# Automatic Heuristic Construction for General Game Playing

**Gregory Kuhlmann and Peter Stone**
Department of Computer Sciences
The University of Texas at Austin
1 University Station [C0500]
Austin, TX 78712
{kuhlmann,pstone}@cs.utexas.edu

## Introduction

Creating programs that can play games such as chess, checkers, and backgammon, at a high level has long been a challenge and benchmark for AI. While several game-playing systems developed in the past, such as Deep Blue (Campbell, Jr., & Hsu 2002), Chinook (Schaeffer *et al.* 1992), and TD-gammon (Tesauro 1994) have demonstrated competitive play against human players, such systems are limited in that they play only one particular game and they must be supplied with large amounts of expert knowledge. While their performance is impressive, it could be argued that the demonstrated intelligence is that of the system's designers more so than that of the system itself.

*General* Game Playing is the more challenging problem of designing an agent capable of playing many different previously unseen games (Pell 1993). The first AAAI General Game Playing Competition was held at 2005 AAAI meeting in Pittsburgh in order to promote research in this area (Genesereth & Love 2005). We were one of nine participants in that competition.

We survey some of the issues involved in creating a general game playing system and introduce our entry to that event. The main feature of our approach is a novel method for automatically constructing effective search heuristics based on the formal game description. Our agent is fully implemented and tested in a range of different games.

## General Game Playing

The General Game Playing problem is the challenge of creating a system capable of playing games with which it has had no prior experience. The definition of a "game" can be quite broad, ranging from single-state matrix games such as the "Prisoner's Dilemma" to complex, dynamic tasks like robotic soccer. The class of games considered for the General Game Playing Competition is restricted to discrete state, deterministic, perfect information games. The games may be single or multi-player. They may be turn-taking or simultaneous decision. Chess, checkers and tic-tac-toe are all examples of games that fit into this class. However, backgammon would not because dice rolls are nondeterministic. Also, games like poker or Battleship do not qualify because they contain hidden state.



In the general game playing scenario that we consider, agents are supplied with a description of the game to be played immediately before the game begins. The Game Description Language (GDL) describes a game in first-order logic and includes the initial state of the game, the set of legal moves for each player in each state, the state transition function, the set of terminal states, and the utility rewarded to each player in each of the terminal states. A game description contains sufficient information to allow a player to simulate the game, itself, using a theorem prover.

In the competition setup, each player runs as an independent server process. At the start of a game, another process called the Game Manager connects to the players and sends the game description along with announced time limits for making moves. Players may analyze the game description and contemplate their moves before announcing their action within the time constraints. After each time step, the game manager informs the players of the moves made by their opponents. The game continues until a terminal state is reached. Agents are evaluated by their one-shot reward against unknown opponents.

## Approach

Our overall approach to the General Game Playing problem is distributed heuristic search, with a focus on developing methods for generating heuristics automatically from game descriptions. Through static analysis of the game description, we find occurrences of common game structures, such as boards, pieces, and time steps. We further refine these features by internally simulating the game. Finally, the features directly suggest heuristic functions, which are evaluated online as part of a distributed search algorithm.

Our player identifies game structures by syntactically matching game description clauses to a set of template patterns. Because the game description tokens were scrambled during the competition, we cannot use any lexical clues in the pattern matching. Therefore, all structures are detected by their syntactic structure alone.

One of the most basic structures we look for are numbers. Because GDL lacks arithmetic, relations are often defined in a game description to support numerical relationships. Another important structure that our player looks for in a game is a "board". A board is simply a two dimensional grid of cells that change state, such as a chess or checkers board. An

example of a board would be CELL in the set of relations:

```
(CELL 1 1 X)
(CELL 1 2 O)
. . .
```

meaning there is an X at coordinate (1,1), an O at coordinate (1,2), etc.

We begin by assuming that all ternary relations are boards. However, a board must meet the additional requirement of having one of its arguments correspond to the cell's state, which can never have two values at the same time. Rather than attempting to prove this invariant formally, we instead use internal simulation to determine if the invariant holds for our hypothesized board. We visit states randomly for the first 10% of the startup time, checking to see if our condition holds. While simulation may never prove the invariant, sufficient simulation of the game allows us to be reasonably confident that it holds.

Using the same technique, we identify related structures such as *markers*, which are objects that occupy the cells of the board, and *pieces*, which are markers that can only be in one board cell at a time.

While the specific templates we describe are tailored to the idiosyncrasies of GDL, our method of syntactic template matching combined with internal simulation could be applied to any formal domain description language.

The game structures identified by our agent suggest interesting features that can feed into a static evaluation function. For example, if the agent identifies a board with pieces and ordered coordinates, it can then compute the Manhattan distance between each of the pieces. Or if a board contains a couple of different kinds of markers, the quantities of those markers could be useful features.

From the set of features that apply to our particular game, we generate candidate heuristics to guide search. Although more complex possibilities exist, we define each heuristic to be the maximization or minimization of a single feature. We implement the heuristic functions as static board evaluators that linearly map the feature's value to an expected reward between $R^- + 1$ and $R^+ - 1$, where $R^-$ and $R^+$ are, respectively, the minimum and maximum goal values achievable by the player, as defined in the game description. We chose to scale the heuristic function in this way so a definite loss is always worse than any approximated value, and likewise, a definite win is always better than an unknown outcome. The value of the heuristic function $H(s)$ is defined as follows:

For Maximizing:

$$H(s) = 1 + R^- + (R^+ - R^- - 2) * V(s) \qquad (1)$$

For Minimizing:

$$H(s) = 1 + R^- + (R^+ - R^- - 2) * [1 - V(s)] \qquad (2)$$

where $V(s)$ is the value of the feature in state $s$ scaled between 0 and 1.

Not all of the candidate heuristics that the player generates from the game description will be particularly good, of course. Nor will the best choice of heuristic necessarily remain the same throughout the course of the game.

We evaluate the candidate heuristics online using distributed search. In addition to the main game player process, our player launches several remote slave processes. The main process informs each slave of the current state and assigns them a heuristic to use for search. The same heuristic may be assigned multiple times if the number of heuristics is small and the number of slave processes is large. Although this redundancy does typically lead to significant overlap of work, randomization in the search process results in the different processes exploring different parts of the search space. Occasionally, the slave processes announce their best move so far to the main process. Before the time expires, the game player evaluates the suggested actions, chooses the *best* one and sends the move to the game manager.

The search algorithm employed by our player is an iterative deepening minimax search with alpha-beta pruning. Two general-purpose enhancements were used: transposition tables, which cache the value of states encountered previously in the search, and the history heuristic, which reorders children based on their values found in lower-depth searches. Several other general techniques exist, but the combination of these two techniques is known to account for almost all of the search space cutoff when combined with the other techniques (Schaeffer 1989). We extended this basic minimax search to allow simultaneous decision games and games with more than two players.

## Results and Conclusion

During the competition, our agent consistently identified relevant structures such as boards and step counters in the games it encountered. One of our player's biggest successes was in Nothello, a game similar to Othello but with the goal of ending up with *fewer* pieces than your opponent. Because we included both the maximization and minimization of each feature as candidate heuristics, our player was able to identify this counterintuitive goal condition.

Experiments following the competition have been promising in that a good automatically constructed heuristic outperforms blind search. Further work is needed, however, to evaluate the strengths and weaknesses of our approach.

## References


Campbell, M.; Jr., A. J. H.; and Hsu, F. H. 2002. Deep blue. *Artificial Intelligence* 134(1–2):57–83.

Genesereth, M., and Love, N. 2005. General game playing: Overview of the AAAI competition. *AI Magazine* 26(2).

Pell, B. 1993. Strategy generation and evaluation for meta-game playing. PhD thesis, University of Cambridge.

Schaeffer, J.; Culberson, J. C.; Treloar, N.; Knight, B.; Lu, P.; and Szafron, D. 1992. A world championship caliber checkers program. *Artificial Intelligence* 53(2-3):273–289.

Schaeffer, J. 1989. The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-11(11):1203–1212.

Tesauro, G. 1994. Td-gammon, a self-teaching backgammon program, achieves masterlevel play. *Neural Computation* 6:215–219.