aspects of the learning process. First, we would like to validate that our learning algorithm exhibits the typical behavior of performance improvement as a result of increased

learning resources. In our framework, the total learning resources are determined by two factors: the number of examples used and the tagging resources invested in each example. Also, since the success of a learning process depends on the quality of the features used, we study the performance of our learning algorithm with different types of features.

3.4.1. *Learning Curves*
In this experiment, we varied the number of training examples used for model induction, and tested the performance of the modelling agent with the induced models. Such an experiment yields a learning curve. Figure 9 shows an average of 10 such learning curves for both the *Connect-Four* and *checkers* domains.

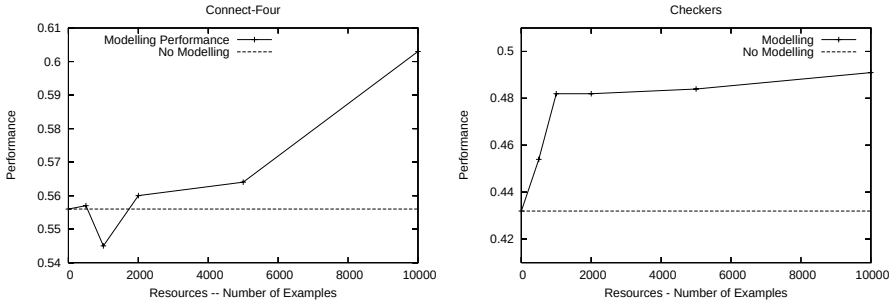

Figure 9. The performance of the modelling agent as a function of the learning resources)

The graphs represent typical learning curves. The only notable exception is a slight decline in performance in the *Connect-Four* domain with a small number of examples.

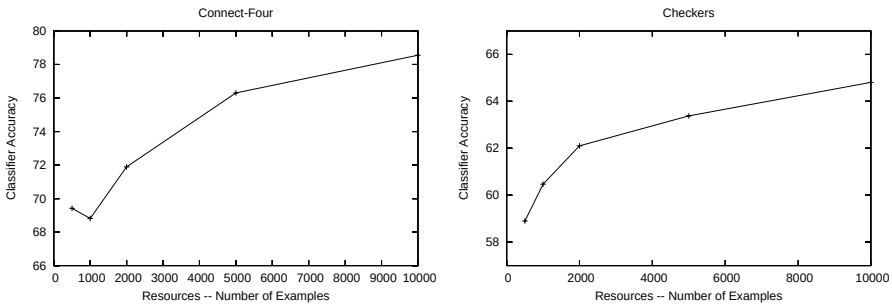

Figure 10. The accuracy of the induced model as a function of the learning resources.

We also tested the accuracy of the resulting classifiers, using a standard 10-fold cross-validation test. Figure 10 shows the learning curves

with respect to accuracy for the two domains. We note that in the *checkers* domain, despite the relatively low classifier accuracy, its use still yields an increase in performance. To understand why, recall that $\delta_w = 0$ implies that the weakness model is used as a tie-breaker. Thus, using a classifier with 64% accuracy has a higher probability of selecting an action leading to opponent weakness than random selection.

3.4.2. *The Effect of Teacher Quality on Performance*
In this subsection, we test the performance effect of the resources invested in the tagging procedure by varying the teacher's search depth limit. Figure 11 illustrates the effects of teacher search depth on performance in the two domains. As expected, the greater the resource investment by the teacher, in the form of greater search depth used to evaluate each opponent action example, the better the performance of the resulting classifier when used by a modelling player in an interaction.

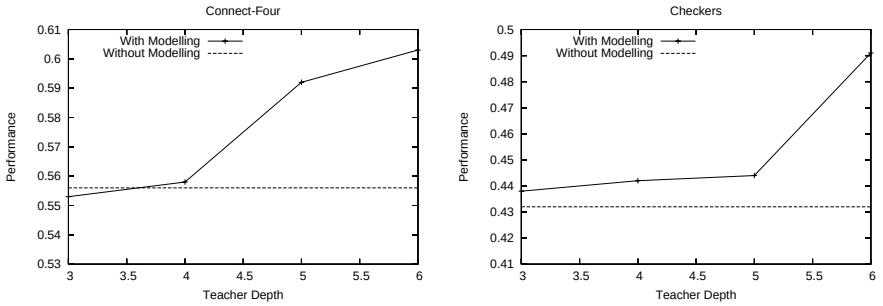

Figure 11. A comparison of various teacher depths on the performance of models

3.4.3. *The Effect of Feature Sets on Performance*
The success of the weakness-learning algorithm is dependent on the availability of a domain-specific feature set that is able to have some predictive power with regard to the weakness concept. We have tested our system with two types of features: *pre-defined features* and *automatically-generated features.*

3.4.3.1. *Pre-Defined Features* We experimented with two types of pre-defined features: features that are used to describe the actual states, such as the set of pieces occupying the game board, and statistical features that are usually used as terms in evaluation functions, such as material advantage.

In both domains, we made use of "simple features", which are a simple description of pieces on the game board. Despite the simplicity of

this feature set, it has the advantage of being very efficient to compute, and presents a concise way of describing the game state. In addition, a learned model may pick up on some patterns of pieces, in the way the tree is developed.

Additionally, in the checkers domain we utilized the set of features used in the evaluation function, such as material difference, control over the center of the board, threats, etc.

3.4.3.2. *Automatically-Generated Features* An alternative approach to the use of predefined features is the automatic generation of features. Many methods have been developed for feature construction, (for example, (Markovitch and Rosenstein, 2001; Hu and Kibler, 1996; Matheus and Rendell, 1989; Pagallo and Haussler, 1990)). We utilized the approach of Markovitch and Sella (1996). This approach searches the space of pattern-based features, which are spatial patterns of a predetermined format that are deduced from the given domain state, and collects a fixed number of patterns to make up the feature set[8].

Each of the pattern-based feature sets used undergoes pruning before it is used by the model-learning algorithm, since the number of all possible configurations of a certain pattern type is typically much too high to be used as a complete feature set[9]. Thus, the method of using board patterns as features involves generating the possible patterns, pruning them to an acceptable number (by computing their information gain on a given set of tagged examples), and then using a vector of pattern features to build the model and to use it. Each pattern is counted in the number of times it appears on a board, and the resulting count for each pattern represents the feature vector. Alternatively, the value for a particular pattern may be binary, i.e. 1 if the pattern appears on the board and 0 if it does not.

A pattern in the *Connect-Four* domain is defined as a sequence of four elements. Each element is either an empty square, white piece, black piece, or wild-card (any). A pattern is matched with every sequence of four squares on the board, (horizontal, vertical, and diagonal).

Each pattern in the *checkers* domain consists of a 3 X 3 sub-board that contains 5 checkers squares, and a position specification. Each of the sub-board elements can have one of the values: don't care, free, any-white, any-black, white-pawn, white-king, black-pawn, black-king. The value don't-care is more general than the value any-white, which in turn

[8] The patterns are used as features for building the opponent model only - they are not used in the player's evaluation function.

[9] While decision trees are known to be more tolerant to irrelevant features, their performance still deteriorates when the number of such features becomes too high.

is more general than the value white-pawn. The position component can take one of the following values: don't care, white-baseline, black-baseline, middle-court, right-edge, left-edge. Each pattern is matched with all the sub-boards located according to its position specification. For example, for the white-baseline position there are three possible sub-boards with which to match.

3.4.3.3. *Results* The *basic experiment* has been repeated using each of the described feature sets. Table II shows results of this experiment.

Table II. *Results comparing modellers' performances with various feature sets. We report confidence intervals for p=0.05.*

	Feature Set	Modeller's Performance
Connect-Four	No Modelling	$0.556_{\pm 0.0097}$
	Default (Simple-Connect Features)	$0.629_{\pm 0.0095}$
	Mask-Pattern Features	$0.583_{\pm 0.0096}$
Checkers	No Modelling	$0.432_{\pm 0.0097}$
	Default (Checkers-Evaluation Features)	$0.495_{\pm 0.0098}$
	Simple-Checkers Features	$0.482_{\pm 0.0098}$
	5-Pattern Features	$0.524_{\pm 0.0098}$

In both domains, we find that the automatically-generated pattern-based features were informative in the sense that they contributed to a performance improvement. In the *checkers* domain, the pattern features used yielded a significant performance increase over the use of the *Simple Features*, and also over the *Evaluation Features*. In the *Connect-Four* domain, however, the result achieved with the *Simple-Connect Features* was better. A possible explanation would be that the decision tree mechanism enabled the generation of more sophisticated patterns than our rather simple pattern language.

One example of a simple feature that was found to be "effective" in the *Connect-Four* domain was board square #22, which is the square in the second row from the bottom, in the central column. This board square, apparently, is an important one in determining the position/ weakness of a player in a game. With a simple game-board analysis, we know that this is an important square to occupy in the game, and so this feature's significance is intuitive.

An example of an automatically-generated pattern-based feature that was found to be effective in the *checkers* domain is the pattern illustrated in Figure 12. Here, the machine clearly discovered the im-

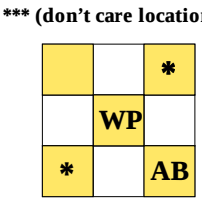

*Figure 12. Threat Pattern - discovered by the 5-Pattern automatic feature-extraction procedure. WP stands for White Pawn, AB stands for Any Black, and * stands for don't care. The value of the position component is "don't care".*

portance of the "threat" concept, where the black piece is in a position to attack the white pawn in the next move.

3.5. Empirical Study of the Performance System

This section presents results of experiments that were performed to study the various ways that an opponent model can be used once acquired. In these experiments, we attempt to answer the following four research questions:

1. Is using the model deep in the lookahead search more effective than using it for the immediate states?
2. If so, does the overhead associated with deep evaluation outweigh this advantage when time is restricted?
3. Does the hybrid usage method provide a good compromise between deep evaluation and time efficiency?
4. How does the value of the risk factor affect the performance of the modelling agent?

Table III. *Comparing the window-based model-usage to the utility-function method. Confidence intervals are reported for p=0.05.*

	Before Modelling	Modelling - window-based	Modelling - utility-function
Connect-Four	0.556 ±0.0097	0.629 ±0.0095	0.690 ±0.0091
Checkers	0.432 ±0.0097	0.495 ±0.0098	0.537 ±0.0098

Table III presents the *basic experiment* re-performed, with the modelling agent applying the utility-function model-usage strategy. We

observe drastic performance improvements in both domains with the application of the utility-function strategy, proving this method to indeed be very powerful.

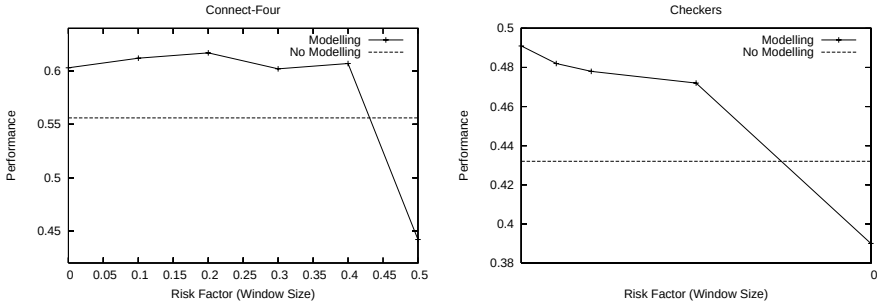

Figure 13. Testing the δ_w parameter, for determining the top level window from which the model selects the next action

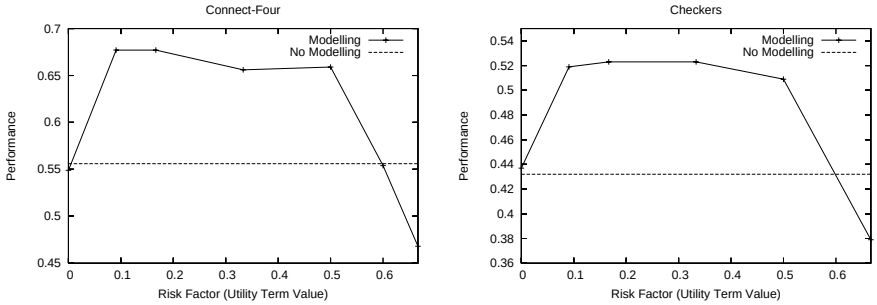

Figure 14. Testing the δ_u parameter for utility-function level usage of the weakness model.

Figure 13 and Figure 14 show the performance of both model-usage methods with varying values of the risk factors.

The results for the utility-function method (Figure 14) show the expected "upside-down U-shaped" pattern, where taking too little risk reduces the benefit of modelling, and taking too much risk is too great a deviation from the Minimax strategy.

The performance of the window-based method surprisingly does not deteriorate even with a risk factor of 0. We attribute this method's success with $\delta_w = 0$ primarily to the "simplicity" of the heuristic functions used by the modelling agents. The heuristic functions used return equal values for many different states, and so there are always several states with equal value in the *window*. However, in the general case, in which more sophisticated heuristic functions are used, then we predict that the "0-window" will be much less effective (because there will rarely be states with equal value).

One problem with the result shown in Table III is that using search depth as a resource limit does not take into account the overhead incurred in the utility-function method from applying the model at the leaf level of the search tree.

The hybrid usage approach described in Section 2.3.3 combines the benefits of the window-based and utility-function approaches, applying the model to states deep in the search tree, though referencing it only for the number of states that are in the immediate search window. We tested the use of this method in time-limited tournaments (using iterative deepening search), and compared the modeller's performance with the use of the window-based and utility-function usage techniques. The result is shown in Table IV.

Table IV. *Timed tournaments, comparing window-based model-usage, the utility-function method, and the propagated-states method. Confidence intervals are reported for p=0.05.*

	Sec/ Move	Before Modelling	Modelling Window	Modelling Utility	Modelling Propagated
Connect4	0.1	$0.552_{\pm 0.0097}$	$0.583_{\pm 0.0097}$	$0.558_{\pm 0.0097}$	$0.619_{\pm 0.0095}$
	0.2	$0.559_{\pm 0.0097}$	$0.578_{\pm 0.0097}$	$0.605_{\pm 0.0096}$	$0.624_{\pm 0.0095}$
Checkers	0.1	$0.693_{\pm 0.0090}$	$0.763_{\pm 0.0083}$	$0.478_{\pm 0.0098}$	$0.782_{\pm 0.0081}$
	0.2	$0.612_{\pm 0.0095}$	$0.684_{\pm 0.0091}$	$0.473_{\pm 0.0098}$	$0.713_{\pm 0.0088}$

We find that the use of the propagated-states usage method gives higher performance results than either the window-based or the utility-function method, in both domains and with both time limits tested. The result confirms the benefit of the propagated-states method in timed competition, offering a good compromise between the two alternative usage methods.

3.6. Incorporation of a Self Model

In order to test the self-modelling approach described in Section 2.4, the *Basic Experiment* has been repeated with this new strategy concept. All system parameters remain the same, only that the learning phase now includes a series of self-play tournaments, for learning the *self model* in addition to the opponent model. These self-play tournaments are executed under the same time/ depth constraints as those of the opponent.

Table V shows the result of the *Basic Experiment* repeated with the incorporation of a *self-model* in the modelling player's decision procedure, in addition to the opponent model.

Table V. *Only opponent model-usage vs. opponent and self-model usage, using the window-based method. Confidence intervals are reported for p=0.05.*

	Before Modelling	Opponent Model	Opponent and Self Model
Connect-Four	0.556 $\pm$0.0097	0.629$\pm$0.0095	0.657$\pm$0.0093
Checkers	0.432 $\pm$0.0097	0.495 $\pm$0.0098	0.525 $\pm$0.0098

The addition of the *self model* clearly gives the modelling player an additional performance boost, in both domains. This result confirms that the exploitation of our agent's *relative advantage* over its opponent leads to an effective strategy in a competitive interaction.

4. Related Work

The learning and utilization of a weakness model differs from work previously done in the field of *Opponent Modelling* in a number of ways. Most previous techniques focused on a particular model of interaction, such as repeated games (Carmel and Markovitch, 1996c; Carmel and Markovitch, 1998; Carmel and Markovitch, 1999; Mor et al., 1996; Freund et al., 1995), two-player, zero-sum, perfect information games (Reibman and Ballard, 1983; Jansen, 1992; Carmel and Markovitch, 1993; Iida et al., 1993; Sen and Arora, 1997; Donkers et al., 2001), imperfect information games (Billings et al., 1998), and market systems (Vidal and Durfee, 1995; Schapire et al., 2002). Stone et al. (2000) and Bruce et al. (2002) applied Opponent Modelling techniques to a robotic soccer game, which is a dynamic environment with continuous action space and incomplete information.

In principle, our method of learning and utilization of the opponent weakness model is not restricted to any particular model of interaction. We are not even limited to a fully observable environment, provided that the agent has an appropriate strategy. To learn and use the weakness model, however, we require that the states be fully observable from the point of view of the set of features used in learning the weakness model.

Furthermore, our method assumes that opponent weakness is Markovian, though the opponent decision procedure can be either Markovian or non-Markovian. In practice, however, it is unlikely that the opponent

strategy is dependent on the history of interaction, while its weakness is not.

An additional benefit of our proposed approach is that we make minimal assumptions about the opponent's strategy. The only assumption that we do make is that we have access to a set of features that is sufficiently rich to represent its weakness. Previous work in the field has been based on various representation schemes for modelling the opponent strategies. The particular selection of such a scheme can potentially limit the types of opponent strategies that can be handled by the modelling agent. For instance, opponents assumed to be using regular strategies are modelled by Carmel and Markovitch (1996c; 1998) and by Mor et al. (1996). Jansen (1992) and Iida et al. (1993) represent their opponent strategies as utility functions, while Carmel and Markovitch (1996a) and Gao et al. (2001) represent strategies as pairs consisting of the utility function and search depth. Schapire et al. (2002) do not represent the opponent strategy explicitly, but include features indicating the opponents' identities in order to learn their effect on predicted prices.

Several researchers have explored probabilistic models of opponents, using various forms of representation. Opponents are modelled as having a fixed probability of error in (Reibman and Ballard, 1983), whereas opponents are modelled as making decisions according to conditional action probabilities in (Sen and Arora, 1997; Billings et al., 1998). Opponent models are represented as probabilistic automata in (Freund et al., 1995). Some approaches utilize a mixed model of the opponent, representing uncertainty regarding the model by maintaining a probability distribution over possible models (Carmel and Markovitch, 1999; Donkers et al., 2001). Finally, several techniques work with recursive models of the opponent, in which the model also includes the model the opponent holds about the agent, and so on recursively (Carmel and Markovitch, 1996a; Gmytrasiewicz and Durfee, 1995; Vidal and Durfee, 1996).

Many of the above approaches aim to fully represent the opponent strategy, and so they make the restrictive assumption that the opponent strategy can be represented by the specified structure. In our weakness model, we make no assumption about the form of the opponent's strategy, but instead simply aim to recognize its weaknesses relative to a particular domain. Obviously, our approach also limits the potential gain of Opponent Modelling, since we are only learning one aspect of the opponent's strategy.

Several learning paradigms have been used in previous work for inferring the opponent model. All the learning approaches assume access to examples of opponent behavior. Some of the learning approaches

are done offline, and some are applied online, during interaction. Most online learning agents employ an incremental approach, where each additional example leads to a small modification of the model. For example, probabilistic learning approaches have been applied to learn an opponent model via a conditional probabilistic update rule (Sen and Arora, 1997; Billings et al., 1998; Gmytrasiewicz and Kellogg, 1998). An incremental approach is also used by reinforcement learning methods, although it is used for inferring a strategy rather than a model (Sandholm and Crites, 1995; Littman, 1994; Uther and Veloso, 1997).

An alternative method of learning utilizes a batch processing approach, where the learner attempts to construct a model which is consistent with a set of opponent action decisions from past interactions. For example Carmel and Markovitch (1996c) and Mor et al. (1996) attempt to infer a regular model, i.e. a DFA consistent with past examples. An alternative method attempts to infer a function consistent with constraints derived from the particular opponent actions (Carmel and Markovitch, 1996b).

Our learning method, which infers a decision tree consistent with past examples, is an instance of the batch processing approach. The advantage of this approach is that it usually requires less interaction examples, at the expense of more computational resources associated with each learning example. Another advantage that our approach has over reinforcement learning is that it can be applied to state spaces that are too large to enumerate in a policy.

Our work differs from previous work also in the way that it *uses* the opponent model. There are two primary approaches to using the opponent model in existing works. The first is by constructing a *best-response* strategy, that is optimal against the learned model (Carmel and Markovitch, 1998; Carmel and Markovitch, 1999; Mor et al., 1996). The more common approach uses the model to simulate the actual opponent actions, commonly in lookahead search. This simulation can be deterministic (Iida et al., 1993; Iida et al., 1994; Gao et al., 1997; Gao et al., 1999; Stone et al., 2000), probabilistic (Sen and Arora, 1997; Reibman and Ballard, 1983; Billings et al., 1998; Carmel and Markovitch, 1999; Donkers et al., 2001), or recursive (Carmel and Markovitch, 1996b; Carmel and Markovitch, 1996a; Gmytrasiewicz and Durfee, 1995; Vidal and Durfee, 1996).

We use a different approach, (similar to that of Reibman and Ballard (1983)) that utilizes the opponent model only to *bias* the agent's actions. Our approach has the advantage of being much less prone to deterioration of performance as a result of modelling error.

The work done by Bruce et al. (2002) also exploits weaknesses to gain an advantage over opponent agents. In their work, various aspects of

the opponent's strategy are represented with histograms. For instance, the *occupancy histogram* represents positions the opponents occupy on the field. The system designer then analyzes these histograms to infer various weaknesses of the opponents' strategy, and specifies a set of heuristic rules that exploit these weaknesses to gain an advantage. For instance, if the opponent is found to be using static positions, the designer may specify a rule recommending that robots be sent to these locations to disrupt the opponent.

This approach is similar to ours in that it learns and uses opponent weaknesses to bias the agent's strategy. It differs from our method in representation of the opponent model, since the opponent weaknesses are not expressed in an explicit model, as in our approach. Their system has the advantage that it does not define a general *weakness* concept, but rather considers various aspects of weakness. Another difference between our method and that of Bruce et al. is the usage of the weakness model. Our methodology includes a general, domain-independent method of exploiting weaknesses, while theirs requires hand-crafted, domain-specific heuristics. For instance, in the example used above, our agent must explicitly observe states where a static position of the opponent is blocked, thus causing the opponent's suboptimal play. Their agent must only observe that the position is held frequently by an opponent player. Our agent, however, will automatically infer that blocking the position is a good action, (exploiting the weakness), while they must manually compose the rule expressing this strategy.

5. Discussion

This paper studies the learning and utilization of a model of opponent weaknesses to be used by an agent in an adversarial interaction. The strategy developed falls under the general category of *Opponent Modelling*, in which an agent develops a model of its opponent in order to adapt its behavior to this particular adversary in appropriate situations.

Our approach is based on the observation that the opponent agent's behavior is not uniform over all domain states. We thus try to characterize states according the opponent's relative strength in a particular domain, and use this knowledge to our agent's advantage. We have described a method of acquiring a model of these *relative opponent weaknesses*, via a learning process that generalizes over examples of opponent behavior in the specific domain. The process is performed in an offline learning phase, using a teacher with strong evaluation ability to assess the quality of opponent action decisions. A model is

then inferred, and incorporated into the agent's decision procedure by consulting it as to preferred domain states, i.e. states in which the opponent is predicted to be relatively fallible.

In addition, we presented the inclusion of a *self model*, which is a similar weakness model of our own agent. This model is consulted along with the opponent model, so that the agent prefers states in which it has a *relative advantage* over the opponent, i.e., the opponent weakness model predicts the opponent to be weak, and the self weakness model predicts the agent to be strong.

Empirical results were presented for both the opponent weakness model strategy and the additional inclusion of the self model, applying the strategies in actual competition against learned opponents. Experiments were performed for several different parameters, demonstrating the effectiveness of the proposed method under various conditions. We found that the method was able to be successful for several different feature sets, several different agent heuristic functions, and variations of time constraints. We also demonstrated the tradeoffs involved in varying the risk factor for both the window-based usage method and the utility-function usage method.

We discovered that incorporating the model as a term of the agent's utility function is a highly effective method, significantly more so than using the model as a tie-breaking mechanism among a window of "almost-best" actions. However, this proved to be true primarily on the condition that time limit was not a factor. We found that the hybrid strategy method, of applying the model on states propagated from deep in the search tree, was quite effective under time constraints, thus addressing the problem of computational overhead introduced by the utility-function method. Finally, we found that taking into consideration our agent's *relative advantage* over an opponent, by the usage of a self model, offered an additional improvement to the performance of the modelling agent.

In contrast to previous work done in the field of *Opponent Modelling*, we do not aim to accurately learn the opponent's strategy, but instead simply recognize its weaknesses relative to a particular domain. Thus, we do not have to assume that the strategy being learned can be fully described by a DFA (Carmel and Markovitch, 1996c), nor do we make any assumptions regarding the opponent agent's search depth (Iida et al., 1993; Gao et al., 2001).

In addition, the *risk* introduced to the agent's strategy by our method (under certain parameter constraints) is minimal. Unlike algorithms such as M* (Carmel and Markovitch, 1996a) and OM-Search (Iida et al., 1993), we do not replace search steps with the "predicted" opponent actions. Instead, we simply add a bias toward states of in-

ferred opponent weakness, aside from which the agent keeps its original strategy. Thus, the effect of incorporating the weakness model into the strategy can not potentially cause harm. If, indeed, the weakness model is reasonably accurate, then using it is likely to improve the agent's performance. If, however, the learner fails to induce an accurate classifier, whether due to insufficient features, a poor teacher, or otherwise, then the extent of the potential harm of modelling is bounded by the risk factor. For example, when using the safe strategy of using the model in a tie-breaking scheme, even a random model cannot harm the agent's performance.

The weakness-model strategy approaches the *Opponent Modelling* problem from the perspective of concept learning. We have defined *opponent weakness* as a concept, and have developed a system to learn and exploit it. This differs from the probabilistic approaches, which assume the opponent errs with a fixed probability, and thus develop a model based on this. Sen and Arora (1997), for example, present a "Maximum Expected Utility player" which uses a probabilistic model of the opponent, and learns this model via the updating of conditional probabilities. Defining Opponent Modelling as a concept learning problem has the advantage of a large selection of well-studied induction algorithms.

As in all concept learning problems, our approach depends on the availability of informative features that are sufficient for generalizing the weakness concept. When the quality of the selected features is low, the utility of our method may be negative. However, the harm to the agent's performance can be controlled by the risk factor. Furthermore, this control can be automated by incorporating self-monitoring into the learner. The agent can, for example, test its learned model on a set-aside validation set, and dynamically reduce the risk factor if the accuracy of the learned weakness model is found to be low.

Our tagging method assumes that the "goodness" of an action in a given state can be evaluated regardless of the particular opponent agent that executes it. One interesting exception is an opponent that selects sub-optimal actions to avoid states where it estimates itself to be weak (for example, by using self-modelling).

To summarize, in order to apply our methodology to a real-world domain, the following components must be supplied:

1. *A set of examples of the opponent's actions:* The source for such examples can be the history of the current online interaction, or observations of the opponent's past interactions with our agent or other agents. Obviously, to allow for the assessment of the quality of an opponent action, the example must include all the information

that was available to the opponent when it made the decision to take this action. This may include hidden information.

2. *A teacher:* For automatic tagging, we need a procedure that can accurately assess the quality of the opponent's actions. Such a procedure is readily available when our agent uses a contract algorithm, whose quality improves with more resources. For example, lookahead-based search algorithms usually improve the quality of evaluation with increased depth. Alternatively, the examples can be manually tagged by a human expert.

3. *A set of informative features:* In many domains, the set of features used by the agent's strategy are sufficient to characterize states of opponent weakness. However, it may be helpful to supply additional features in complex domains.

4. *An agent strategy:* The minimal requirement for our framework is a strategy that returns a set of almost-best actions. To be able to control the risk, a utility function over actions or states should also be supplied.

The learning and exploitation of an *opponent weakness model* can be expanded in a number of different directions, and further applied to various classes of problems. One interesting direction is the application of our methodology to dynamic strategies, i.e. strategies that change over the course of time. It is quite possible that, while the strategy changes, the weakness of the opponent, i.e. the set of states where its performance is inferior, will be relatively fixed. In such cases, we expect our methodology to retain its effectiveness. In addition, we may test our method against opponents using probabilistic strategies.

One interesting direction for exploration is the modelling of an opponent that uses different strategies against different opponents. It is possible in this case that the opponent *weakness* is still independent of its opponents, i.e. the opponent exhibits weakness in certain states regardless of its particular rival, (and the specific strategy used against it). In such a case, our method will still be effective. If, however, the opponent's weakness is relative to its opponent-tailored strategy, then the learning agent should use for training only interactions between the opponent and itself. If such interactions are not available, then the agent should prefer examples of interaction with agents that have similar strategies to its own. To apply this technique, we need to design a metric for evaluating similarities between strategies.

A related direction for future study is the development of a weakness model of an opponent that also utilizes a weakness model; e.g.

applying the method *recursively*. In the tradition of the *Recursive Modelling Method* (Gmytrasiewicz and Durfee, 1995), we can study the application of the method to any level of recursion.

A further direction for research is the extension of the model to a multi-agent system consisting of several (more than two) competing agents. An agent may build models of several different agents, and incorporate the set of models into its strategy in some way. Alternatively, a multi-agent weakness model may be developed, which combines the weaknesses of several opponents into one classification model, relative to domain states. Such a model may be used to bias the interaction toward states that are predicted to yield maximal utility for the agent, based on the relative shortcomings of other agents in the system.

We believe that this work presents a good foundation for the learning and utilization of opponent weakness models, including the usage of a self model in a *relative advantage* perspective. A framework has been developed that can be used as a basis for further research and experimental work.

References

Allis, V.: 1988, 'A knowledge-based approach of Connect-Four - the game is solved: White wins'. Master's thesis, Department of mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands.

Angluin, D.: 1978, 'On the Complexity of Minimum Inference of Regular Sets'. *Information and Control* **39**, 337–350.

Atkeson, C. and J. Santamaria: 1997, 'A comparison of direct and model-based reinforcement learning'.

Billings, D., D. Papp, J. Schaeffer, and D. Szafron: 1998, 'Opponent Modeling in Poker'. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Madison, Wisconsin, pp. 493–499.

Bruce, J., M. Bowling, B. Browning, and M. Veloso: 2002, 'Multi-Robot Team Response to a Multi-Robot Opponent Team'. In: *Proceedings of IROS-2002 workshop on Collaborative Robots*.

Carmel, D. and S. Markovitch: 1993, 'Learning Models of the Opponent's Strategy in Game-Playing'. In: *Proceedings of The AAAI Fall Symposium on Games: Planning and Learning*. North Carolina.

Carmel, D. and S. Markovitch: 1996a, 'Incorporating Opponent Models into Adversary Search'. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Portland, Oregon, pp. 120–125.

Carmel, D. and S. Markovitch: 1996b, 'Learning and Using Opponent Models in Adversary Search'. Technical Report CIS9609, Technion.

Carmel, D. and S. Markovitch: 1996c, 'Learning Models of Intelligent Agents'. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Portland, Oregon, pp. 62–67.

Carmel, D. and S. Markovitch: 1998, 'Model-based learning of interaction strategies in multi-agent systems'. *Journal of Experimental and Theoretical Artificial Intelligence* **10**(3), 309–332.

Carmel, D. and S. Markovitch: 1999, 'Exploration Strategies for Model-based Learning in Multiagent Systems'. *Autonomous Agents and Multi-agent Systems* **2**(2), 141–172.

Donkers, H., J. Uiterwijk, and H. van den Herik: 2001, 'Probabilistic opponent-model search'. *Information Sciences* **135**(3-4), 123–149.

Freund, Y., M. Kearns, Y. Mansour, D. Ron, and R. Rubinfeld: 1995, 'Efficient algorithms for learning to play repeated games against computationally bounded adversaries'. In: *Proc. of the 36th Annual Symposium on Foundations of Computer Science*. pp. 332–341, IEEE Computer Society Press, Los Alamitos, CA.

Gao, X., H. Iida, J. W. Uiterwijk, and H. J. van den Herik: 2001, 'Strategies anticipating a difference in search depth using opponent-model search'. *Theoretical Computer Science* **252**(1–2), 83–104.

Gao, X., H. Iida, J. W. H. M. Uiterwijk, and H. J. van den Herik: 1997, 'Performance of (D,d)-OM Search in Othello'. In: *Proceedings of JSSST 14th Conference*. Shikawa, Japan, pp. 229–232.

Gao, X., H. Iida, J. W. H. M. Uiterwijk, and H. J. Van den Herik: 1999, 'A Speculative Strategy'. *Lecture Notes in Computer Science* **1558**, 74–92.

Gmytrasiewicz, P. J. and E. H. Durfee: 1995, 'A rigorous, operational formalization of recursive modeling'. In: V. Lesser and L. Gasser (eds.): *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*. San Francisco, CA, USA, AAAI Press.

Gmytrasiewicz, P. J. Noh, S. and T. Kellogg: 1998, 'Bayesian Update of Recursive Agent Models'. *User Modeling and User-Adapted Interaction: An International Journal, Special Issue on Learning for User Modeling* **8**(1-2), 49–69.

Hsu, F.-H., T. Ananthraman, M. Campbell, and A. Nowatzyk: 1990, 'Deep Thought'. In: T. Marsland and J. Schaeffer (eds.): *Computers, Chess and Cognition*. Springer New York, pp. 55–78.

Hu, Y.-J. and D. F. Kibler: 1996, 'Generation of Attributes for Learning Algorithms'. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*. Menlo Park, pp. 806–811, AAAI Press / MIT Press.

Iida, H., J. W. H. M. Uiterwijk, H. J. van den Herik, and I. S. Herschberg: 1993, 'Potential Applications of Opponent-Model Search, Part I: The Domain of Applicability'. *ICCA Journal* **16**(4), 201–208.

Iida, H., J. W. H. M. Uiterwijk, H. J. van den Herik, and I. S. Herschberg: 1994, 'Potential Applications of Opponent-Model Search, Part II: Risks and Strategies'. *ICCA Journal* **17**(1), 10–14.

Jansen, P. J.: 1992, 'Using Knowledge about the Opponent in Game-tree Search'. Ph.D. thesis, Carnegie Mellon University.

Junghanns, A. and J. Schaeffer: 1997, 'Search Versus Knowledge in Game-Playing Programs Revisited'. In: *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*. Nagoya, Japan, pp. 692–697.

Kaelbling, L. P., M. L. Littman, and A. P. Moore: 1996, 'Reinforcement Learning: A Survey'. *Journal of Artificial Intelligence Research* **4**, 237–285.

Littman, M. L.: 1994, 'Markov Games as a Framework for Multi-Agent Reinforcement Learning'. In: *Proceedings of the 11th International Conference on Machine Learning (ML-94)*. New Brunswick, NJ, pp. 157–163, Morgan Kaufmann.

Markovitch, S. and D. Rosenstein: 2001, 'Feature Generation Using General Constructor Functions'. *Machine Learning* **49**, 59–98.

Markovitch, S. and Y. Sella: 1996, 'Learning of Resource Allocation Strategies for Game Playing'. *Computational Intelligence* **12**(1), 88–105.

Matheus, C. J. and L. A. Rendell: 1989, 'Constructive Induction On Decision Trees'. In: N. S. Sridharan (ed.): *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. Detroit, MI, USA, pp. 645–650, Morgan Kaufmann.

Moore, A. W. and C. G. Atkeson: 1993, 'Prioritized Sweeping: Reinforcement Learning With Less Data and Less Time'. *Machine Learning* **13**, 103–130.

Mor, Y., C. Goldman, and J. Rosenschein: 1996, 'Learn your opponent's strategy (in polynomial time)!'. In: G. Weiss and S. Sen (eds.): *Adaptation and Learning in Multi-agent Systems, Lecture Notes in Artificial Intelligence*, Vol. 1042. Springer-Verlag.

Nau, D. S.: 1980, 'Pathology on game trees : A summary of results'. In: *Proceedings of the First National Conference on Artificial Intelligence*. Stanford, California, pp. 102–104.

Nau, D. S.: 1982, 'An investigation of the causes of pathology in games'. *Artificial Intelligence* **19**, 257–278.

Pagallo, G. and D. Haussler: 1990, 'Boolean feature discovery in empirical learning'. *Machine Learning* **5**(1), 71–99.

Pearl, J.: 1983, 'On the nature of pathology in game searching'. *Artificial Intelligence* **20**, 427–453.

Quinlan, J. R.: 1986, 'Induction of Decision Trees'. In: *Machine Learning*, Vol. 1. Morgan Kaufmann, pp. 81–106.

Reibman, A. and B. Ballard: 1983, 'Non-minimax strategies for use against fallible opponents'. In: *Proceedings of the international conference on artificial intelligence AAAI-83*. Los Altos, CA, pp. 338–343, William Kaufman.

Russell, S. and E. Wefald: 1991, *Do the right thing : studies in limited rationality*, Artificial Intelligence. Cambridge, Mass: MIT Press.

Sandholm, T. W. and R. H. Crites: 1995, 'Multiagent reinforcement learning and the iterated Prisoner's Dilemma'. *Biosystems Journal* **37**, 147–166.

Schapire, R., P. Stone, D. McAllester, M. Littman, and J. Csirik: 2002, 'Modeling auction price uncertainty using boosting-based conditional density estimation'. In: *Proceedings of the Nineteenth International Conference on Machine Learning*.

Sen, S. and N. Arora: 1997, 'Learning to take risks'. In: *AAAI-97 Workshop on Multiagent Learning*. pp. 59–64.

Sen, S. and G. Weiss: 1999, 'Learning in Multiagent Systems'. In: G. Weiss (ed.): *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, Massachusetts: The MIT Press, Chapt. 6, pp. 259–298.

Simon, H. A.: 1982, *Models of Bounded Rationality, Volume 1*. Cambridge, Massachusetts: The MIT Press.

Stone, P., P. Riley, and M. Veloso: 2000, 'Defining and Using Ideal Teammate and Opponent Agent Models'. In: *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*. Menlo Park, CA, pp. 1040–1045, AAAI Press.

Sutton, R.: 1990, 'Integrated architectures for learning, planning, and reacting based on approximating dynamic programming'. In: *Proceedings of the Seventh International Conference on Machine Learning*. pp. 216–224.

Uther, W. T. B. and M. M. Veloso: 1997, 'Generalizing Adversarial Reinforcement Learning'. In: *Proceedings of the AAAI Fall Symposium on Model Directed Autonomous Systems*.

Vidal, J. M. and E. H. Durfee: 1995, 'The impact of nested agent models in an information economy'. In: V. Lesser (ed.): *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS'96)*. Kyoto, Japan, The MIT Press: Cambridge, MA, USA.

Vidal, J. M. and E. H. Durfee: 1996, 'Using Recursive Agent Models Effectively'. In: M. Wooldridge, J. P. Müller, and M. Tambe (eds.): *Proceedings on the IJCAI Workshop on Intelligent Agents II : Agent Theories, Architectures, and Languages*, Vol. 1037 of *LNAI*. pp. 171–186, Springer-Verlag: Heidelberg, Germany.

Watkins, C. J.: 1989, 'Learning from delayed rewards'. Ph.D. thesis, University of Cambridge.

Watkins, C. J. and P. Dayan: 1992, 'Q-Learning'. *Machine Learning* **8**, 279–292.

Weiss, G. and S. Sen: 1996, *Adaptation and learning in multi-agent systems, Lectures Notes in Articial Intelligence*, Vol. 1042. Springer-Verlag.

Zilberstein, S.: 1995, 'Optimizing decision quality with contract algorithms'. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Montreal, Canada, pp. 1576–1582.