Extending the Applicability of Pattern and Endgame Databases

Mehdi Samadi, Fatemeh Torabi Asr, Jonathan Schaeffer, and Zohreh Azimifar

***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 (single-agent domains or puzzles), pattern databases have been effective at improving the quality of the heuristic values used in a search. This paper introduces new ways to extend the utility of pattern and endgame databases. Through the use of abstraction: 1) single-agent pattern databases can be applied to two- or more-player games; knowledge of the capabilities of one player (being oblivious to the opponent) can be an effective evaluation function for a class of game domains, and 2) endgame database positions can be viewed as an abstraction of more complicated positions; database lookups can be used as evaluation function features. These ideas are illustrated using the games of *Chinese Checkers*, *Chess*, and *Thief and Police*. 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 (GGP) programs. For two- or more-player applications where pattern and/or endgame databases can be built, abstraction can be used to automatically construct an evaluation function.**

***Index Terms*—Board games, *Chess*, game-tree search.**

I. 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)* [17], *Chinook (Checkers)* [28], *TD-Gammon (Backgammon)* [35], *Logistello (Othello)* [5], and *Maven (Scrabble)* [31]. Research into two-player games is one of the most visible accomplishments in AI 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 deterministic); usually the effort is invested in fine tuning the search for high performance. The knowledge component varies significantly from domain to domain. All of the above systems required expert input, especially the *Deep Blue* and *Chinook* programs.

Developing these high-performance programs required substantial effort over many years. In all cases, a major commitment had to be made to develop 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. Linear regression (as in *Logistello*), temporal difference learning (as in *TD-Gammon*), and manual (as in *Deep Blue* and *Chinook*) are popular techniques for determining the weights. Usually the weighted features are summed up to form the assessment. The above method has proven to be effective at building high-performance evaluation functions, albeit labor intensive. However, it 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 the Association for the Advancement of Artificial Intelligence (AAAI) conference has made the community more aware of the need for general-purpose solutions rather than custom solutions.

Most high-performance game-playing programs are computer intensive and benefit from faster and/or more central processing units (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 [4]. 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 (the *Chinook Checkers* program being an extreme example [29]). 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 puzzle [12], Rubik's cube [14], planning [9], and multiple sequence alignment [23]. 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). Pattern 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.

Manuscript received January 14, 2009; revised March 19, 2009; accepted March 19, 2009. First published April 07, 2009; current version published May 01, 2009.

M. Samadi and J. Schaeffer are with the Department of Computing Science, University of Alberta, Edmonton, AB T6G 2E8 Canada (e-mail: msamadi@cs.ualberta.ca; jonathan@cs.ualberta.ca).

F. T. Asr and Z. Azimifar are with the Department of Computer Science and Engineering, Shiraz University, Shiraz, Fars 71344, Iran (e-mail: torabi@cse.shirazu.ac.ir; azimifar@cse.shirazu.ac.ir).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2009.2019632

The main theme of this paper is to investigate and propose a new approach to using endgame and pattern databases to assist in automating the construction of an evaluation function for two- or more-player games. The key idea is to extend the benefits of these 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- or more-player games.
2) Pattern-database-based evaluation functions are shown to produce state-of-the-art play in the games of *Chinese Checkers* (ten pieces a side) and *Thief and Police*. Experimental results on *Chinese Checkers* show that against a baseline program containing the state-of-the-art evaluation function enhancements, the pattern-database-based program scores 68%–79% of the possible points. Similar results are also reported for *Thief and Police*.
3) Abstraction is used to treat the multiplayer simultaneous-move *Thief and Police* game as a single-agent domain. The experimental results show that the abstraction-based evaluation function is effective in the five-player game, significantly outperforming the Manhattan-distance-based evaluation function.
4) Abstraction is used to extend endgame databases for constructing evaluation functions for a class of two-player games.
5) *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%–80% of the possible points.

Abstraction is a key to extending the utility of 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 play.

Preliminary results of this research were previously published [27].

II. Background and Related Work

In this section, we briefly overview general background of this research and describe related works on game abstraction.

A. Endgame Databases

Endgame databases have been in use for two-player perfect-information games for almost 30 years. An endgame database contains all possible configurations for a few pieces on the board. For each possible arrangement of the pieces, the database records the perfect-play result (win, loss, draw) and possible additional information (e.g., the minimum number of

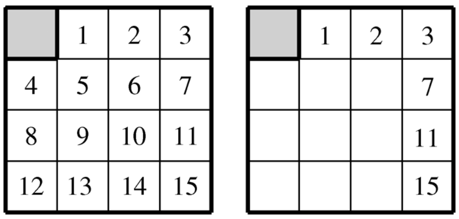


Fig. 1. The 4 × 4 sliding-tile puzzle. (a) Goal state. (b) An abstraction.

moves to win if the position is winning or the maximal number of moves to postpone a loss if the position is losing). These databases can be constructed using retrograde analysis [33], [36]. One can enumerate all positions with one piece on the board, and record which positions are wins, losses, or draws. These results can be backed up to solve all positions with two pieces on the board, and so on.

The databases are beneficial because of reducing the search effort (replacing a subtree by a database lookup) and introducing accurate results into the search. Instead of using a heuristic to evaluate an endgame position (with the associated error), a game-playing program can directly query from the database (i.e., perfect information). The limitation, however, is that each position in the database is applicable to a single position in the search space.

Chess was the original application domain that used endgame databases, where databases for all positions with six or fewer pieces have been built [3]. Endgame databases were essential for solving the game of *Checkers*, where all positions with ten or fewer pieces have been computed [29].

B. Pattern Databases

Pattern databases (PDB) also use retrograde analysis, in this case to optimally solve simplified versions of a single-agent state space [8]. A PDB is constructed by abstracting the problem domain (e.g., only considering a subset of the features) and solving the simplified version of the given problem. A PDB stores the cost of an optimal solution for each instance of an abstraction applied to the original problem. The stored costs are then used as an admissible heuristic for the original problem domain. PDBs have been used as effective lower bounds for solving problems in domains such as combinatorial puzzles [7], [8], [12], [13], [20], [21], multiple sequence alignment [10], [26], [30], [38], vertex cover [12], and planning problems [9].

Consider the well-known 4 × 4 sliding-tile puzzle, a classic AI benchmark. The goal state is shown in Fig. 1(a). A subgoal can be defined as a permutation of a subset of tiles, such as those in Fig. 1(b). For any initial state, the task is to move all the tiles included by this subset into their goal position. Clearly, the minimum number of moves needed to fulfill this subgoal is a lower bound on the optimal solution to the entire puzzle. For each state, the minimum number of moves required to reach the goal is recorded in a lookup table (a pattern database). A simple breadth-first search (BFS), moving backward from the goal state, can be used to fill the table.

Pattern databases have led to a significant improvement in the heuristic values used for some domains, leading to a large reduction in the time required to solve problem instances.

C. Related Work on Abstraction for Games

Using and developing abstraction techniques to improve the performance of search algorithms employed by multiagent systems has been studied by numerous researchers. Abstraction has been used for playing both imperfect-information games (e.g., *Poker* [2], [16], [32]) and perfect-information games (e.g., *Go* [6]). Here, we review recent studies on the use of abstraction applied to games.

Abstraction is used to treat multiple instances of similar subproblems as a single one. The simplest case is where multiple states are isomorphic to each other, allowing each of them to be treated identically. Beyond this, there are nonisomorphic states in the game tree that are similar enough such that the appropriate strategy is essentially identical. For example, in card games, the smallest card in a hand being a deuce instead of a trey may have no bearing on the outcome. This is analogous to the approach used in *Bridge* by Ginsberg for his partition search algorithm [16]. He defined equivalence classes for the smallest cards of each suit in a *Bridge* hand. Shi and Littman used this idea to produce near-optimal solutions for a scaled-down version of *Texas Hold'em Poker* [32]. Billings *et al.* proposed a new lossy abstraction technique for two-player *Poker* to reduce the large search space [2]. They showed that their abstraction technique can produce viable "pseudo-optimal" strategies for this game. The resulting *Poker*-playing programs demonstrated a tremendous improvement in performance, and new advances in this area have produced world-class play [39].

Cazenave used abstraction to design admissible heuristics for two-player games [6]. An admissible heuristic gives the minimum number of moves required to win the game. He relaxed the rules of a game, and used the moves in the relaxed game to play in a real situation. Experimental results on *Atari-Go*, a simplified version of the game *Go*, were presented. Although he discussed how to generalize the technique for other games, this is still an application-specific technique which needs human knowledge. It only works in domains where an admissible heuristic can be defined in the abstract space.

In this paper, we propose a new approach to using abstraction in a class of perfect-information games with two or more players. The method is also extendible to imperfect-information games.

III. Abstraction

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.

Definition 1 (Abstraction Transformation): An abstraction transformation $\phi : S \rightarrow S'$ maps:

1) states $u \in S$ to states $\phi(u) \in S'$;

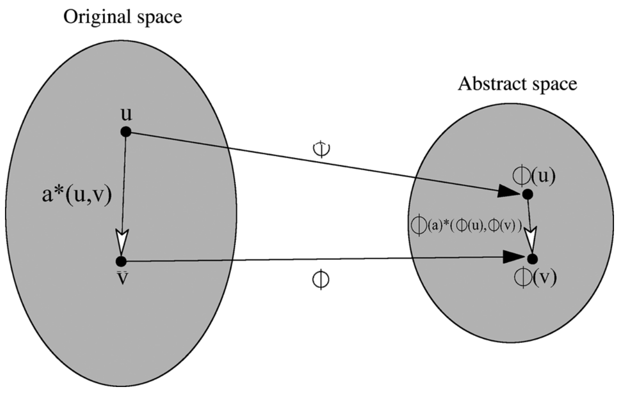


Fig. 2. Original states and edges mapped to an abstract space.

2) actions $a \in S$ to actions $\phi(a) \in S'$.

This is illustrated in Fig. 2, where a^* represents zero or more actions.

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 that contains kings 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 exist in the abstract state.

Definition 2 (Homomorphism): An abstraction transformation ϕ is a homomorphism transformation if for all series of actions that transform state u to state v in S, there exists a corresponding transformation of $\phi(u)$ to $\phi(v)$ in S'. This is illustrated in Fig. 2.

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 the 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 ϕ_i as $h_{\phi_i}(u, v)$. For example, for an endgame or pattern database, v being set to a goal state means that $h_{\phi_i}(u, v)$ is the minimum number of moves needed to achieve the goal.

Using offline 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 minimum number of moves to win

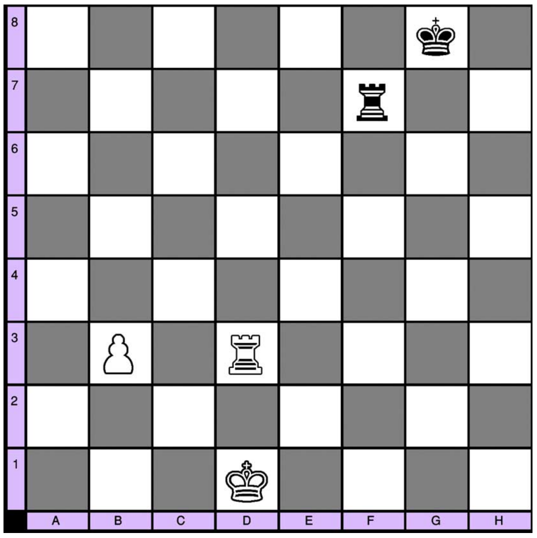


Fig. 3. An example of KRPKR position, white to move and win in 37 moves.

(maximal moves to postpone losing) is recorded. This is the standard way that these databases are constructed.

Given a problem instance to solve, during the search, 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 $h = \langle h_{\phi_1}, h_{\phi_2}, \ldots, h_{\phi_n} \rangle$. The evaluation function value for state p is calculated as a function of h. There are popular ways to combine h_{ϕ_i} values such as temporal difference learning to linearly combine the heuristic values [1], and neural networks to achieve nonlinear relations [35].

For example, consider evaluating a position p in the KRPKR *Chess* endgame. In this case, the abstracted states could come from the databases KRPK, KRKR, KRK, and KPK (all being proper subsets of the pieces on the original board). First, for each abstraction i, the abstract state is computed and the heuristic 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 is looked up in the KRKR database; etc. The heuristic value for p could be, for example, the sum of the four abstraction scores.

In the following sections, we explore using abstraction to apply pattern database and endgame database technology to two-player *Chinese Checkers*, *Chess* endgame positions, two-player *Thief and Police*, and multiplayer simultaneous-move *Thief and Police*. Most of the abstractions used in *Chess* are not homomorphisms. For example, Fig. 3 shows a position in the KRPKR endgame in which the white player has an advantage and can win the game. If we construct an abstraction by removing white's pawn from the board, the result of the abstract game is a draw which is different than the original space. Unlike *Chess*, *Chinese Checkers* and some versions of *Thief and Police* possess the homomorphism property.

All the experiments in this paper were performed with each move decision being considered to a fixed search depth. The alternative would be to report results for a fixed time per move. The latter would have been limiting, given that we used multiple

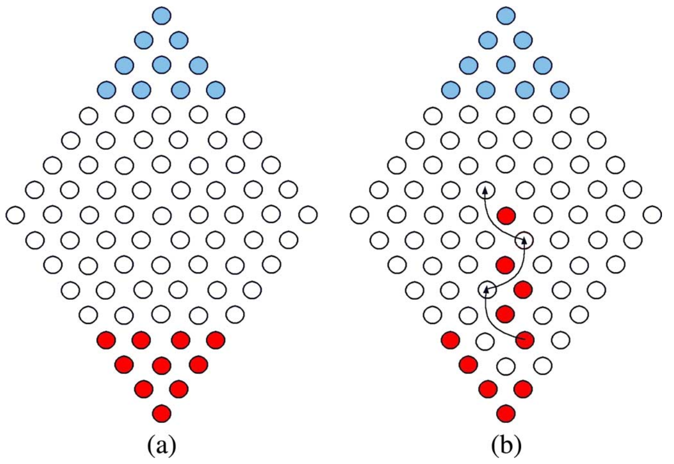


Fig. 4. *Chinese Checkers*. (a) A 9×9 board for two players. (b) Sample jump over three marbles in one turn.

machines with different speeds. The former is preferred, despite the (small) differences in the execution cost of different evaluation functions. For our experiments, it turned out that the cost of an evaluation was dwarfed by the overhead of the game-playing program (in particular, move generation). The biggest difference in program execution times as a result of a more computationally expensive evaluation function was observed to be 8%.

IV. Chinese Checkers

Chinese Checkers is a two- to six-player game played on a star-shaped board with the squares hexagonally connected. The game objective is to move all allied pieces (or marbles) from the player's home zone (typically ten squares) to the opposite side of the board (the opponent's home zone). A player moves one marble at each turn. A marble can move by rolling into one of its six adjacent positions (if empty) 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*). None of the pieces are removed from the board. In general, to reach the goal in the shortest possible time, the player should jump his pieces towards the opponent's home zone. A 9×9 board for two-player *Chinese Checkers* is shown in Fig. 4(a) and a single jump with length three over adjacent marbles is illustrated in Fig. 4(b).

Here we limit our attention to two-player *Chinese Checkers*, although the results presented scale well to more players (not included in this paper). Due to the characteristics of *Chinese Checkers*, two kinds of abstraction might be considered. Given N pieces on each side of the original game, the abstraction can be as follows:

- Abstraction 1 (AB1): playing $K \leq N$ white pieces against $L \leq N$ black pieces (an endgame database of $K + L$ pieces);
- Abstraction 2 (AB2): playing $K \leq N$ white (black) pieces into the opponent's home zone (a pattern database of K pieces).

As more position features are exploited by the set of abstractions, combining the abstraction heuristics will likely be more beneficial in the original problem space. Both of the above abstractions have the homomorphism property. In AB1, a subset of pieces for both players (e.g., a three-piece versus two-piece game) is considered and the minimum number of moves to win (maximum number of moves to lose) is used. The second abstraction AB2 ignores all the opponent's pieces, using the number of moves required to get all of a player's pieces into the opponent's zone. This value is just a heuristic estimate (not a bound), since it does not take into account the likelihood of jumping over the opponent's pieces (which precludes it from being a lower bound) and the 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 baseline for comparison is a *Chinese Checkers* program (ten pieces per side) with all the state-of-the-art enhancements [18], [37]. The evaluation function is based on the Manhattan distance for each player's pieces to reach the goal area. Recent studies have improved this simple heuristic by adding extra properties: 1) curved board model, incremental evaluation, left-behind marbles [37]; and 2) learning [18]. All of these features have been implemented in our baseline program.

Experiments included the baseline program playing against a program using a pattern-database- or endgame-database-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 by each side from the starting position have been made). A win is worth two points, a draw one, and a loss zero. The result of a match is expressed as the percent of the number of possible points obtained.

Experiments are reported for search depths of 1, 3, 5, 7, and 9 ply for small games (six-versus-six pieces) and search depths of 1, 3, and 5 for large games (ten-versus-ten pieces). Other search depth results are similar. The branching factor in the middle game of *Chinese Checkers* is roughly 50–70. Move generation is expensive because of the combination of jumps for each side. This affects the program's speed (positions evaluated per second), limiting the search depth that can be achieved in a reasonable amount of time. The average response time for a search depth of 7 in the middle game is more than 10 min per move (18 h per game).

We report the results of using AB1 (i.e., endgame databases) and AB2 (i.e., pattern databases). To study the effect of different abstraction methods on various aspects of the problem domain, a number of experiments were performed using simplified domains (smaller sized boards and/or fewer pieces).

A. Endgame Databases

The AB1 (endgame database) is the abstraction used in the evaluation function: $K \leq N$ white (black) pieces playing against $L \leq N$ black (white) pieces. The cost used in this abstraction is the minimum number of moves to win the game when the opponent plays his best strategy. This abstraction has a large state space; on the full 81-square board, the endgame database of three-versus-three pieces requires roughly 6.1 GB. In contrast, a pattern database size for six pieces (of the same side) needs roughly 309 MB, 20 times less space. In this section, only the experimental results of using the endgame databases on small boards are presented.

Two sets of experiments were performed. The first experiment uses only three pieces per player. One player uses the endgame database as an evaluation function and the other player uses the state-of-the-art heuristic. As the endgame databases cover all the pieces on the board, the white player has perfect information and will play the best moves. This experiment was done to show the capability of the abstraction in the best case (when the abstraction covers all the pieces on the board). In the second experiment, each side has six pieces; the abstraction does not cover all the pieces on the board. The pieces are partitioned into two disjoint sets, each containing three white and three black pieces. The evaluation function is the sum of two lookups into the endgame database for each group of three-versus-three pieces.

TABLE I
CHINESE CHECKERS: PERFORMANCE OF EDB(6) LOOKUP (SMALL BOARDS)

No. of pieces per side	**Win % depth 1**	**Win % depth 3**	**Win % depth 5**	**Win % depth 7**	**Win % depth 9**
6×6 board size					
3-3	100	97	97	97	95
6-6	54	60	63	63	61
7×7 board size					
3-3	100	100	100	93	96
6-6	61	57	62	60	60

Table I shows the results of using endgame databases (EDB). These experiments were performed on 6×6 and 7×7 *Chinese Checkers* boards, with either three or six pieces per player. An endgame database of three pieces per player was used (EDB(6)), containing the win/loss/draw result and the minimum number of moves needed to achieve that result. The sizes of the EDB(6) databases for 6×6 and 7×7 boards are 38 and 266 MB, respectively. The first column of Table I shows the number of pieces on the board for each side. Columns 2–6 show the winning percent for the EDB-based program versus the state-of-the-art program given different depths for the search tree. For each size of board (6×6 and 7×7), the three-versus-three and six-versus-six-piece results are given.

The results for the three-versus-three-piece game are predictably good on both board sizes (over 93% wins). The reason is obvious; the endgame database contains all the pieces on the board and produces the best playing strategy for the side using that heuristic. When playing the six-versus-six-piece game, the results are good (averaging 60%). Here the three-versus-three-piece endgame database is used as an abstraction; the evaluation function is the sum of two lookups into this database (to cover all the pieces on the board). Clearly something gets lost when the interactions between the two subsets of pieces are not properly considered.

There was one application-specific complication when building the three-versus-three-piece endgame database: most of the database entries were draws. The reason for this is that when a player cannot find a winning strategy, then the player prefers to leave one of the pieces in their initial area to prevent the opponent from winning. In these cases, the draw is the best strategy, preventing the player from losing the game. To solve this problem, small changes had to be made to the rules used to build these endgame database.

1) The player must play only those moves which decrease its Manhattan distance to the goal area (the player must move towards the goal, if possible).

TABLE II
CHINESE CHECKERS: PERFORMANCE OF PDB(6) ABSTRACTION (SIX-VERSUS-SIX-PIECE GAME)

Board size	Win % depth 1	Win % depth 3	Win % depth 5	Win % depth 7	Win % depth 9
7×7	82	86	82	80	80
8×8	76	80	84	94	80
9×9	96	94	82	94	92

TABLE III
CHINESE CHECKERS: PERFORMANCE OF PDB(3) ABSTRACTION (SIX-VERSUS-SIX-PIECE GAME)

Board size	Win % depth 1	Win % depth 3	Win % depth 5	Win % depth 7	Win % depth 9
7×7	52	74	70	66	77
8×8	60	64	68	64	56
9×9	62	76	66	76	72

2) If the player cannot play any moves to satisfy 1), then he prefers moves only to the right or to the left positions.
3) If scenarios 1) and 2) do not apply, then backward moves are allowed.

Now there are no draw entries. Nevertheless, having to do this is unsatisfactory.

In the next section, we present and elaborate on the results of using a PDB as an abstraction. It will be shown that during the game K cooperating pieces will result in longer jumps (hence, fewer moves to reach the goal) than K adversarial pieces. Although AB1 (endgame databases) seems to better reflect the domain, AB2 (pattern databases) gives better experimental results.

B. Pattern Databases

This section presents the second abstraction approach (AB2), in which no opponent piece is taken into consideration. This abstraction is first evaluated using different board sizes with fewer pieces, and then it is applied to the full-sized board with ten pieces.

1) Small Games: This section evaluates and compares the effectiveness of two different abstraction techniques:

- when all the pieces for one side are included in the abstraction;
- when a subset of the pieces for one side is included in the abstraction.

Clearly the first approach takes into account all pieces and if sufficient memory space is available for the PDB this approach is preferred. For the full game of *Chinese Checkers*, this is not possible—the PDB is 1791 GB. The alternative is to use a smaller PDB (for fewer pieces) and use the sum of multiple PDB lookups. For example, for a game with six pieces per side, one could build an evaluation function using a three-piece PDB (PDB(3)). The white player's pieces can be divided into disjoint subsets of three; each set of three is looked up in the PDB and the sum is used as the evaluation.

Table II presents the results of using a pattern-database-based evaluation function for different sized boards when each side has six pieces. The experiments were conducted using a six-piece PDB (PDB(6)). The first column in the table shows the board size. The next columns report on the winning percentage when the depth varies.

The results show that when the PDB covers all of the pieces, the PDB-based evaluation is strong, scoring at least 76% of the points in all the scenarios. It is interesting to note that the PDB performance is increased with larger board sizes. For example, the winning percentage increases from 80% for the 6×6 board to 94% for the 9×9 board (depth 7). The reason is that the larger board decreases the interaction of the white and black pieces resulting in the PDB returning a more accurate heuristic value.

Table III shows the results of using a PDB that covers a subset of the pieces on the board (PDB(3)). The sum of two lookups into the PDB, each for one set of three out of the six pieces, is used as the evaluation function. Even with this small PDB, this abstraction technique outperforms the state-of-the-art heuristic evaluation function for *Chinese Checkers*.

2) Full Game (Ten-Versus-Ten Pieces): In this section, we report the results for three interesting heuristic evaluation functions on the full game (ten-versus-ten pieces played on a 9×9 board). For the following abstractions, the pieces were labeled 1 to 10 in a right-to-left, bottom-up manner. The abstractions used are shown in Fig. 5.

— PDB(4): Four-piece pattern database 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. Note the overlap; pieces 4 and 7 are included twice so that a single database can be used to build an evaluation that includes all ten pieces on the board [Fig. 5(a)].
— PDB(6): Six-piece pattern database with the goal defined as the top six squares in the opponent's home zone. Two database lookups were used to cover all ten pieces: pieces 1–6 and 5–10. Again, two pieces are counted twice in an evaluation (pieces 5 and 6) [Fig. 5(b)].
— PDB(6+4): A probe from the six-piece PDB is added to a probe from the four-piece PDB. Two abstractions (two lookups) were used to cover all ten pieces: pieces 1–4 from PDB(4) and pieces 5–10 from PDB(6). In this case, PDB(4) is defined with the goal of moving all four pieces into the first four squares (1–4) in the goal area [this differs from PDB(4) above]. Thus the six-piece and four-piece databases are trying to move their pieces into different parts of the goal area [Fig. 5(c)].

The weighting of each probe is a simplistic linear combination of the abstraction heuristic values. The weights were tuned by hand based on the performance of the evaluation function on a set of random positions (test data). The resulting evaluation function was used on a different set of positions for the reported experiments (training data).

Table IV presents the results achieved using the above evaluation functions for search depths of 1, 3, and 5 (the experiments take too long to run for larger depths). PDB(6+4) has the best performance, winning about 66%-79% of the games against the baseline program. Perhaps surprisingly, for depth 5, PDB(6) performs better than PDB(6+4). One would expect it to be the other way around since PDB(6+4) implicitly contains more knowledge of the piece's 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

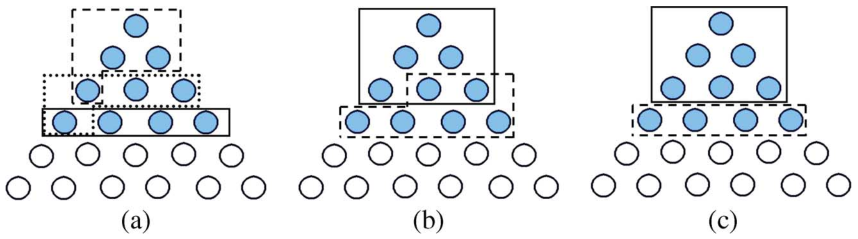


Fig. 5. *Chinese Checkers*: PDB abstractions (9 × 9 board). (a) PDB(4). (b) PDB(6). (c) PDB(6+4).

TABLE IV
CHINESE CHECKERS: PERFORMANCE OF PDB ABSTRACTIONS (9 × 9 BOARD)

Depth	PDB(4)	PDB(6)	PDB(6+4)
1	65	67	79
3	52	61	68
5	62	78	66

the probability that it can be realized, given that there are other pieces on the board. Hence, we conclude that a large PDB may not be as accurate as a small PDB.

Note that not only does PDB(6+4) have a better winning percentage than PDB(4), but also this result was achieved with one less database lookup per evaluation. For some applications, this may represent an important performance consideration.

The results reported here do not necessarily represent the best result that could be attained. 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 and pattern databases can be used in a new, nontraditional way.

V. CHESS

For *Chess*, 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 try using a pattern database for *Chess*, however, we expect that an evaluation function learning algorithm (such as the popular TDLeaf algorithm [1]) would 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.

In our experiments, we used the four- and five-piece endgame databases. Note that the state-of-the-art in endgame database construction is six pieces [3]. 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 1 GB 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.

For the 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 [19]. It has competed in numerous World Computer Chess Championships, often placing near the top of the standings. Table V shows the impact of two parameters on performance: the endgame database size and the search depth. The table gives results for three representative seven-piece endgames. The second column gives the endgame, the third and fourth columns give the win percentage for search depths of seven and nine respectively, and the last column shows the abstractions used. For each depth, the first three lines show the results when three- and four-piece databases are used 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 evaluation. 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 7 and 4% better at depth 9). Clearly, these databases are closer to the original position (i.e., less abstract) and hence are

TABLE V
CHESS: EDB(3), EDB(4), AND EDB(5) ABSTRACTIONS (SEVEN-PIECE ENDGAMES)

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

TABLE VI
CHESS: EDB(5) ABSTRACTIONS (EIGHT-PIECE ENDGAMES)

Endgame	Win % depth=7	Win % depth=9	Abstractions Used
KQP–KRNP	64	64	KQKRP, KQKNP, KPKRN , KQKRN
KRRPP–KQR	76	76	KQKRP, KQKNP, KPKRN
KRPP–KRN	60	64	KRPKN, KPPKR, KPKRN
KQP–KNNPP	76	72	KPKNN, KQKNN, KQKNP
KQP–KRBPP	64	62	KPKNN, KQKNN, KQKNP

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 differentiation 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.

Table VI 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%–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.

The endgame database evaluation function idea has also been applied to *Checkers* endgames. Against *Chinook*, the world man-machine *Checkers* champion [28], the database-based evaluation function played to a statistical draw. *Chinook* is a very strong program and *Checkers* is inherently a drawish game, so a new evaluation function would not be expected to yield much of an advantage. Note that the *Chinook* endgame evaluation function was developed and manually tuned over many years. In contrast, the database-based evaluation function consisted of lookups in already existing data. That comparable performance achieved with a small amount of programming is an impressive endorsement of the new idea.

VI. THIEF AND POLICE GAME

Thief and Police is played on a maze of size $m \times n$ with squares that are either empty or blocked. The maze configuration includes K policemen and one thief, each of which can move horizontally or vertically one square to an empty square.

A single move by the police side consists of simultaneous moves by all the policemen. The police player wins the game whenever he catches the thief by surrounding him (occupying all adjacent empty squares). The thief wins if he reaches an exit point defined on the board. The *Thief and Police* game is, in effect, a simplified version of a scenario occurring in numerous commercial games. A popular game *Scotland Yard* is based on this theme.

Two-player *Thief and Police* can be abstracted to a single-agent domain by fixing all but one of the players on the board. This can be used for strong play in a simplified two-player version of the game. We then show that the PDBs can be built to be an effective evaluation function in the multiplayer domain.

A. Abstraction to Single-Agent Domain

Two PDBs are needed: one for the policemen and one for the thief. For the police, the state space can be abstracted into a single-agent domain by using only $K' < K$ of the police players on the board. In this approach, the abstraction is defined as if from each player's viewpoint with the position of the other players being fixed. Accordingly, one can construct the PDB for the police player in which each policeman—in cooperation with his colleagues—tries to minimize his path to catch the thief. The PDB will include all configurations of putting K' policemen on the maze, while the thief position is assumed fixed, giving the shortest path to catch the thief. For the thief, the positions of all policemen are assumed to be fixed. The thief PDB will give the fewest moves to reach the goal (the exit point defined on the board).

To create the police PDB, the multiple-goal PDB construction method is used [22]. First, a queue consisting of all goal

TABLE VII
THIEF AND POLICE: 20 $\times$ 20 BOARDS OF FIVE DIFFERENT CONFIGURATIONS

Maze number	Win % depth 5	Win % depth 7	Win % depth 9	Win % depth 11
1	74	71	70	69
2	81	81	79	77
3	63	73	79	70
4	70	67	70	56
5	77	79	77	76

TABLE VIII
THIEF AND POLICE: 60 $\times$ 60 BOARDS OF FIVE DIFFERENT CONFIGURATIONS

Maze number	Win % depth 5	Win % depth 7	Win % depth 9	Win % depth 11
1	64	67	66	63
2	67	66	64	64
3	72	69	68	65
4	68	68	65	64
5	74	72	72	68

TABLE IX
SIMULTANEOUS-MOVE THIEF AND POLICE: 20 $\times$ 20 BOARDS OF FIVE DIFFERENT CONFIGURATIONS

Maze number	Win % depth 5	Win % depth 7	Win % depth 9	Win % depth 11
1	72	68	68	67
2	76	74	73	71
3	60	66	66	62
4	67	66	64	64
5	68	67	64	64

states with depth zero is initialized. Then, from each initial state defined in the queue a breadth-first search is run to find all reachable states in the state space. All states reached by the BFS algorithm plus their depth value are stored as a PDB table. Finally, the states not reached by the BFS algorithm are assigned an "unreachable" label along with a depth value that is greater than the largest depth value appearing in the table.

During the actual search of a game state, to evaluate a police move, we divide the police side into groups of K' players: $K = m \times K' + n$. Thus, there are m groups of K' players and possibly one group with less than K' players. Then, $m + 1$ lookups are done into the police PDB, one for each group of policemen. The results are linearly combined. For the thief side, a single lookup is done into the thief PDB.

To evaluate the above abstraction, consider the *Thief and Police* game variant with four policemen, a thief, and two different board sizes:

1) Small-sized maze (20 $\times$ 20) with all four police pieces included in a single abstraction (i.e., $K' = K = 4$);
2) large-sized maze (60 $\times$ 60) with only two policemen being considered in the abstracted domain (i.e., $2 = K' < K = 4$).

A 20 $\times$ 20 board contains about 150 empty squares. With all four police locations being considered, the PDB size is about 2.9 GB. However, for a 60 $\times$ 60 board with many more empty squares (roughly 1610 empty locations), the PDB will be unacceptably large unless fewer policemen are considered. Hence, the PDB abstraction includes only two policemen (1.94 GB). Unfortunately, with only two policemen allowed to move in the PDB, in general, it is not possible to catch the thief (for example, two policemen cannot surround a thief that is adjacent to four empty squares). Therefore, for the construction of this PDB, the goal is restated as "the thief is caught if two policemen surround him, regardless of the number of empty squares around the thief."

In this study, we considered five different mazes, each with 50 random starting configurations of the thief and police pieces. The mazes were generated using the hierarchical open graph (HOG) framework [34]. For each configuration, two games were performed, one with the police against a baseline player and one with the thief against a baseline player. The baseline players use Manhattan distance as their evaluation function (there is no strong evaluation function for this game in the literature). As before, two points are assigned to the winner, one point each for a draw, and zero points for losing.

Table VII presents the results for 20 $\times$ 20 boards with search depths of 5–11. In all cases, the PDB-based play significantly outperforms the Manhattan-distance-based evaluation function. Note that with increased depth, the Manhattan distance results improve. This is a consequence of the deeper searches seeing far enough to correct the errors of a shallow search.

Table VIII presents the results for 60 $\times$ 60 boards with search depths of 5–11. The boards averaged 1610 empty squares. The original space considers four policemen while the constructed PDB only includes combinations of two police pieces. To evaluate a state, the police pieces are divided into two sets of size two. Two PDB lookups are done and the results are linearly combined.

Again, the PDB-based evaluations outperform the Manhattan-distance-based evaluations. Comparison between Table VII and VIII suggests that abstraction was more successful for small-sized mazes. However, this largely reflects the abstraction: the larger mazes could only consider subsets of the police.

VII. SIMULTANEOUS-MOVE THIEF AND POLICE GAME

In this section, abstraction is applied to the multiplayer simultaneous-move *Thief and Police* game. At each turn all the policemen move simultaneously. This dramatically increases the branching factor of the search tree, limiting the possible depth of search that is possible. A simultaneous-move game with K police and one thief can be treated as a cooperative multiplayer game with $K + 1$ players.

As before, while constructing the thief PDB, it is assumed that the positions of all the police players are fixed. The police PDB considers $K' < K$ police with a fixed thief player and each entry contains the solution cost for the K' police players to catch the thief by moving simultaneously.

Table IX presents the results for 20 $\times$ 20 simultaneous-move *Thief and Police* game, and Table X presents the results for a 60 $\times$ 60 board. The opponent players use Manhattan distance as their evaluation function. In both cases, the results are similar to those reported in the previous section. Again, a PDB-based evaluation function outperforms the baseline evaluation. Comparison between the results of simple and simultaneous-move *Thief and Police* shows that the winning percentage of the PDB-based evaluation function is slightly lower in the simultaneous-move game. This is to be expected, given that abstracting the game as a single-agent search is "closer" to the two-player game than to the multiplayer game.

TABLE X
SIMULTANEOUS-MOVE THIEF AND POLICE: 60 × 60 BOARDS OF FIVE DIFFERENT CONFIGURATIONS

Maze number	**Win % depth 5**	**Win % depth 7**	**Win % depth 9**	**Win % depth 11**
1	61	60	60	58
2	62	64	61	60
3	58	57	57	55
4	65	63	62	60
5	66	67	64	59

VIII. CONCLUSION

The research presented in this paper is a step towards increasing the advantages of precomputed lookup tables for the larger class of multiagent problem domains. The main contribution of this research was to show that the idea of abstraction can be used to extend the benefits of precomputed databases for use in new ways in building an effective 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 to create an evaluation function.

The abstraction technique was applied to three different domains: *Chinese Checkers*, *Chess*, and the *Thief and Police* game. For all these domains, we showed that even small databases are useful to produce strong game play. In *Chinese Checkers*, for all the configurations, the PDB-based evaluation function outperformed the state-of-the-art evaluation function. The *Chess* benchmark showed that an evaluation function based on endgame databases can produce strong game play, as compared to a strong baseline program. Similarly, good results were demonstrated for both the simple and the simultaneous-move versions of *Thief and Police*.

IX. FUTURE WORK

Since 2005, there has been interest in the AI community in building a general game-playing (GGP) program [15]. 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 [11]. It is still early days for developing GGP programs; the recent technique for developing evaluation functions uses several well-known heuristics that have been proven to be effective in a variety of games, and then tests them to see if they are applicable to the current domain [25].

Research has already been done in identifying GGP domains for which databases can be built [24]. For those domains, abstraction is a promising way to build an effective evaluation function. An automated system has been developed to build pattern databases for planning domains using a bin-packing algorithm to select the appropriate symbolic variables for a pattern database [9]. A similar approach can be used to automatically select variables in GGP to build endgame/pattern databases. It remains an open problem how to automate the discovery and learning of an effective abstraction and evaluation function for an arbitrary game.

In this paper, we presented results of different abstractions for several games. We showed that the selection of the abstraction has a direct influence on the performance of the evaluation function. Currently, choosing the best abstraction for a game requires the use of problem-specific knowledge. An important future direction of this research is to automate the process of choosing good abstractions. Our experience with the games in this paper has given us insights into techniques that we anticipate will be general over a wide class of games.

REFERENCES

[1] J. Baxter, A. Tridgell, and L. Weaver, "Learning to play chess using temporal differences," *Mach. Learn.*, vol. 40, no. 3, pp. 243–263, 2000.

[2] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron, "Approximating game-theoretic optimal strategies for full-scale poker," in *Proc. Int. Joint Conf. Artif. Intell.*, 2003, pp. 661–668.

[3] E. Bleicher, "Web query for Nalimov endgame tablebases," 2008 [Online]. Available: http://k4it.de/index.php?topic=egtb&lang=en

[4] D. Breuker, "Memory versus search in games," Ph.D. dissertation, Dept. Comput. Sci., Univ. Maastricht, Maastricht, The Netherlands, 1998.

[5] M. Buro, "Improving heurisitic mini-max search by supervised learning," *Artif. Intell.*, vol. 134, no. 1–2, pp. 85–99, 2002.

[6] T. Cazenave, "Admissible moves in two-player games," in *Proc. Symp. Abstraction Reformulation Approximation*, 2002, pp. 52–63.

[7] J. Culberson and J. Schaeffer, "Efficiently searching the 15-puzzle," Dept. Comput. Sci., Univ. Alberta, Edmonton, AB, Canada, Tech. Rep., 1994.

[8] J. Culberson and J. Schaeffer, "Pattern databases," *Comput. Intell.*, vol. 14, no. 3, pp. 318–334, 1998.

[9] S. Edelkamp, "Planning with pattern databases," in *Proc. Eur. Conf. Planning*, 2001, pp. 13–34.

[10] S. Edelkamp and P. Kissmann, "Externalizing the multiple sequence alignment problem with affine gap costs," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2007, vol. 4467, pp. 444–447.

[11] T. Fawcett and P. Utgoff, "Automatic feature generation for problem solving systems," in *Proc. Int. Conf. Mach. Learn.*, 1992, pp. 144–153.

[12] A. Felner, R. Korf, and S. Hanan, "Additive pattern database heuristics," *J. Artif. Intell. Res.*, vol. 22, pp. 279–318, 2004.

[13] A. Felner, R. Meshulam, R. Holte, and R. Korf, "Compressing pattern databases," in *Proc. Nat. Conf. Artif. Intell.*, 2004, pp. 638–643.

[14] A. Felner, U. Zahavi, R. Holte, and J. Schaeffer, "Dual lookups in pattern databases," in *Proc. Int. Joint Conf. Artif. Intell.*, 2005, pp. 103–108.

[15] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the AAAI competition," *AI Mag.*, vol. 26, pp. 62–72, 2005.

[16] M. Ginsberg, "Partition search," in *Proc. Nat. Conf. Artif. Intell.*, 1996, pp. 228–233.

[17] F. Hsu*, Behind Deep Blue*. Princeton, NJ: Princeton Univ. Press, 2002.

[18] A. Hutton, "Developing computer opponents for Chinese Checkers," M.S. thesis, Dept. Comput. Sci., Univ. Glasgow, Glasgow, U.K., 2001.

[19] R. Hyatt, Welcome to the Crafty Chess page, 2008 [Online]. Available: http://www.craftychess.com/

[20] R. Korf, "Finding optimal solutions to Rubik's Cube using pattern databases," in *Proc. Nat. Conf. Artif. Intell.*, 1997, pp. 700–705.

[21] R. Korf and A. Felner, "Disjoint pattern database heuristics," *Artif. Intell.*, vol. 134, pp. 9–22, 2002.

[22] R. Korf and A. Felner, "Recent progress in heuristic search: A case study of the four-peg towers of Hanoi problem," in *Proc. Int. Joint Conf. Artif. Intell.*, 2007, pp. 2324–2329.

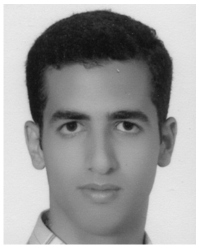
[23] R. Korf and W. Zhang, "Divide-and-conquer frontier search applied to optimal sequence alignment," in *Proc. Nat. Conf. Artif. Intell.*, Aug. 2000, pp. 910–916.

[24] A. Kostenko, "Calculating end game databases for general game playing," M.S. thesis, Fakultat Informatik, Technische Universitat Dresden, Dresden, Germany, 2007.

[25] G. Kuhlmann and P. Stone, "Automatic heuristic construction for general game playing," in *Proc. Nat. Conf. Artif. Intell.*, 2006, pp. 1457–1462.

[26] M. McNaughton, P. Lu, J. Schaeffer, and D. Szafron, "Memory efficient A° heuristics for multiple sequence alignment," in *Proc. Nat. Conf. Artif. Intell.*, 2002, pp. 737–743.

[27] M. Samadi, J. Schaeffer, F. T. Asr, M. Samar, and Z. Azimifar, "Using abstraction in two-player games," in *Proc. Eur. Conf. Artif. Intell.*, 2008, pp. 545–549.
[28] J. Schaeffer, *One Jump Ahead: Computer Perfection at Checkers*. New York: Springer-Verlag, 2009.
[29] J. Schaeffer, N. Burch, Y. Bjornsson, A. Kishimoto, M. Mueller, R. Lake, P. Lu, and S. Sutphen, "Checkers is solved," *Science*, vol. 317, no. 5844, pp. 1518–1522, 2007.
[30] S. Schroedl, "An improved search algorithm for optimal multiple-sequence alignment," *J. Artif. Intell. Res.*, vol. 23, pp. 587–623, 2005.
[31] B. Sheppard, "World-championship-caliber Scrabble," *Artif. Intell.*, vol. 134, no. 1–2, pp. 241–275, 2002.
[32] J. Shi and M. Littman, "Abstraction methods for game theoretic poker," in *Computers and Games*. New York: Springer-Verlag, 2002, pp. 333–345.
[33] T. Strohlein, "Untersuchungen ber kombinatorische Spiele," (in German) Ph.D. dissertation, Technische Universitat Munchen, Munchen, Germany, 1970, English translation: "Research on combinatorial games".
[34] N. Sturtevant, HOG, 2009 [Online]. Available: http://www.cs.ualberta.ca/~nathanst/hog.html
[35] G. Tesauro, "Temporal difference learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995.
[36] K. Thompson, "Retrograde analysis of certain endgames," *J. Int. Comput. Chess Assoc.*, vol. 9, no. 3, pp. 131–139, 1986.
[37] P. Ulfhake, "A Chinese Checkers-playing program," M.S. thesis, Dept. Inf. Technol., Lund Univ., Lund, Sweden, 2000.
[38] R. Zhou and E. Hansen, "Space-efficient memory-based heuristics," in *Proc. Nat. Conf. Artif. Intell.*, 2004, pp. 677–682.
[39] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," *Neural Inf. Process. Soc.*, 2007 [Online]. Available: http://books.nips.cc/papers/files/nips20/NIPS2007_0682.pdf



Mehdi Samadi received the B.Sc. degree in computer science and engineering from Shiraz University, Shiraz, Fars, Iran, in 2007 and the M.S. degree in computer science from University of Alberta, Edmonton, AB, Canada, in 2008.

His research interests include heuristic search, AI in games, planning, multirobot planning, and multiagent systems.

Fatemeh Torabi Asr received the B.Sc. degree in computer software engineering from Shiraz University, Shiraz, Fars, Iran, in 2007, where she is currently working towards the M.Sc. degree, which she will receive in 2009.

She is interested in interdisciplinary studies of intelligence and cognitive algorithms. Now she is working on the nature of human categorization in natural languages

Jonathan Schaeffer is a Professor in Computer Science and Vice Provost at the University of Alberta, Edmonton, AB, Canada. His research is in artificial intelligence, using computer games as his experimental test bed. He is best known for creating the checkers program Chinook, the first computer program to win a human world championship in any game. In 2007, he announced that *Checkers* was solved.

Zohreh Azimifar received the B.Sc. degree in computer science and engineering from Shiraz University, Shiraz, Fars, Iran, in 1994 and the Ph.D. degree in systems design engineering from the University of Waterloo, Waterloo, ON, Canada, in 2005.

In 2005, she was a Postdoctorate Fellow in Medical Biophysics at the University of Toronto, Toronto, ON, Canada. Since 2006, she has been a Faculty Member in Computer Science and Engineering at Shiraz University. Her research interest includes AI in games as well as statistical learning of predictive models for massive data sources with particular attention in computer vision and pattern recognition.