ECAI 2008
M. Ghallab et al. (Eds.)
IOS Press, 2008

doi:10.3233/978-1-58603-891-5-545

Using Abstraction in Two-Player Games

Mehdi Samadi , Jonathan Schaeffer[1], Fatemeh Torabi Asr , Majid Samar , Zohreh Azimifar [2]

Abstract. For most high-performance two-player game programs, a significant amount of time is devoted to developing the evaluation function. An important issue in this regard is how to take advantage of a large memory. For some two-player games, endgame databases have been an effective way of reducing search effort and introducing accurate values into the search. For some one-player games (puzzles), pattern databases have been effective at improving the quality of the heuristic values used in a search.

This paper presents a new approach to using endgame and pattern databases to assist in constructing an evaluation function for two-player games. Via abstraction, single-agent pattern databases are applied to two-player games. Positions in endgame databases are viewed as an abstraction of more complicated positions; database lookups are used as evaluation function features. These ideas are illustrated using Chinese checkers and chess. For each domain, even small databases can be used to produce strong game play. This research has relevance to the recent interest in building general game-playing programs. For two-player applications where pattern and/or endgame databases can be built, abstraction can be used to automatically construct an evaluation function.

1 Introduction and Overview

Almost half a century of AI research into developing high-performance game-playing programs has led to impressive successes, including DEEP BLUE (chess), CHINOOK (checkers), TD-GAMMON (backgammon), LOGISTELLO (Othello), and MAVEN (Scrabble). Research into two-player games is one of the most visible accomplishments in artificial intelligence to date.

The success of these programs relied heavily on their ability to search and to use application-specific knowledge. The search component is largely well-understood for two-player games (whether perfect or imperfect information; stochastic or not); usually the effort goes into building a high-performance search engine. The knowledge component varies significantly from domain to domain. Various techniques have been used, including linear regression (as in LOGISTELLO) and temporal difference learning (as in TD-GAMMON). All of them required expert input, especially the DEEP BLUE [10] and CHINOOK [16] programs.

Developing these high-performance programs required substantial effort over many years. In all cases a major commitment had to be made to developing the program's evaluation function. The standard way to do this is by hand, using domain experts if available. Typically, the developer (in consultation with the experts) designs multiple evaluation function features and then decides on an appropriate weighting for them. Usually the weighted features are summed to form the assessment. This technique has proven to be effective, albeit labour intensive. However, this method fails in the case of a new game or for one in which there is no expert information available (or no experts). The advent of the annual General Game Playing (GGP) competition at AAAI has made the community more aware of the need for general-purpose solutions rather than custom solutions.

Most high-performance game-playing programs are compute intensive and benefit from faster and/or more CPUs. An important issue is how to take advantage of a large memory. Transposition tables have proven effective for improving search efficiency by eliminating redundancy in the search. However, these tables provide diminishing returns as the size increases [3]. For some two-player games, endgame databases (sometimes called tablebases) have been an effective way of reducing search effort and introducing accurate values into the search. These databases enumerate all positions with a few pieces on the board and compute whether each position is a provable win, loss or draw. Each database position, however, is applicable to only one position.

The single-agent (one-player) world has also wrestled with the memory issue. Pattern databases have been effective for improving the performance of programs to solve numerous optimization problems, including the sliding-tile puzzles and Rubik's Cube [8]. They are similar to endgame databases in that they enumerate a subset of possible piece placings and compute a metric for each (e.g., minimum number of moves to a solution). The databases are effective for two reasons. First they can be used to provide an improved lower bound on the solution quality. Second, using abstraction, multiple states can be mapped to a single database value, increasing the utility of the databases.

The main theme of this paper is to investigate and propose a new approach to use endgame and pattern databases to assist in automating the construction of an evaluation function for two-player games. The research also carries over to multi-player games, but this is not addressed in this paper. The key idea is to extend the benefits of endgame and pattern databases by using abstraction. Evaluation of a position with N pieces on the board is done by looking up a subset of pieces $M < N$ in the appropriate database. The evaluation function is built by combining the results of multiple lookups and by learning an appropriate weighting of the different lookups. The algorithm is simple and produces surprisingly strong results. Of greater importance is that this is a new general way to use the databases.

The contributions of this research are as follows:

1. Abstraction is used to extend pattern databases (even additive pattern databases) for constructing evaluation functions for a class of two-player games.
2. Pattern-database-based evaluation functions are shown to produce state-of-the-art play in Chinese checkers (10 pieces a side). Against a baseline program containing the latest evaluation func-

[1] Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2E8, email: {msamadi,jonathan}@cs.ualberta.ca
[2] Department of Computer Science and Engineering Shiraz University, Shiraz, Iran, email:{torabi,samar,azimifar}@cs.shirazu.ac.ir

tion enhancements, the pattern-database-based program scores 68% to 79% of the possible points.
3. Abstraction is used to extend endgame databases for constructing evaluation functions for a class of two-player games.
4. Chess evaluation functions based on four- and five-piece endgame databases are shown to outplay CRAFTY, the strongest freeware chess program available. On seven- and eight-piece chess endgames, the endgame-database program scores 54% to 80% of the possible points.

Abstraction is a key to extending the utility of the endgame and pattern databases. For domains for which these databases can be constructed, they can be used to build an evaluation function automatically. As the experimental results show, even small databases can be used to produce strong game plays.

2 Related Work

Endgame databases have been in use for two-player perfect information games for almost thirty years. They are constructed using retrograde analysis [18]. Chess was the original application domain, where databases for all positions with six or fewer pieces have been built. Endgame databases were essential for solving the game of checkers, where all positions with ten or fewer pieces have been computed [6]. The databases are important because they reduce the search tree and introduce accurate values into the search. Instead of using a heuristic to evaluate these positions (with the associated error), a game-playing program can use the database value (perfect information). The limitation, however, is that each position in the database is applicable to a single position in the search space.

Pattern databases also use retrograde analysis to optimally solve simplified versions of a state space [4]. A single-agent state space is abstracted by simplifying the domain (e.g., only considering a subset of the features) and solving that problem. The solutions to the abstract state are used as lower bounds for solutions to a set of positions in the original search space. For some domains, pattern databases can be constructed so that two or more database lookups can be added together while still preserving the optimality of the combined heuristic [13]. Abstraction means that many states in the original space can use a single state in the pattern database. Pattern databases have been used to improve the quality of the heuristic estimate of the distance to the goal, resulting in many orders of magnitude reduction in the effort required to solve the sliding-tile puzzles and Rubik's Cube [8].

The ideas presented in this paper have great potential for General Game Playing (GGP) programs [9]. A GGP program, given only the rules of the game/puzzle, has to learn to play that game/puzzle well. A major bottleneck to producing strong play is the discovery of an effective evaluation function. Although there is an interesting literature on feature discovery applied to games, to date the successes are small [7]. It is still early days for developing GGP programs, but the state of the art is hard coding into the program several well-known heuristics that have been proven to be effective in a variety of games, and then testing them to see if they are applicable to the current domain [15]. It remains an open problem how to automate the discovery and learning of an effective evaluation function for an arbitrary game.

3 Using Abstraction in Two-Player Games

Abstraction is a mapping from a state in the original search space into a simplified representation of that state. The abstraction is often a relaxation of the state space or a subset of the state. In effect, abstraction maps multiple states in the original state space to a single state in the abstract search space. Information about the abstract state (e.g., solution cost) can be used as a heuristic for the original state (e.g., a bound on the solution cost).

Here we give the background notation and definitions using chess as the illustrative domain. Let S be the original search space and S' be the abstract search space.

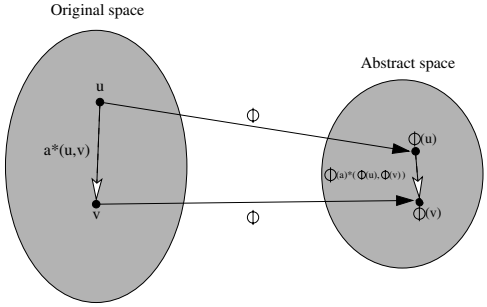

Figure 1. Original states and edges mapped to an abstract space.

Definition 1 (Abstraction Transformation): An abstraction transformation $\phi : S \rightarrow S'$ maps 1) states $u \in S$ to states $\phi(u) \in S'$, and 2) actions $a \in S$ to actions $\phi(a) \in S'$. This is illustrated in Figure 1.

Consider the chess endgame of white king, rook, and pawn versus black king and rook (KRPKR). The original space (S) consists of all valid states where these five pieces can be placed on the board. Any valid subset of the original space can be considered as an abstraction. For example, king and rook versus king (KRK) and king, rook, and pawn versus king (KRPK) are abstractions (simplifications) of KRPKR. For any particular abstraction S', the search space contains all valid states in the abstract domain (all piece location combinations). The new space S' is much smaller than the original space S, meaning that a large number of states in S are being mapped to a single state in S'. For instance, for every state in the abstracted KRK space, all board positions in S where the white king, white rook and black king are on the same squares as in S' are mapped onto a single abstract state (i.e., white pawn and black rook locations are abstracted away). Actions in S' contain all valid moves for the pieces that are in the abstracted state.

Definition 2 (Homomorphism): An abstraction transformation ϕ is a homomorphism transformation if for all series of actions that transforms state u to state v in S then there is a corresponding transformation for $\phi(u)$ to $\phi(v)$ in S'. This is illustrated in Figure 1, where a^* represents zero or more actions.

If there is a solution for a state in the original space S, then the homomorphism property guarantees the existence of a solution in the abstracted space S'. Experimental results indicate that this characteristic can be used to improve search performance in S.

Various abstractions can be generated for a given search problem. The set of relaxing functions is defined as $\phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$, where each ϕ_i is an abstraction. Define the distance between any two states u and v in the relaxed environment as ϕ_i with $h_{\phi_i}(u, v)$. For example, for an endgame or pattern database, v is usually set to a goal state meaning that $h_{\phi_i}(u, v)$ is the minimal number of moves needed to achieve the goal.

Using off-line processing, the distance from each state in ϕ_i to the nearest goal can be computed and saved in a database (using retrograde analysis). For a pattern database (one-player search), the minimal distance to the goal is stored. For an endgame database (two-

player search), the minimal number of moves to win (maximal moves to postpone losing) are recorded. This is the standard way that these databases are constructed.

Given a problem instance to solve, during the search all values from those lookup tables are retrieved for further processing. To evaluate a position p from the original space, the relaxed state, $\phi_i(p)$, is computed and the corresponding $h_{\phi_i(p)}$ is retrieved from the database. The abstract values are saved in a heuristic vector $h =< h_{\phi_1}, h_{\phi_2}, \ldots, h_{\phi_n} >$. The evaluation function value for state p is calculated as a function of h. For example, popular techniques used for two-player evaluation functions include temporal difference learning to linearly combine the h_{ϕ_i} values [1], and neural nets to achieve non-linear relations [17].

For example, let us evaluate a position p in the KRPKR chess endgame. In this case, the abstracted states could come from the databases KRPK, KRKR, KRK and KPK. First, for each abstraction, the abstract state is computed and the heuristics value $h_{\phi_i(p)}$ is retrieved from the database. In this case, the black rook is removed and the resulting position is looked up in the KRPK database; the white pawn is removed and the position looked up in the KRKR database; etc. The heuristic value for p could be, for example, the sum of the four abstraction scores.

4 Experimental Results

In this section, we explore using abstraction to apply pattern database technology to two-player Chinese checkers and chess endgame database technology to playing more complicated chess endgames. Unlike chess, Chinese checkers has the homomorphism property (the proof is simple, but not shown here for reasons of space).

4.1 Chinese Checkers

Chinese checkers is a 2-6 player game played on a star shaped board with the squares hexagonally connected. The objective is to move all of one's pieces (or marbles) from the player's home zone (typically 10) to the opposite side of the board (the opponent's home zone). Each player moves one marble each turn. A marble can move by rolling to an adjacent position (one of six) or by repeatedly jumping over an adjacent marble, of any color, to an adjacent empty location (the same as jumps in 8×8 checkers/draughts). In general, to reach the goal in the shortest possible time, the player should jump his pieces towards the opponent's home zone.

Here we limit ourselves to two-player results, although the results presented here scale well to more players (not reported here). Due to the characteristics of Chinese checkers, three different kinds of abstractions might be considered. Given N pieces on each side of the original game:

1. Playing $K \leq N$ white pieces against $L \leq N$ black pieces;
2. Playing $K \leq N$ white pieces to take them to opponent's home zone (a pattern database including no opponent's marble); and
3. Playing $K \leq N$ white pieces against $L \leq N$ black pieces, but with a constraint that the play concentrates on a partition of the board.

For any given search space, the more position characteristics that are exploited by the set of abstractions, the more likely that the combination of abstraction heuristics will be useful for the original problem space. The first two abstractions above have the homomorphism property, and the empirical results indicate that they better approximate the original problem space. In the first abstraction, a subset of pieces for both players (e.g., the three-piece versus two-piece game) is considered and the minimal number of moves to win (most moves to lose) is used. The second abstraction ignores all the opponent's pieces. This abstraction gives the number of moves required to get all of one's pieces into the opponent's zone. This value is just a heuristic estimate (not a bound), since the value does not take into account the possibility of jumping over the opponent's pieces (which precludes it from being a lower bound) and does not take into account interference from the opponent's pieces (precluding it from being an upper bound). Clearly, the first abstraction is a better representation of the original problem space. The third abstraction considers only a part of the board to build a pattern database. For example, the goal of the abstraction can be changed so that the pieces only have to enter the goal area (without caring about where the end up).

The state space for the first abstraction is large; the endgame database of three versus two pieces requires roughly 256MB. The second relaxation strategy makes the search space simpler, allowing for pattern databases that include more pieces on the board. The database size for five pieces of the same side needs roughly 25MB, 10% of the first abstraction database. Our experience with Chinese checkers shows that during the game five cooperating pieces will result in more (and longer) jump moves (hence, less moves to reach the goal) than five adversarial pieces. Although the first abstraction looks more natural and seems to better reflect the domain, the second abstraction gives better heuristic values. Thus, here we present only the second and third abstractions.

The baseline for comparison is a Chinese checkers program (10 pieces a side) with all the current state-of-the-art enhancements. The evaluation function is based on the Manhattan distance for each side's pieces to reach the goal area. Recent research has improved on this simple heuristic by adding additional evaluation terms: 1) curved board model, incremental evaluation, left-behind marbles [19]; and 2) learning [11]. All of these features have been implemented in our baseline program.

Experiments consisted of the baseline program playing against a program using a PDB- or endgame-based evaluation function. Each experimental data point consists of a pair of games (switching sides) for each of 25 opening positions (after five random moves have been made). Experiments are reported for search depths of three and five ply (other search depth results are similar). The branching factor in the middlegame of Chinese checkers is roughly 60-80. Move generation can be expensive because of the combination of jumps for each side. This slows the program down, limiting the search depth that can be achieved in a reasonable amount of time. The average response time for a search depth of six in the middlegame is more than thirty seconds per move (1.5 hours per game). Our reported experiments are limited to depths three through five because of the wide range of experiments performed.

In this paper, we report the results for three interesting heuristic evaluation functions. Numerous functions were experimented with and achieved similar performance to those reported here. For the following abstractions, the pieces were labeled 1 to 10 in a right-to-left, bottom-up manner. The abstractions used were:

PDB(4): four-piece pattern database (second abstraction) with the goal defined as the top four squares in the opponent's home zone. Three abstractions (three lookups) were used to cover all available ten pieces: pieces 1-4, 4-7, and 7-10. We also tested other lookups on this domain. Obviously increasing the number of lookups can increase the total amount of time to evaluate each node. On the other hand, the overlap of using pieces four and seven in the evaluation function does not have a severe effect on the cost of an

evaluation function.

PDB(6): six-piece pattern database (second abstraction) with the goal defined as the top six squares in the opponent's home zone. Two abstractions (two lookups) were used to cover all 10 pieces: pieces 1-6 and 5-10. Again, two pieces are counted twice in an evaluation (pieces 5 and 6), as a consequence of minimizing the execution overhead.

PDB(6+4): a probe from the six-piece PDB is added to a probe from the four-piece PDB (a combination of second and third abstraction). Two abstractions (two lookups) were used to cover all 10 pieces: pieces 1-6 from the PDB(6) and 7-10 from the PDB(4) with its goal defined as passing all pieces from the opponent's front line (third abstraction). In other words, for the four-piece abstraction we delete the top six squares of the board such that the new board setup introduces our new goal.

The weighting of each probe is a simplistic linear combination of the abstraction heuristic values.

Abstraction (Pieces)	Search Depth	Win %
PDB (4)	3	79
PDB (6)	3	68
PDB (6+4)	3	74
PDB (4)	4	69
PDB (6)	4	68
PDB (6+4)	4	80
PDB (4)	5	78
PDB (6)	5	70
PDB (6+4)	5	78

Table 1. Experiments in Chinese checkers.

Table 1 presents the results achieved using these abstractions. The three rows of results are given for each of search depths three, four and five. The win percent reflects two points for a win, one for a tie and zero for a loss.

Evidently, PDB(6+4) has the best performance, winning about 80% of the games against the baseline program. Perhaps surprisingly, PDB(4) performs very well, even better than PDB(6) does. One would expect PDB(6) to perform better since it implicitly contains more knowledge of the pieces interactions. However, note that the more pieces on the board, the more frequent long jump sequences will occur. The longer the jump sequence the smaller the probability that it can be realized, given that there are other pieces on the board. Hence, we conclude that a larger PDB may not be as accurate as a smaller PDB.

The additive evaluation function (using PDB(4+6)) gives the best empirical results. Not only is it comparable to the PDB(4), but also it achieves its good performance with one fewer database lookup per evaluation. Although the experiments were done to a fix search depth (to ensure a uniform comparison between program versions), because of the relative simplicity of the evaluation function an extra database lookup represented a significant increase in execution cost. In part this is due to the pseudo-random nature of accessing the PDB, possibly incurring cache overhead. Our implementation takes advantage of obvious optimizations to eliminate redundant database lookups (e.g., reusing a previous lookup if still applicable). By employing these optimizations, we observed that the time for both heuristic functions are very close and does not change the results.

Several experiments were also performed using the first abstraction, with three-against-two-piece endgame databases. A program based on the second abstraction (pattern databases with five pieces) significantly outperformed the first abstraction. The values obtained from five cooperating pieces were a better heuristic predictor than that obtained from the adversarial three versus two pieces database.

The results reported here do not necessarily represent the best possible. There are numerous combinations of various databases that one can try. The point here is that simple abstraction can be used to build an effective evaluation function. In this example, single-agent pattern databases are used in a new way for two-player heuristic scores.

4.2 Chess

This section presents experimental results for using four- and five-piece chess endgame databases to play seven- and eight-piece chess endgames. The abstracted state space is constructed using a subset of available pieces. For example, for the KRPKR endgame one can use the KRK, KRPK, and KRKR subset of pieces as abstractions of the original position. All the abstractions are looked up in their appropriate database. The endgame databases are publicly available at numerous sites on the Internet. For each position, they contain one of the following values: win (the minimum number of moves to mate the opponent), loss (the longest sequence of moves to be mated by the opponent) or draw (zero). The values retrieved from the abstractions are used as evaluation function features. They are linearly combined; no attempt at learning proper weights has been done yet.

In chess, as opposed to Chinese checkers, ignoring all the opponent pieces does not improve the performance given the tight mutual influence they have on each other (i.e., piece captures are possible). Hence pattern databases are unlikely to be effective. One could use pattern databases for chess, even though, we expect a learning algorithm to discover a weight of zero for such abstractions.

The chess abstraction does not have the homomorphism property because of the mutual interactions among the pieces. In other words, it is possible to win in the original position while not achieving this result in the abstract position. For example, there are many winning positions in the KRPKR endgame but in the abstraction of KRKR almost all states lead to a draw.

Our experiments used the four- and five-piece endgame databases. Note that the state-of-the-art in endgame database construction is six pieces [2]. These databases are too large to fit into RAM, making their access cost prohibitively high. Evaluation functions must be fast, otherwise they can dramatically reduce the search speed. Hence we restrict ourselves to databases that can comfortably fit into less than 1GB of RAM. This work will show that even the small databases can be used to improve the quality of play for complex seven- and eight-piece endgames.

In our experiments the proposed engine (a program consisting solely of an endgame-database-based evaluation) played against the baseline program (as the opponent). Each experimental data point consisted of a pair of games (switching sides) for each of 25 endgame positions. The programs searched to a depth of seven and nine ply. Results are reported using four- and five-piece abstractions of seven- and eight-piece endgames. Because of the variety of experiments performed, the search depth was limited to nine.

The baseline considered here is CRAFTY, the strongest freeware chess program available [12]. It has competed in numerous World Computer Chess Championships, often placing near the top of the standings.

Table 2 shows the impact of two parameters on performance: the endgame database size and the search space depth. The table gives results for three representative seven-piece endgames. The first column gives the endgame, the second gives the win percentage (as stated before, wins is counted as two, draws as one and losses as zero), and

Endgame	Search Depth	Win %	Abstractions Used
KRPP–KBN	7	60	KPPK, KKBN, KRK
KRPP–KNN	7	68	KRK, KRKP, KRPK, KNKP
KRP–KNPP	7	72	KKPP, KBKP, KPKN, KRK
KRPP–KBN	7	68	KPKBN, KRPKB, KRPKN
KRPP–KNN	7	76	KRPKN, KPPKN, KPKNN
KRP–KNPP	7	80	KRPKN, KRKNP, KPKNP, KPKPP
KRPP–KBN	9	54	KPPK, KKBN, KRK
KRPP–KNN	9	64	KRK, KRKP, KRPK, KNKP
KRP–KNPP	9	70	KKPP, KBKP, KPKN, KRK
KRPP–KBN	9	56	KPKBN, KRPKB, KRPKN
KRPP–KNN	9	68	KRPKN, KPPKN, KPKNN
KRP–KNPP	9	76	KRPKN, KRKNP, KPKNP, KPKPP

Table 2. Experiments in chess (four-piece and five-piece abstractions).

the last column shows the abstractions used. The first six lines are for a search depth of seven; the remaining six for a search depth of nine. For each depth, the first three lines show the results for using three- and four-piece databases as an abstraction; the last three rows show the results when five-piece databases are used.

CRAFTY was used unchanged. It had access to the same endgame databases as our program, but it only used them when the current position was in the database. For all positions with more pieces, it used its standard endgame evaluation function. In contrast, our program, using abstraction, queried the databases every time a node in the search required to be evaluated. By eliminating redundant database lookups, the cost of an endgame-database evaluation can be made comparable to that of CRAFTY's evaluation.

Not surprisingly, the five-piece databases had superior performance to the four-piece databases (roughly 8% better for depth seven and 4% better at depth nine). Clearly, these databases are closer to the original position (i.e., less abstract) and hence are more likely to contain relevant information. Further, a significant drawback of small-size abstraction models is the large number of draw states in the database (e.g. KRKR), allowing little opportunity to differentiate between states. The five-piece databases contain fewer draw positions, giving greater decision domain to the evaluation function.

As the search depth is increased, the benefits of the superior evaluation function slightly decrease. This is indeed expected, as the deeper search allows more potential errors by both sides to be avoided. This benefits the weaker program.

Position	Search Depth	Win %	Abstractions Used
KQP–KRNP	7	64	KQKRP, KQKNP, KPKRN , KQKRN
KRRPP–KQR	7	76	KQKRP, KQKNP, KPKRN
KRPP–KRN	7	60	KRPKN, KPPKR, KPKRN
KQP–KNNPP	7	76	KPKNN, KQKNN, KQKNP
KQP–KRBPP	7	64	KPKNN, KQKNN, KQKNP
KQP–KRNP	9	64	KQKRP, KQKNP, KPKRN, KQKRN
KRRPP–KQR	9	76	KQKRP, KQKNP, KPKRN
KRPP–KRN	9	64	KRPKN, KPPKR, KPKRN
KQP–KNNPP	9	72	KPKNN, KQKNN, KQKNP
KQP–KRBPP	9	62	KQKRB, KQPKR, KQKRP, KQKBP

Table 3. Experiments for chess.

Table 3 shows the results for some interesting (and complicated) seven- and eight-piece endgames, all using five-piece abstraction. These represent difficult endgames for humans and computers to play. Again, the endgame-database-based evaluation function is superior to CRAFTY, winning 60% to 76% of the games. This performance is achieved using three or four abstraction lookups, in contrast to CRAFTY's hand-designed rule-based system.

Why is the endgame database abstraction effective? The abstraction used for chess is, in part, adding heuristic knowledge to the evaluation function about exchanging pieces. In effect, the smaller databases are giving information about the result when pieces come off the board. This biases the program towards lines which result in favorable piece exchanges, and avoids unfavorable ones.

5 Conclusion and Future Works

The research presented in this paper is a step towards increasing the advantages of pre-computed lookup tables for the larger class of multi-agent problem domains. The main contribution of this research was to show that the idea of abstraction can be used to extend the benefits of pre-computed databases for use in new ways in building an accurate evaluation function. For domains for which pattern and/or endgame databases can be constructed, the use of this data can be extended beyond its traditional usage and be be used to build an evaluation function automatically. As the experimental results show, even small databases can be used to produce strong game play.

Since 2005, there has been interest in the AI community in building a general game-playing (GGP) program. The application-specific research in building high-performance games is being generalized to handle a wide class of games. Research has already been done in identifying GGP domains for which databases can be built [14]. For those domains, abstraction is a promising way to automatically build an evaluation function. An automated system has been developed to build a pattern databases for planning domains using bin-packing algorithm to select the appropriate symbolic variables for pattern database [5]. Similar approach can be used to automatically select variables in GGP to build endgame/pattern databases.

REFERENCES

[1] J. Baxter, A. Tridgell, and L. Weaver, 'Learning to play chess using temporal differences', *Machine Learning*, **40**(3), 243–263, (2000).

[2] E. Bleicher, 2008. `http://k4it.de/index.php?topic=egtb&lang=en`.

[3] D. Breuker, *Memory Versus Search in Games*, Ph.D. dissertation, University of Maastricht, 1998.

[4] J. Culberson and J. Schaeffer, 'Searching with pattern databases', in *Canadian Conference on AI*, pp. 402–416, (1996).

[5] S. Edelkamp, 'Planning with pattern databases', in *Proceedings of the 6th European Conference on Planning (ECP-01)*, pp. 13–34, (2001).

[6] J. Schaeffer *et al.*, 'Checkers is solved', *Science*, **317**(5844), 1518–1522, (2007).

[7] T. Fawcett and P. Utgoff, 'Automatic feature generation for problem solving systems', in *ICML*, pp. 144–153, (1992).

[8] A. Felner, U. Zahavi, J. Schaeffer, and R. Holte, 'Dual lookups in pattern databases', in *IJCAI*, pp. 103–108, (2005).

[9] M. Genesereth, N. Love, and B. Pell, 'General game playing: Overview of the AAAI competition', *AI Magazine*, **26**, 62–72, (2005).

[10] F h. Hsu, *Behind Deep Blue*, Princeton University Press, 2002.

[11] Alistair Hutton, *Developing Computer Opponents for Chinese Checkers*, Master's thesis, University of Glasgow, 2001.

[12] R. Hyatt, 2008. `http://www.craftychess.com/`.

[13] R. Korf and A. Felner, 'Disjoint pattern database heuristics', *Artificial Intelligence*, **134**, 9–22, (2002).

[14] Arsen Kostenko, *Calculating End Game Databases for General Game Playing*, Master's thesis, Fakultat Informatik, Technische Universitat Dresden, 2007.

[15] G. Kuhlmann and P. Stone, 'Automatic heuristic construction for general game playing', in *AAAI*, pp. 1457–1462, (2006).

[16] J. Schaeffer, *One Jump Ahead*, Springer-Verlag, 1997.

[17] G. Tesauro, 'Temporal difference learning and TD-Gammon', *CACM*, **38**(3), 58–68, (1995).

[18] K. Thompson, 'Retrograde analysis of certain endgames', *Journal of the International Computer Chess Association*, **9**(3), 131–139, (1986).

[19] Paula Ulfhake, *A Chinese Checkers-playing program*, Master's thesis, Department of Information Technology Lund University, 2000.