Distributed Decision Making in Checkers

J. Ignacio Giráldez and Daniel Borrajo

Universidad Carlos III de Madrid
c/ Butarque, 15
28911 Leganés, Madrid, Spain
{giraldez,dborrajo}@ia.uc3m.es

Abstract. The game of checkers can be played by machines running either heuristic search algorithms or complex decision making programs trained using machine learning techniques. The first approach has been used with remarkable success. The latter approach yielded encouraging results in the past, but later results were not so useful, partly because of the limitations of current machine learning algorithms. The focus of this work is the study of techniques for distributed decision making and learning by Multi-Agent DEcision Systems (MADES), by means of their application to the development of a checkers playing program. In this paper, we propose a new architecture for knowledge based systems dedicated to checkers playing. Our aim is to show how the combination of several known models for checkers playing can be integrated into a MADES, that learns how to combine individual decisions, so that the MADES plays better than any of them, without "a priori" knowledge of the quality or area of expertise of each model. In our MADES, we integrate well known search algorithms along standard machine learning algorithms. We present results that clearly show that the team as a single entity plays better than any of its components working in isolation.

1 Introduction

Computer programs for checkers play have been traditionally built using quite different approaches and paradigms. Heuristic search combined with database lookups has yielded impressive results [13], while machine learning algorithms have fared poorly. As it is stated in [13], some methods like genetic algorithms, neural nets and function optimization have been tried for the task of learning to classify checkers situations (as either win, loss, or draw for a given color), but were discarded because of unacceptably high error rates. The authors of the present paper have experimented with several machine learning paradigms (ID3 [7], C4.5 [8], bayesian learning [14], and backpropagation [10]) and have obtained similar results. In section 2 we discuss this issue and report on our own experience.

On the one hand, heuristic search combined with database lookup requires the use of huge resources and takes a limited advantage of available knowledge, but performs satisfactorily. On the other hand, machine learning programs should provide a satisfactory solution to a problem that is full of learning opportunities, but they fail to do so in the general setting given the huge hypothesis spaces. Since we believe that you can take advantage of both approaches, we propose to integrate them in a distributed checkers playing system. With this aim, we built autonomous decision making and learning systems for playing checkers, based on heuristic search and different machine learning paradigms. Each one of these systems is built as an autonomous agent using a single paradigm, and is able to play checkers. We have organized them as a Multi Agent DEcision System (MADES) [4].

The MADES decides which move to make on the checkers board by means of a distributed decision making procedure as explained in section 3. The idea of building a distributed decision making and learning system to play checkers is based on the following belief: given appropriate conditions, a group of agents forming a MADES is expected to play better than any of them playing in isolation. The aforementioned conditions were discussed at length in [3] by the authors, in a general context. The purpose of integrating individual monolithic systems into a MADES is to obtain a team performance unattainable when the individual systems work in isolation. We believe this to be an issue of paramount importance since it provides a performance enhancement mechanism (again, only when certain conditions are met).

A similar approach was used by Epstein [2]. In that case, she built a set of game-independent *Advisors*, some of them could also learn using different learning techniques. Her system had a meta-theory on how to play independent of the actual game that was been played. In our case, we differentiate between the agents that propose a decision (move in the case of game playing) and the advisors that decide on which agent is more appropriate for that decision. This allows us to learn two different concepts: how to make a decision, and how to give credit to someone making a decision. Also, her meta-theory depends very much on the game playing paradigm, while our architecture is domain-independent.[1] Finally, her advisors did not collaborate for making decisions, while our agents are allowed to ask for advice to the rest.

In section 4 we explain the experiments we have carried out and give the results obtained. We evaluate these results and draw some conclusions in section 5.

2 Machine Learning and the Game of Checkers

The game of checkers, like most games, is full of learning opportunities for machine learning systems. The pioneering work of Arthur Samuel [11], demonstrated the use of two learning mechanisms which noticeably improved the behaviour of his checkers program. The learning mechanisms he used in his program were very primitive, compared to the range of machine learning formalisms available nowadays. Nonetheless, these formalisms have not yet provided a satisfactory

[1] We are currently applying it to a hard induction problem, with very encouraging results.

machine learning solution to checkers. The focus of Samuel's work was the study of machine learning techniques in the context of checkers, so he resisted the temptation of hardwiring expert knowledge into his program, because he insisted in letting the program discover that knowledge by itself.

Supervised machine learning paradigms can be used to build game playing programs [6]. These learning systems use past play experience, and create a summarised representation of it, that forms the basis for decision making systems, that can make the decision of which move to make. Past play experience is contained in a training set, usually as a series of board descriptions, each followed by the final outcome (class). Supervised machine learning paradigms try to find common patterns in the boards that belong to the same class, and the collection of patterns encountered is used to build a class membership criterium that is used to classify (possibly) unseen checkers situations.

The authors have built for their experiments four different supervised learning systems, based on the following paradigms: ID3, C4.5, bayesian learning and backpropagation. The training set used was a subset of 29000 randomly chosen elements from the DB5 database [12]. The target concepts were `win`, `draw` and `loss` for white (white to move in all the situations). For selecting which move to make next, the successors of the current situation are presented to these programs. The program classifies every successor and gives it a score; the successor that scores highest is the preferred one, and indicates which move to make next. The situations of DB5 are endgames of 5 pieces at most, so the four systems were trained with endgames of 5 pieces or less.[2] Nonetheless, since we expected to obtain a powerful generalization as a result of the inductive nature of the algorithms involved, we tested the four systems with endgames of 8 pieces or less.

The four programs showed a very selective performance: a given program of the four may play certain endgames very well, but it plays others poorly (the set of checkers problems that an agent plays satisfactorily is known as its *competence region*). Moreover, the endgames that were played well by one program did not coincide with those played well by another[3] except for, as one would expect, in the case of ID3 and C4.5, given that C4.5 uses the same basic techniques as ID3. Since C4.5 handled well most of the situations that ID3 handled well, plus many others, we stopped using ID3 and used C4.5 instead.

With the aim of improving the overall performance of the programs, a second training set with $100,000$ situations, randomly taken from DB5, was built. The four systems were trained with this new training set. Surprisingly, the backpropagation system performed worse with this second training set. We modified the topology of the neural network with the purpose of making more expressive power available for internal representation, but none of the enlarged nets per-

[2] Examples were randomly selected from the database. In case the database examples have any kind of bias towards a specific type of position, this does not affect to our goal; our aim is not to build the best machine learning system out of that data, but to learn how to better combine it with other systems.

[3] The reasons of this behaviour are explained in [3].

formed any better than the one trained with the first training set; the neural network did not scale up well. On the other hand, the Bayes and C4.5 systems improved remarkably, but still showed the same highly selective behaviour.

None of the four systems that were tested performed satisfactorily. We believe that this is because of the effect of the representation formalism used to represent the checkers situations in the training set. The use of an inadequate representation formalism can cause an unacceptable error rate. Sometimes this is due to the difficulty of expressing the target concept in terms of adequate input attributes or combinations of them. The use of meaningful intermediate concepts, with a more direct relation to the target concept, can alleviate this problem. We will illustrate this point with an example. Suppose that having one more king is to some extent determinant in some situations. Using the raw board description, the machine learning program will only notice that having some men in certain locations leads to some advantage (because of the existence of a crowning chance). To identify the boards that lead to that advantage, raw features will have to be combined, and many such combinations will have to be remembered, and in some way related to that advantage. The feature combinations that are thus grouped will be quite dissimilar. Now, let us imagine a higher level description for the checkers situations, using intermediate concepts (e.g. crowning chance in next move, crowning chance in n moves, capture chance in n moves, dominance of the center, victory chance in n moves, and so on), besides a raw description. The prior series of combinations of raw features is expressed now more easily, because more descriptive features are being used. This means that we are making the work easier for the machine learning program, because the common pattern is now expressed in a simpler way (that involves less features of the checkers situation description, combined more simply). This could be achieved by careful hand writing of the input features, or by use of automatic methods, such as *constructive induction* [9].

Other learning approaches applied to game playing have ranged from chunking in chess [1], temporal differences in backgammon [15], or bayesian learning of evaluation functions in Othello [5]. A similar multi-agent approach applied to game playing was followed by Wiering [17] by learning game evaluation functions using hierarchical neural networks architectures. In his case, all the agents implement the same paradigm (they are all neural networks). All the expert networks used by Wiering are equally suitable for being specialized to deal with any subset of the domain, as opposed to MADES hybrid approaches like ours, where some agents are more likely to correctly classify some situations, given that the paradigm they implement is better suited for that task.

3 Multi Agent Decision Systems for Checkers Play

In this section we describe how the overall architecture works as a problem solving (decision making) and learning model.

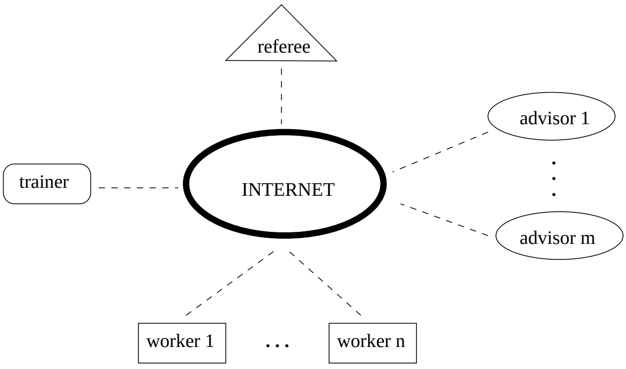

Fig. 1. The Intelligent Agents Organisation.

3.1 The Composition of a MADES

The Intelligent Agents Organization is a model composed by multiple intelligent heterogeneous agents that cooperate to attain a common overall goal. The IAO structure, is shown in Figure 1.

- One agent, known as the **referee**, is in charge of the overall system control. It broadcasts problem instance descriptions (in our application they are checkers situations), and control signals to the rest of the team. It then receives the respective replies from the rest of the agents. These replies may be either advice, or problem solving proposals (move proposals in our application). The services the referee may request to an agent are: solution proposal synthesis (only to worker agents), execution of a learning session (if the agent has learning capabilities), and advice request (only to selected agents). These service requests are scheduled in a way that maximizes parallellism (every agent runs on a different machine), so the MADES response time is minimized.
- The **worker agents** receive problem descriptions (checkers situations) from either the referee, or another worker, and reply with solution proposals. They work in parallel on a solution proposal to the same problem instance, are capable of autonomous decision making, and, some of them, have learning capabilities. Any of them could be the basis of a monolithic system aimed to solve each problem. The MADES should learn how to organize these worker agents to obtain a joint performance superior to the one that would be obtained in case we built a monolithic system with just one of the worker agents. The learning mechanism that accomplishes this task is distributed reinforcement learning of workers competencies [4].
- Several agents may play the role of **advisors**. They are contacted by other agents that wish to know who is the worker that is expected to handle best a given problem instance. The advisor replies with the identification of the worker that is expected to solve best the problem instance. The advice is used

by the referee as an aid for conflict resolution,[4] and it is also used by workers who wish to know which worker is the most appropriate for collaborating in the solution of a problem instance.
– A **trainer** agent produces problem instances that are used for training and testing. The criteria for problem synthesis affects the success of the learning effort. We are currently working on procedures to determine how to produce problems that speedup learning, and to force the learning of knowledge to handle the worst solved problem instances.

3.2 IAO Decision Making

When a problem instance arrives at the referee, it consults the advisors to determine whether any worker agent is expected to solve that instance of the problem satisfactorily. In that case, the proposal that this agent provides will be given a privileged status when it has to compete with the proposals of its fellow workers. The problem instance description is broadcast to the workers, so that they can work on it, and reply with a solution proposal. One advantage IAO presents, is that the advisors and the workers work in parallel, so the IAO response time is a very small overhead longer than the one that would be obtained from a monolithic system built from the most time consuming IAO worker.

When the referee receives the proposals of all the worker agents, and the advice from its advisors, it has to decide which proposal to use (most of the times this proposals will be incompatible and contradictory). The referee uses a poll mechanism for conflict resolution: the proposal that gets the greatest support is the one the referee will follow. The advisors' candidates receive extra votes in this poll, so they have some advantage over less credited workers.

One of the problems we perceived in previous experiments was that when the number of classifier workers was greater than the number of searcher workers, the system biased towards decisions made by classifiers, producing undesired results. So, we defined an automatic weighting mechanism that equals the maximum number of votes attainable by classifiers, and the maximum number of votes attainable by searchers.

3.3 Learning in IAO

Two different kinds of learning take place in IAO. First, the autonomous worker agents with learning capabilities can learn on their own about how to do their respective work. This is usually called *centralized learning* [16]. And, second, the advisors learn the workers competencies. This is a form of *distributed learning.* Centralized learning deals with knowledge about solutions, that will permit a worker agent to solve the problems it is presented, so it can be carried out locally by the agent, isolated from the rest of the team. Conversely, distributed learning of agents competencies requires the use of global information, because it is based

[4] For instance, when the agents disagree about which move should be made, the referee uses this advice to decide about which alternative to take.

on distributed credit assignment, that analyzes the performance of the MADES as a whole and of the workers individually, with the goal to learn competencies. This kind of learning will be used to make the synergetic effect possible.

The centralized learning algorithms are the same ones used in monolithic systems. Distributed learning is actually what will allow the team of agents to perform better than the individuals on their own. In this process, the advisors analyze how satisfactory the solution the MAS produced is.

We have designed an algorithm to learn workers competencies under the following hypothesis: if a worker is the most competent in the solution of a certain problem, it is also expected to be the most competent in the solution of another problem of the same difficulty and *appearance*. If the problem space is partitioned in subsets that contain similar problem instances, the competence data known for a certain problem instance, is expected to be also valid for the rest of the problem instances lying in the same subset of the partition. This is a generalization mechanism whose success depends on the similarity measure used.

How many subsets are used, and what the similarity metric is, depend on the kind of problem. The goal of this partition is to enclose, in a single subset, problem instances for which any worker agent deserves the same credit. The intended learning will be as reliable as the degree of fulfillment to this requirement. A reinforcement table is associated to every subset. In such a table, a reinforcement is associated to every worker agent, whose meaning is how adequate is the worker agent for the solution of problem instances lying in the subset. This tables are used by a reinforcement advisor agent: once a problem instance arrives to it, it locates the subset of the partition the problem belongs to, and the associated reinforcement table. Then, it determines who is the most adequate worker according to the table. If such a worker exists, the advisor replies to the referee with the worker's identification. Otherwise, it informs the referee about the lack of discerning data. As a result of "a posteriori" problem solving episode analysis, the participation of workers in the solution is determined, and they are consequently reinforced. The analysis and the reinforcement learning effort are carried out by the reinforcement advisor (the referee has been collecting and preparing data for this process during the problem solving episode).

4 Implementation and Experiments

To evaluate the IAO model, we have built a MADES composed of 8 agents (see Figure 2):

- The **referee** agent.
- An advisor agent, known as **reinfAG**, that builds and consults the appropriate reinforcement table, in order to advise other agents about the agent that is expected to solve most satisfactorily a given problem instance.
- The **C4.5AG** agent, based on Quinlan's C4.5 running in C trained on 100,000 instances randomly taken from the Schaeffer's DB5 database of checkers endgames [12].

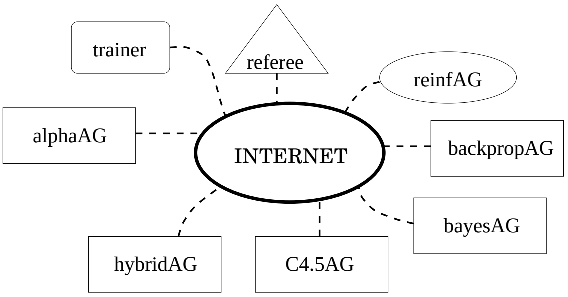

Fig. 2. Architecture of the checkers Multi-Agent System.

- **backpropAG**, a connectionist worker agent. We have built a neural network that learns by means of the backpropagation with momentum learning algorithm [10]. It has been trained with 29,000 examples taken from DB5.
- **bayesAG**, a bayesian classifier trained on the same 100,000 instances used by C4.5AG, running in C.
- **alphaAG**, an alpha-beta based worker agent with decision making capabilities only. Search has been constrained, so that the maximum search depth is limited to 5, and the maximum number of moves that the move generator outputs is 12. The purpose of this severe search constraint is to impose a time limit per move, brief enough to make possible the execution of many experiments,[5] and long enough to yield interesting play. Again, the main goal of this research (for now) is to learn how to combine several agents (strong or not), but not to build the best ever player.

  The knowledge this agent uses is hardwired into its evaluation function and into its move generator (the moves believed to be most interesting are generated first). A simple evaluation function has been used, so that the time it takes to compute it allows to compute it many times. The computation of the evaluation function evaluates the material difference between both sides, weighing the pieces with an amount that reflects the importance of the board area it dominates. This naive evaluation function provides reasonably good play in most common situations.
- **hybridAG**, a heuristic searcher, based on alpha-beta. When a search tree leaf is reached, this worker asks reinfAG who is the worker that is expected to handle best this leaf situation. In case that worker is available, hybridAG requests from it the evaluation of the leaf node. If that worker is not available, hybridAG performs the evaluation of the leaf node locally. Notice that this

[5] Currently the average speed of problems played is around 100 per day, when the MADES runs using 3 Linux PCs, 1 SPI 4MP, and 2 HP Apollo workstations, shared with other users and applications. The availability of more powerful hardware would make it possible to reach selective search depths in accordance to competition programs.

is a loosely coupled hybrid system, and that the searcher will be coupled with different classifiers at different moments.

– A **trainer** agent that produces problem instances that are used for training and testing. These problems are produced in a balanced fashion: there are as many situations that are wins (or losses) for white as there are for black. This has been accomplished by: first, a checkers situation is randomly generated according to some restrictions (e.g. the total amount of pieces must be equal or less than a given constant); then, its inverted form is computed. If the first situation was a win for black, the next situation produced will be its inversion, i.e. a win for white. So, none of the sides is favored by the trainer.

The agents communicate using the TCP/IP protocol over the Internet. Since the computers reside in the same network segment, the communication process takes much less time than the time local servicing of requests takes. In case computers in very distant locations were used, the network slowness in the heavy traffic hours would need to be considered.

For training the MADES, 16,000 checkers problems were generated by the trainer agent. The checkers problems were played until either one side wins, or a draw is reached. The draw criterium we used is to test when a series of moves was being cyclicly repeated.

The MADES played these problems against alphaAG, and a learning session was executed after every problem was played. Since our aim is to prove that the MADES can learn to make decisions in such a way that it beats any of its worker agents playing on its own, the test games were played between the MADES and each of its workers in turn. A set of 100 test problems was produced by the trainer, and this set was used in all the matches. The following results were obtained:

opponent	MADES advantage
c4.5AG	42%
backpropAG	35%
bayesAG	34%
hybridAG	4%
alphaAG	4%

The MADES advantage is computed as the difference between the number of games won by the MADES, minus the number of games won by its opponent, divided by the number of games played, and multiplied by 100 to obtain a percentage. This equals to expressing the game equity as a percentage. We express this calculation mathematically in the following formula:

$$adv(G) = \frac{[w(G) - l(G)] \times 100}{|G|}$$

where G is the set of games played, $|G|$ is the number of played games, $w(G)$ is the number of games won by the MADES, and $l(G)$ is the number of games lost by the MADES. Since we played 100 games per match, the formula above is simplified in this case to the evaluation of the difference between the games won by the MADES and the games won by its opponent. Further experiments with matches consisting of 24 and 50 games, closely approximated the results shown here.

The results show that the MADES beats any of its members. We believe that this gain in the quality of play justifies by itself the construction of the MADES from the standalone systems. We are currently in the MADES training stage, where, after every training game, reinforcement tables are updated. Since the reinforcement tables have not yet converged, we expect that the results will improve as the tables get near their convergence values.

The flexibility of the IAO model allows the replacement of a worker by another, and the adaption of the rest of the system to the new MADES composition, thanks to the adaptive behaviour of the advisor. We replaced two workers during the MADES's lifetime; the first replacement improved the MADES' score in 6.8%, and the second in 4.68%.

5 Discussion

What the results show might seem obvious to anyone: an isolated system should be beaten (or drawn) by a team formed by itself plus other agents. What is not obvious at all is to determine under which conditions this is feasible, and to learn to control the system in a way that assures inter-agent cooperation and prevents inter-agent hampering. It should be noted that there is no "a priori" clue about how the individual systems should be combined, and that the MADES learns how to perform this combination on its own. This is the main contribution of the present work.

Initially, the competence and preference regions of the worker agents are unknown. They are learnt by the system, and this information is used to influence the way the MADES makes decisions. This is quite dissimilar to putting together a program with good openings, a program with good middlegame play and a program with good endgame play. Because, in the latter case, one has "a priori" knowledge of the individual systems capabilities, while, in our model, the individual capabilities are not known "a priori", so the MADES has to learn them. This is very common in machine learning: when one builds a decision system based on the output of a machine learning paradigm, one usually cannot foresee which instances will be either correctly or incorrectly classified (i.e. one lacks "a priori" knowledge about the competence region of the system, except for some very coarse grain guesses based on the composition of the training set). So if computer checkers is to take advantage of decision systems based on multiple autonomous learning (or not) agents, a method for learning dynamic competencies like the one we propose here is needed, because there will not be available "a priori" knowledge in the general case for determining which agents should be heeded at a given time.[6]

[6] There will be no informed nor reliable way to combine individual decisions to reach a common overall decision.

The focus of this work is the study of techniques for distributed decision making and learning in MADES, and their application to the construction of a checkers playing program. We are aware that the checkers system presented here does not take advantage of the latest heuristic search enhancements, as championship level programs do. Moreover, we restricted ourselves to checkers problems of 8 pieces at most in our experiments.[7] The assesment of the results reported here provides an experimental background to support the adequacy of the theoretical IAO model. Now that we have tested the adequacy of the techniques involved, we wish to start working on producing a championship level program, enlarging our scope to the whole game of checkers, and refining and improving the existing agents.

The results improve as learning progresses due to two effects. On the one hand, the more extensively the reinforcement tables cover the domain, the more information that is available for **reinfAG**. We are in this first stage currently, and the domain is not completely covered yet. This means that **reinfAG** can not give advice in some situations because there is no reinforcement table corresponding to the subsets those situations belong to. For the MADES to be mature, the whole domain must be covered, and the reinforcement tables must get near its convergence values.

On the other hand, the convergence of the reinforcement tables is a second learning stage that will primarily take place after the covering is done. We believe the MADES has not even passed the covering stage, because new reinforcement tables are often created during learning sessions. We expect the results to improve when the problem domain be totally covered with the union of the competence regions of the worker agents, and the reinforcement tables have approached their convergence values.

6 Acknowledgements

We would like to thank Jonathan Schaeffer for making the DB5 and DB6 databases available, as well as for his assistance with the code for its access.

References

[1] Hans Berliner and Murray Campbell. Using chunking to solve chess pawn endgames. *Artificial Intelligence Journal*, 23:97–120, 1984.

[2] Susan L. Epstein. *Heuristic Programming in Artificial Intelligence*, chapter The Intelligent Novice. Learning to Play Better, pages 273–284. Ellis Horwood, 1989.

[3] Jos I. Girldez and Daniel Borrajo. The intelligent agents organization. In Eugnio Oliveira and Nick Jennings, editors, *Proceedings of the Workshop on Multi-Agent Systems: Theory and Applications (MASTA'97)*, pages 43–56, Coimbra, Portugal, October 1997.

[7] Due to an initial lack of computing power, and to our desire to avoid that the intricacies of the problem domain would shift our attention from our primary goal.

[4] Jos I. Girldez and Daniel Borrajo. Distributed reinforcement learning in multi-agent decision systems. In Helder Coelho, editor, *Progress in Artificial Intelligence, Iberamia 98*, number LNAI 1484 in Lecture Notes in Artificial Intelligence, pages 148–159, Lisboa, Portugal, October 1998. Springer-Verlag.

[5] Kai-Fu Lee and Sanjoy Mahajan. A pattern classification approach to evaluation function learning. *Artificial Intelligence Journal*, 36:1–25, 1988.

[6] J. Ross Quinlan. Learning efficient classifiation procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach, Volume I.* Morgan Kaufman, 1983.

[7] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[8] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[9] L. De Raedt and M. Bruynooghe. Interactive concept-learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150, March 1992.

[10] D.E. Rummelhart, J.L. McClelland, and the PDP Research Group. *Parallel Distributed Processing Foundations.* The MIT Press, Cambridge, MA, 1986.

[11] Arthur Samuel. Some studies in machine learning using the game of checkers. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought.* McGraw-Hill, New York, NY, 1963.

[12] Jonathan Schaeffer. DB5: Checkers endgames database. `ftp cs.ualberta.ca:/pub/chinook/DB5/`, 1994.

[13] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. Chinook, the world man-machine checkers champion. *AI Magazine*, 17(1):21–29, Spring 1996.

[14] J.Q. Smith. *Decision Analysis, a Bayesian Approach.* Chapman & Hall, 1988.

[15] Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4):257–277, May 1992.

[16] G. Weiss. Some studies in distributed machine learning and organizational design. Technical Report FKI-189-94, Institut fur Informatik, Technische Universität München, 1994.

[17] M. A. Wiering. TD learning of game evaluation functions with hierarchical neural architectures. Master's thesis, University of Amsterdam, 1995.