Summary

Memory versus search in games

In this thesis, research is presented on the trade-off between memory and search. The domain under investigation is the domain of two-player zero-sum games, in particular the games of chess and domineering. The trade-off between memory and search is enhanced by the increase in availability of computer memory and the increase in processor speed.

Currently, the prices of computer memory are decreasing. Therefore, acquiring larger memory configurations is no longer an obstacle, making it easier to equip a computer with more memory. A depth-first search algorithm (such as $\alpha\beta$ search) uses little memory. The large amount of remaining memory can be used, e.g., to prevent the re-search of transpositions (identical positions in the tree). For this purpose, a transposition table, holding the results of previous searches, is maintained in the remaining memory. The trade-off transpires in more memory to be used, in favour of less searching. This leads to the formulation of the first problem statement.

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

In Chapter 2 three methods for improving the efficiency of a transposition table are described. The first method addresses the use of an adequate replacement scheme. When a conflict arises, a replacement scheme decides which positions to keep in the table, and which positions to discard. Experiments show that in this area improvements can still be found. A new replacement scheme, called TwoBig1, based on a two-level table and the number of nodes of the subtree investigated, outperforms all other schemes. It enabled us to solve the game of domineering for several boards, including the standard board. The second method addresses doubling the number of positions in the transposition table. Experiments show that doubling the number of positions is a good method for improving the efficiency of a transposition table. However, beyond a certain table size not much is to be gained from doubling the number of positions. Therefore, the third method concentrates on using the remaining memory not for doubling the number of positions of the table, but for enlarging the size of an entry, by storing more information in an entry. A limited

set of experiments show that – beyond a certain table size – this method gains more than doubling the number of positions in the table, although more experiments are needed to substantiate this claim.

In Chapter 3 proof-number search (pn search) is described. This is a best-first search algorithm, storing the complete search tree in memory. Experiments show that pn search is suitable for solving mate problems in chess. However, there are two drawbacks: (1) a solution cannot be found if the search tree takes up all memory, and (2) identical positions in the search tree (and their subtrees) are doubly searched. These drawbacks are taken care of in Chapters 4 and 5.

Every year there is a large increase in computer speed. Increasing computer speed causes acceleration of search algorithms. A best-first search algorithm (such as pn search) stores the complete search tree in memory. After a relatively short search time no more memory is available since the fast search has generated too many nodes. The increase in computer speed can also be used to do more search at nodes, thereby gaining more knowledge per node. The trade-off transpires in more searching, in favour of less memory to be used. This leads to the formulation of the second problem statement.

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 Chapter 4 the pn^2-search algorithm is presented. The concept behind this algorithm is that the leaves are not evaluated by an evaluation function, but by a secondary pn-search process. Several experiments with different sizes of the secondary search tree show that much can be gained by choosing the right size of the secondary search tree. The conclusion is that the pn^2-search algorithm is a good method to use the increase in computer speed for additional searching, thereby gaining a better assessment of the values of the leaves.

As mentioned above, in pn search identical positions in the search tree (and their subtrees) are doubly searched. In depth-first search algorithms the re-search of a transposition is avoided by implementing a transposition table. A logical way to avoid the re-search of a transposition in best-first search is to store a transposition only once, thereby transforming the tree into a Directed Cyclic Graph (DCG). However, an important aspect of a position is the path leading to it (the history). Ignoring the history of a position introduces the graph-history-interaction (GHI) problem. This leads to the third problem statement.

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

In Chapter 5 the GHI problem is analyzed in the domain of pn search. A different implementation of a DCG is suggested, and the pn-search algorithm is modified to be able to search this DCG implementation. The new BTA (Base-Twin Algorithm) algorithm is based on the distinction of two types of nodes, termed *base nodes* and

twin nodes. The purpose of these types is to distinguish between equal positions with different history. Experiments with this pn-search algorithm for DCGs confirm our solution of the GHI problem. In the test positions submitted the BTA algorithm solves them all and hence outperforms other attempts to overcome the GHI problem as well as the standard tree algorithm.

Summarizing, the main contributions of this thesis are as follows.

1. The discovery of a new replacement scheme (TwoBig), based on a two-level transposition table and number of nodes of the subtree investigated.
2. Solving the game of domineering.
3. The pn^2-search algorithm.
4. The BTA algorithm (implemented for pn search), solving the GHI problem.

Samenvatting

Geheugen versus zoeken in spelen

In dit proefschrift wordt onderzoek gepresenteerd betreffende de uitwisseling tussen geheugen en zoeken. Het onderzoeksdomein is het domein van de tweepersoons nulsom spelen, in het bijzonder de spelen schaken en domineering. De uitwisseling tussen geheugen en zoeken wint aan belangrijkheid door het beschikbaar komen van meer computergeheugen en meer processorsnelheid.

Computergeheugen wordt steeds goedkoper, en komt daardoor in steeds grotere mate beschikbaar. Een *depth-first* zoekalgoritme (zoals $\alpha\beta$ *search*) gebruikt weinig geheugen. Het resterende geheugen kan bijvoorbeeld gebruikt worden om het heronderzoeken van identieke stellingen, de zogenoemde transposities, te vermijden. Daartoe kan een transpositietabel, die resultaten van voorgaande zoekprocessen bewaart, in het resterende geheugen worden opgeslagen. De uitwisseling zien we terug in het gebruik van meer geheugen, zodat minder hoeft te worden gezocht. Dit resulteert in de formulering van de eerste probleemstelling.

Probleemstelling 1: Welke methoden bestaan er om de efficiëntie van een transpositietabel te verbeteren?

In Hoofdstuk 2 worden drie methoden beschreven om de efficiëntie van een transpositietabel te verbeteren. De eerste methode betreft het gebruik van een adequaat vervangingsschema. Deze methodiek bepaalt bij een conflict welke stellingen wel, en welke niet opgeslagen worden. Experimenten tonen aan dat op dit terrein nog steeds verbeteringen gevonden kunnen worden. Een nieuw vervangingsschema, genaamd TwoBig1, gebaseerd op een *two-level* tabel en het aantal knopen van de onderzochte subboom, presteert beter dan alle andere schema's. Dit schema maakte het mogelijk om het spel domineering op te lossen voor verscheidene borden, waaronder het standaard bord. De tweede methode betreft het verdubbelen van het aantal stellingen in een transpositietabel. Experimenten tonen aan dat het verdubbelen van het aantal stellingen een goede methode is om de efficiëntie van een transpositietabel te verbeteren. Vanaf een bepaalde tabelgrootte valt echter weinig winst meer te behalen met het verdubbelen van het aantal stellingen. De derde methode gebruikt het resterende geheugen daarom niet om het aantal stellingen in de tabel te verdubbelen, maar om het formaat van een tabelingang te vergroten, door meer informatie

in een ingang op te slaan. Een beperkt aantal experimenten toont aan dat – vanaf een bepaalde tabelgrootte – deze methode meer oplevert dan het verdubbelen van het aantal stellingen in de tabel.

In Hoofdstuk 3 wordt *proof-number search (pn search)* beschreven. Dit is een *best-first* zoekalgoritme, dat de gehele zoekboom in het geheugen opslaat. Experimenten tonen aan dat *pn search* geschikt is voor het oplossen van matproblemen in schaken. Er zijn echter twee nadelen: (1) er wordt geen oplossing gevonden als het geheugen vol raakt, en (2) identieke stellingen in de zoekboom (en hun subbomen) worden dubbel onderzocht. In Hoofdstukken 4 en 5 wordt op deze nadelen ingegaan.

De snelheid van computers wordt ieder jaar groter. Vergroting van de computersnelheid betekent automatisch ook versnelling van het zoeken. Een *best-first* zoekalgoritme (zoals *pn search*) slaat de gehele zoekboom op in het geheugen. Na een relatief korte zoektijd is geen geheugen meer beschikbaar omdat het snelle zoekproces teveel knopen heeft gegenereerd. De vergroting van de computersnelheid kan echter ook gebruikt worden om meer te zoeken bij de knopen, waardoor meer kennis per knoop wordt verkregen. De uitwisseling komt terug in meer zoeken, zodat minder geheugen gebruikt hoeft te worden. Dit resulteert in de formulering van de tweede probleemstelling.

Probleemstelling 2: Welke methoden bestaan er voor *best-first* zoekalgoritmen om de vraag naar geheugen te verminderen, door meer te zoeken en daardoor meer kennis per knoop te verkrijgen?

In Hoofdstuk 4 wordt het *pn*^{2}*-search* algoritme gepresenteerd. Het concept achter dit algoritme is dat de bladeren niet door een evaluatiefunctie worden geëvalueerd, maar door een tweede *pn search* proces. Verscheidene experimenten met verschillende grootten van de tweede zoekboom tonen aan dat veel gewonnen kan worden door de juiste grootte van de tweede zoekboom te kiezen. De conclusie is dat *pn*^{2} *search* een goede methode is om de vergroting van de computersnelheid te gebruiken om meer te zoeken, waarbij een betere schatting van de waarden van de bladeren wordt verkregen.

Zoals hierboven is genoemd worden in *pn search* identieke stellingen in de zoekboom (en hun subbomen) dubbel onderzocht. In *depth-first* zoekalgoritmen wordt de heronderzoeking van een transpositie vermeden door het gebruik van een transpositietabel. Een logische manier om de heronderzoeking van een transpositie in een *best-first* zoekalgoritme te vermijden is om de transpositie maar één keer op te slaan, waardoor de boom wordt veranderd in een Gerichte Cyclische Graaf (DCG). Een belangrijk aspect van een stelling is echter het pad dat tot deze stelling leidt (de geschiedenis). Het negeren van van de geschiedenis van een stelling introduceert het *graph-history-interaction* (GHI) probleem. Dit resulteert in de derde probleemstelling.

Problemstelling 3: Is het mogelijk om een oplossing te geven voor het GHI probleem voor *best-first* zoekalgoritmen?

In Hoofdstuk 5 wordt het GHI probleem geanalyseerd in het domein van *pn search*. Er wordt een andere implementatie van een DCG geopperd, en het *pn search* algoritme wordt gewijzigd om het mogelijk te maken om deze DCG implementatie te onderzoeken. Het nieuwe BTA (*Base-Twin Algorithm*) algoritme is gebaseerd op het onderscheid tussen twee typen knopen, genaamd *base nodes* en *twin nodes*. Het doel van deze typen is om een onderscheid te kunnen maken tussen knopen met verschillende geschiedenissen. Experimenten met dit *pn-search* algoritme voor DCGs bekrachtigen onze oplossing van het GHI probleem. Het BTA algoritme lost alle teststellingen op en presteert dientengevolge beter dan zowel andere pogingen om het GHI probleem te overwinnen als het standaard algoritme voor bomen.

Samenvattend kunnen de hoofdbijdragen van dit proefschrift als volgt geformuleerd worden.

1. De ontdekking van een nieuw vervangingsschema (TwoBig), gebaseerd op een *two-level* transpositietabel en het aantal knopen van de onderzochte subboom.
2. Het oplossen van het spel domineering.
3. Het pn^2-*search* algoritme.
4. Het BTA algoritme (geïmplementeerd in *pn search*), dat het GHI probleem oplost.