Chapter 1

Introduction

In this thesis, research on the balance between memory and search is presented. As is well known, the trade-off between knowledge and search plays a key role in many domains, such as expert systems, theorem proving, and games. We explicitly note that our research has a focus which differs from the research on the trade-off between knowledge and search, where dealing with knowledge and knowledge representations in order to arrive at intelligent solutions is stressed. In our research we look at just one characteristic of knowledge, viz. the storage of knowledge. More specifically, the research presented in the thesis concentrates on *memory* versus search in games. The domain used is that of two-player zero-sum games, and in particular the games of chess and domineering.

1.1 Games

For as long as computers have existed, people have tried to let them play intelligent (non-trivial) games. The challenge of a computer playing an intelligent game is a classic problem within the field of Artificial Intelligence (AI). Two pioneers seriously considered a computer playing chess, when AI was still in its infancy. Shannon (1950) published a seminal research article in which he described mechanisms to be used in a program playing chess. Turing (1953) was the first to describe a chess-playing program. The first program that could play a reasonable game in a related domain was a checkers program (Samuel, 1959; Samuel, 1967). This program was able to learn from its mistakes, thereby improving its performance.

Since then, much research has been conducted in the domain of games, resulting in very strong programs for many intelligent games. Nowadays, the best checkers program is CHINOOK, written by Martin Bryant, Rob Lake, Paul Lu, Jonathan Schaeffer and Norman Treloar (Schaeffer *et al.*, 1992; Schaeffer, 1997). In 1992 it played the "Man versus Machine World Checkers Championship" against Dr. Marion Tinsley, who is considered the greatest human checkers player in the history of the game (after he became World Champion in 1955 he lost only five serious games out of some thousands (Schaeffer, 1996a)). It was the first time in history that a machine challenged a human for an official title in a non-trivial game of skill. Tinsley won the match (4 wins, 33 draws, and 2 losses). In 1994 a rematch between Tinsley and CHINOOK was played. Tinsley withdrew due to illness after six games (all draws) and he passed the title to CHINOOK. Don Lafferty, the 1994 United States checkers Champion, then challenged CHINOOK and they tied the match (1 win, 18 draws, and 1 loss). Tinsley never recovered from his illness and died in 1995. The subsequent Man versus Machine World Championship against Don Lafferty (January 1995) ended in a victory for CHINOOK (1 win and 31 draws). Since then the gap between man and machine has widened considerably. Therefore, CHINOOK is considered to be the strongest checkers-player in the world.

The best chess program (or better: chess machine or chess computer) is DEEP BLUE, created by Jerry Brody, Murray Campbell, A. Joseph Hoane Jr., Feng-hsiung Hsu and Chung-Jen Tan (Hsu *et al.*, 1990). In February 1996 it played a six-game match under normal tournament conditions against the human chess World Champion of the PCA, Garry Kasparov. Kasparov won the match (3 wins, 2 draws, and 1 loss). However, DEEP BLUE won the first game, surprising many experts (Uiterwijk, 1996; Newborn, 1997). In May 1997 it played a second six-game match against Kasparov. This time DEEP BLUE was able to win the match (2 wins, 3 draws, and 1 loss) (Schaeffer and Plaat, 1997; Goodman and Keene, 1997; King, 1997). This was a milestone in AI history, finally realizing Shannon's dream of 50 years before.

One of today's strongest Othello programs is LOGISTELLO, written by Michael Buro (1994). In the 22 international Othello tournaments it has played so far, it ended first sixteen times, and second five times. In August 1997 it played a six-game match against the Othello World Champion Takeshi Murakami, and won the match with the perfect score of 6–0. This clearly shows that the best Othello programs have become stronger than any human player (Buro, 1997).

We offer four arguments why researchers are so interested in intelligent games.

First, games provide an exact, closed domain (the rules are well-defined), in contrast to real-world problems, which are often rather vague (Van den Herik, 1983). Often many pages are required to give a proper description of (the background of) a real-world problem. Some real-world problems (e.g., in law and legal knowledge-based systems) can be interpreted differently by different people (Van den Herik, 1991). In contrast, games can be defined with sufficient precision.

Second, intelligent games are not trivial. Although it takes only an hour or so to learn the rules of chess, so far it has been impossible, even for the best human, to play chess perfectly. Playing intelligent games is hard and the obstacles that have to be tackled reflect the complexities inherent in real-world problems. Minsky (1968) stated "It is not that the games and the mathematical problems are chosen because they are clear and simple; rather it is that *they give us, for the smallest initial structures, the greatest complexity*, so that one can engage some really formidable situations after a relatively minimal diversion into programming."

Third, the domain of games is well-suited for testing new ideas in problem solving. Therefore, Michie (1980) proposed computer chess as the *drosophila melanogaster* (fruit fly) of machine intelligence. According to Fraenkel (1996) ideas from the domain of games have been used in mathematics, computer science and economics. Nilsson (1971) mentions several applications, including operations research (traveling-salesman problem) and chromosome matching.

Fourth, by creating a machine which plays an intelligent game, it may be possible to gain more insight into the way people reason. The Dutch psychologist De Groot (1946) has investigated the thinking process of chess-players during a game. The American psychologists Newell and Simon (1972) tried to build models of the human mind, based on the results of their research on computer chess (Newell *et al.*, 1958). Recently, a follow-up to De Groot's (1946) book has been published (De Groot and Gobet, 1996) concentrating on perception and memory of chess players.

All four arguments provide a legitimate reason to perform research on games. Developing computer programs or a new computer technique may help to model a domain adequately or may remove an obstacle. Our research is inspired by the arguments two and three. We have developed two new techniques (for transposition tables and for proof-number search) and solved two open problems (domineering and the graph-history-interaction problem).

1.2 Knowledge versus search

One of the best known trade-offs in Computer Science is the trade-off between space and time. In the domain of AI, especially game playing, this comes down to the trade-off between knowledge and search (Clarke, 1977). In theory, all problems with a finite state space are solvable (Allis *et al.*, 1991) in two distinct ways.

1. Solve the problem by knowledge, not using any search. This is possible if all information for the initial state and the subsequent states necessary for solving the problem is available. Moreover, there is sufficient *space* to store the information, and there is a way of discovering and representing the knowledge necessary for solving the problem. For instance, the game of nim can be solved by knowledge alone (Bouton, 1901).

2. Solve the problem by search, not using any knowledge. This is possible if there is sufficient *time* available to do a sufficiently large search of the state space. For instance, the game of tic-tac-toe can be solved by brute force alone (Berlekamp *et al.*, 1982a).

Most problems cannot be solved in practice by using knowledge only or search only. If the state space is too large to be searched in a reasonable time, knowledge is needed to guide the search and to reduce the state space. If the state space is too large to be stored in memory, search is needed to compensate for the loss of knowledge. Thus, for solving the majority of problems a combination of knowledge and search is needed. Some examples of games which have been solved by a combination of knowledge and search are qubic (Patashnik, 1980; Allis and Schoo, 1992), connectfour (Allis, 1988; Uiterwijk *et al.*, 1989; Allen, 1989), go-moku (Allis *et al.*, 1993; Allis, 1994; Allis *et al.*, 1996) and nine men's morris (Gasser, 1995).

When opting for search to solve a problem, we basically distinguish two search categories.

1. *Full-width* search (henceforth called brute-force search uses minimal knowledge to guide the search. After expanding a node in the search, knowledge is used to sort the children. The choice of the node to expand next comes from a (small) subset of the nodes in the tree. For instance, in breadth-first search the next node to be expanded is one of the siblings of the last expanded node, and in depth-first search the next node to be expanded is one of the children of the last expanded node.

2. *Best-first* search uses more knowledge to guide the search. The knowledge is used for the choice on which node to expand next. The node selected can be any leaf in the tree.

Two basic types of knowledge interact with search (Berliner, 1984).

1. *Directing* knowledge. In brute-force, search directing knowledge affects the order in which the descendants of a node are examined. In best-first search, directing knowledge guides the search (i.e., selects which node to expand next).

2. *Terminal* knowledge. Terminal knowledge is applied to the leaves of the search tree. It produces either an exact value (win, draw, or loss), in case the leaf is a terminal node, or a measure of the goodness of the position the leaf represents. Terminal knowledge is used both in brute-force search and in best-first search[1].

In brute-force search many leaves will be evaluated during a search process. Terminal knowledge is applied to all leaves (millions of times during a single search process). Thus, each term in the terminal-knowledge function (the evaluation function) contributes to the cost of an evaluation, which affects the speed of the search process, and should be carefully weighed.

1.3 Memory versus search

The trade-off between knowledge and search mostly deals with knowledge and knowledge representations. We only look at one characteristic of knowledge, viz. the storage of knowledge in *memory.* The purpose of storing knowledge acquired during the search process is to re-use it at a later time. Here we introduce the trade-off between memory and search, by giving two points of view.

As a first observation we note that the size of computer memory is no longer an obstacle, making it easier to equip a computer with more memory. Now the question is: can we make use of the large amount of memory, by storing more knowledge into

---

[1] In best-first search terminal knowledge is closely related to directing knowledge, and may be identical (Berliner, 1984).

this memory, thereby decreasing the need for searching? Depth-first search needs little memory. Only the path from the root to the position under investigation needs to be stored in memory. To use the remaining memory, knowledge about positions encountered in the search process may be stored in a large table, the so-called *transposition table*. The knowledge stored in the table is used to relieve the search. Thus, searching is reduced at the cost of using more memory.

As a second observation we mention the noteworthy increase in computer speed. Can we make use of this increase by using speed to accelerate the search, thereby acquiring more knowledge, decreasing the need for memory? Most best-first search algorithms need a large amount of memory to store the entire search tree. The quality of a best-first search algorithm depends on the quality of the directing knowledge. The speed of a computer increases faster than the amount of internal memory in a computer. Thompson (1996b) states that from 1985 to 1996, the speed of a typical high-quality workstation has increased by a factor of 150 (from 1 MIP to 150 MIPs), whereas its internal memory only has increased by a factor of 16 (from 16 MB to 256 MB). At current computer speeds, memory is quickly filled. Therefore, ways have to be found to use the increase in speed to acquire more knowledge per node, improving the directing knowledge. Then, the search process will search the state space more efficiently, reducing the need for memory at the cost of more searching.

1.4 Problem statements

Three problem statements are considered. The first problem statement addresses decreasing the need for search by increasing the use of memory.

Problem statement 1: Which methods exist to improve the efficiency of a transposition table?

A transposition table is normally used in combination with a depth-first-search algorithm. The most commonly used depth-first algorithm for two-person games is the $\alpha\beta$ algorithm. In the thesis we present research on improving the efficiency of a transposition table used in the $\alpha\beta$ algorithm. The research is performed in two domains: chess and domineering.

The second problem statement addresses decreasing the need for memory by increasing the use of search.

Problem statement 2: Which methods exist for best-first search to reduce the need for memory by increasing the search, thereby gaining more knowledge per node?

In the thesis we present research on a relatively new best-first-search algorithm, proof-number search (pn search). Like many best-first search algorithms, pn search stores the complete search tree in memory. An attempt is made to reduce the need for memory for pn search, realized in a pn-search variant, called *pn*^{2} *search*.

Summarizing, in the first problem statement the need for search is reduced by increasing the use of memory. Analogously, in the second problem statement the need for memory is reduced by increasing the use of search. An attempt to combine the advantages of both approaches (reducing the need for search *and* reducing the need for memory) is the following. In a search tree, it may happen that identical nodes are encountered at different places. If these so-called *transpositions* are not recognized, the search algorithm unnecessarily expands identical subtrees. Therefore, it is profitable to recognize transpositions and to ensure that for each set of identical nodes, only one subtree is expanded. If a best-first search algorithm (which stores the whole search tree in memory) is used, the search tree is converted into a search graph, by joining identical nodes into one node. This causes subtrees to be merged, decreasing the need for memory. Since the graph contains fewer nodes than the tree, less searching is needed as well. However, joining identical nodes into one node introduces the so-called *graph-history-interaction* (GHI) problem, since determining whether *nodes* are identical is not the same as determining whether the *search states* represented by the nodes are identical.

This is laid down in the third problem statement, addressing both the decrease in the need for memory *and* the decrease in the need for search.

Problem statement 3: Is it possible to give a solution for the GHI problem for best-first search?

1.5 Outline of the thesis

The contents of the thesis is as follows.

Chapter 1 contains an introduction, the three problem statements and an outline of the thesis.

Chapter 2 tries to answer the first problem statement by discussing enhancements to the implementation of a transposition table. First, some important notions and concepts, used throughout the thesis, are defined. Thereafter, it is explained why a transposition table is needed. Next, implementation details are given and an experimental set-up is presented. Three series of experiments are described: (1) when different positions compete for storage in the same entry of the transposition table, a *replacement scheme* has to be used for priority arguments; several replacement schemes are compared; (2) the merits of storing different characteristics of a position are quantified; (3) the use of additional memory is discussed, and it appears that there is still room for improvements. Finally, it is shown that a transposition table is a useful way of reducing the search at the cost of using more memory.

Chapter 3 presents the pn-search algorithm. Experiments with pn search have been performed to obtain more insight into the strengths and weaknesses of this algorithm when applied to a complex game such as chess (i.e., to positions of which it is possible to prove the game-theoretic value). The algorithm will be used as a test bed in the Chapters 4 and 5. First, an informal description of the algorithm is given, followed by the pseudo-code. Experiments are reported, comparing the pn-search algorithm to the $\alpha\beta$-search algorithm. The strengths and weaknesses of the pn-search algorithm are discussed.

Chapter 4 tries to answer the second problem statement and presents the pn^2-search algorithm. This is a modification of the pn-search algorithm when only little memory is available, using less memory at the cost of more searching. Experiments are given, showing that this algorithm solves more positions in the test set than the standard pn-search algorithm, implying that pn^2 search is a useful algorithm when little memory is available.

Chapter 5 answers the third problem statement. A review of attempted solutions to the GHI problem is presented. Next, our practical solution for best-first search algorithms that keep the whole search tree in memory is presented. Thereafter, the pseudo-code for the implementation in pn search is shown. This algorithm is compared to the standard pn-search algorithm and its modifications. Experiments are reported, showing that this graph algorithm for pn search performs well.

The evaluation of the three problem statements, final conclusions, and future research are given in Chapter 6.

Appendix A lists the test set used for the chess middle-game transposition-table experiments. The test set used for the chess endgame transposition-table experiments is presented in Appendix B. Appendix C lists the results of all experiments described in Chapter 2. The test set used for the proof-number search experiments is given in Appendix D. Appendix E presents the results of the pn-search and $\alpha\beta$-search experiments described in Chapter 3. The results of the pn^2 experiments described in Chapter 4 are shown in Appendix F. Finally, Appendix G lists and compares the results of the experiments given in Chapter 5 with the results of the pn tree algorithm.