Heuristic Evaluation Functions for General Game Playing

James Clune
Department of Computer Science
The University of California, Los Angeles
jclune@cs.ucla.edu

Abstract

A general game playing program plays games that it has not previously encountered. A game manager program sends the game playing programs a description of a game's rules and objectives in a well-defined game description language. A central challenge in creating effective general game playing programs is that of constructing heuristic evaluation functions from game descriptions. This paper describes a method for constructing evaluation functions that represent exact values of simplified games. The simplified games are abstract models that incorporate the most essential aspects of the original game, namely payoff, control, and termination. Results of applying this method to a sampling of games suggest that heuristic evaluation functions based on our method are both comprehensible and effective.

Introduction

The idea of general game playing is to create a program that effectively plays games that it has not previously encountered. A game manager program sends the game playing programs a description of a game in a well-defined game description language. The description specifies the goal of the game, the legal moves, the initial game state, and the termination conditions. The game manager also sends information about what role the program will play (black or white, naughts or crosses, etc), a start time (time allowed for pre-game analysis), and a move time (time allowed per move once game play begins). The game playing programs compete by sending messages over a network indicating their moves until the game is completed. The class of games covered is intentionally broad, including games of one or more players with alternating or simultaneous moves, with arbitrary numeric payoffs. Our research focus is the automatic construction of heuristic evaluation functions that enable a general game-playing program to win games.

Game playing programs have been of interest to the AI community since the field's inception because they provide a well-defined test-bed for theories of how computational processes can produce intelligent behavior. Although impressive achievements have been made in a number of games, a criticism of this work has been that the success of high-performance game-playing programs has come at the expense of generality.

The idea of general game playing as a test-bed for AI is presented in Barney Pell's thesis (Pell 1993). Pell submits general game playing as a way to preserve the elements that have made game-playing research attractive to the AI community, while maintaining generality. He defines a class of *symmetric chess-like games* and describes a system that plays games within this class. The system creates heuristics through the application of metagame-level analysis of the class of games, applying concepts such as mobility, centrality, and promotion, automatically specializing these concepts for the particular games.

The version that is the focus of the present work is based on the AAAI General Game Playing Competition, organized by Michael Genesereth (Genesereth, Love, & Pell 2005) and the Game Description Language, or GDL (Genesereth & Love 2005). One of the challenges in automatically constructing heuristic evaluation functions for games from GDL descriptions is the generality of the language. For example, although the game of chess is describable in GDL (with some minor caveats), GDL has no notion of a grid-like board or pieces that move, capture, and promote. Instead, each of these concepts are constructed in a game-specific way in first-order logic. This approach forces program authors to implement extremely general game-playing agents.

The first published work on generating heuristics for GDL is (Kuhlmann, Dresner, & Stone 2006). They describe a method of creating features from syntactic structures by recognizing and exploiting relational patterns such as successor relations and board-like grids. They then perform distributed search, where each machine uses a single heuristic based on either the maximization or the minimization of one of the detected features.

Another approach to constructing heuristic evaluation functions for GDL is described in (Schiffel & Thielscher 2007). Their approach also recognizes structures such as successor relations and grids in game descriptions. Beyond structure recognition, they utilize fuzzy logic and techniques for reasoning about actions to quantify the degree to which a position satisfies the logical description of winning. They also assess how close the game is to termination, seeking terminal states when goals are reached and avoiding terminal

states when goals are not yet attained.

The work presented here is part of an ongoing research project which has produced general game playing programs that have placed first and second in the First and Second Annual AAAI General Game Playing Competitions, respectively. Algorithms and results presented here reflect the current state of the ongoing project.

The core ideas governing our present approach are the following. One source of a heuristic evaluation function is that of an exact solution to simplified version of the original problem (Pearl 1984). Applying this notion to general game playing, we abstract the game to its core aspects and compute the exact value of the simplified game, where the value of a game is interpreted as the minimax value as described in (von Neumann & Morgenstern 1953). Because this simplified game preserves key characteristics of the original game, we use the exact value of the simplified game as an approximation of the value of the original game. The core aspects of the game that we model are the expected payoff, the control (or relative mobility), and the expected game termination (or game longevity). These core game aspects are modeled as functions of game-state features. The features must be automatically selected, and the key idea guiding this selection is *stability*, an intuitive concept for which we will provide a more technical meaning. Our approach strives to evaluate states based on only the most stable features.

Our approach shares some aspects with previous approaches. Like Kuhlmann, Dresner, & Stone, binary relations are extracted from the game description. Like Schiffel & Thielscher, the approach incorporates a measure of degree of goal attainment. Distinguishing aspects of the present work include the simplified model of the game in terms of payoff, control, and termination, and a particular notion of stability as a key characteristic of relevant features.

The remainder of this paper is as follows. We explore an example game, discussing both its representation in GDL and the game's analysis. We then present preliminary results of automatically constructed heuristic evaluation functions for a sampling of board games, and end with discussion.

Example: Cylinder Checkers

We illustrate GDL and a technique for constructing heuristic evaluation functions with a checkers variant called cylinder checkers. The game was introduced in the Second Annual AAAI General Game Playing Competition. The primary difference from traditional checkers is that the playing board has a topology in which the sides wrap around to form a cylinder. Also unlike traditional checkers, where the goal is to make one's opponent unable to move, the goal in cylinder checkers is to maximize the piece count of one's own pieces and minimize the piece count of the opponent's pieces. The payoff values range from 0 to 100 and are proportional to the difference in piece count. As in traditional checkers, jumps are forced. The game ends when either player cannot move or after fifty moves, whichever comes first.

GDL Representation

In GDL, states are represented by the set of facts (or fluents) true in the given state. Rules are logical statements composed of *expressions* consisting of fluents, logical connectives, a distinguished set of GDL relations, and game-specific relations. Here we briefly describe the GDL representation of cylinder checkers, omitting the syntax.

The *role* relation lists the roles of the players in the game (red and black). The *init* relation enumerates the set of fluents true in the initial state of the game. These indicate the initial board configuration and that red moves first.

The *legal* relation indicates the legal moves for each role from a given state. The *next* and *does* relations are used to describe resulting fluents in terms of existing fluents and player's actions. For example, one rule is that moving a piece from cell (x, y) results in cell (x, y) being blank.

The *terminal* relation is used to indicate termination conditions for the game, in this case after 50 moves or when a player cannot move. The *goal* relation is used to indicate payoffs for terminal states. A rule says that if black has no pieces, then the payoffs are 100 for red and 0 for black.

Identifying Stable Features

When a human first learns the game of checkers, there are some obvious metrics that appear relevant to assessing states: the relative cardinalities of regular pieces of each color and kings of each color. The more sophisticated player may consider other factors as well, such as the number of openings in the back row, degree of center control, or distance to promotion. We will use the term *feature* to refer to a function from states to numbers that has some relevance to assessing game states. A feature is not necessarily a full evaluation function, but features are building-blocks from which evaluation functions can be constructed.

Consider the cues that human checkers players exploit when learning to evaluate positions. Pieces are represented as physical objects, which have properties of persistence and mobility that humans have extensive experience with. The coloring of the pieces, the spatial representation of the board, and the symmetry of the initial state each leverage strengths of the human visual system. The GDL game description, although isomorphic to the physical game, has none of these properties, so feature identification is not so simple.

The approach taken here is to extract expressions that appear in the game description and impose interpretations on these expressions to construct candidate features. There are two aspects to this: identifying a set of potentially interesting candidate expressions and imposing interpretations on these expressions. To identify the set of candidate expressions, the analyzer starts by simply scanning the game description to see what expressions appear. The theory behind this seemingly naive approach is that a game that is succinctly described will necessarily be described in terms of its most salient features. In the case of cylinder checkers, this includes expressions representing red pieces, black pieces, red kings, and black kings.

In addition to the expressions appearing directly in the game description, the feature-generating routine has some built-in rules for generating additional expressions. Chief among these is a routine which performs domain analysis to determine what constants can appear in what positions for expressions involving variables. These constants are then

substituted into expressions involving variables to generate new, more specific candidate expressions. When the domain of a particular variable has only a few constants, they are each substituted into the expression. In cylinder checkers, variables corresponding to pieces are found to permit symbols for red pieces, black pieces, red kings, and black kings. When the domain has more constants, then additional structure is sought to determine if some of the constant values are distinct from the rest. In cylinder checkers, the first and last rows are found to be distinct from the rest because the first and last rows have a single adjacent row, while the other rows have two adjacent rows.

When the expressions are identified, the set of candidate features is generated by imposing various interpretations of these expressions. The analyzer has three primary interpretations, each defining a different type of feature.

We call the first interpretation the *solution cardinality* interpretation. The idea is that we take the given expression, and find the number of distinct solutions to this expression there are in the given state. In the case of the expression (cell ?x ?y black_king), the number of solutions corresponds to the number of black kings on the board.

The second interpretation is the *symbol distance* interpretation. Binary relations among symbols in GDL are game-specific relations with arity two. For example, the cylinder checkers description utilizes a game-specific *next_rank* relation to establish ordering of rows. We construct a graph of game-specific symbols appearing in the description, where each constant symbol is a vertex in the graph. Edges are placed between vertices that appear together in the context of a binary relation in the game description. For example, rank3 and rank4 have an edge between them due to the rule (next_rank rank3 rank4). Once this graph is constructed, the symbol distance between two symbols is the shortest path between the two symbols along this graph.

To impose the symbol distance interpretation on an expression such as (cell ?x rank8 red_piece), we first identify constants within the expression that are in the domain of a binary relation. In this case, the only such constant is rank8, which appears in the *next_rank* relation. We substitute a variable for this symbol, obtaining an abstract expression (cell ?x ?y red_piece). We find all the solutions to the abstract expression. For each solution, we find the distance to our original expression based on the binding of the newly introduced variable. For example, if the solutions are (cell a rank2 red_piece) and (cell f rank5 red_piece), then the distances are 6 and 3, respectively, because according to the *next_rank* rules, the distance from rank2 to rank8 is 6 and from rank5 to rank8 is 3. Finally, the overall symbol distance is the sum of the distances for the individual solutions to the abstract expression. In this example, the value would be 3 and could represent the distance to king promotion.

The third interpretation is the *partial solution* interpretation. This interpretation only applies to compound expressions involving multiple conjuncts or disjuncts. For example, in Connect-4, the rule for winning involves a conjunction of four pieces of the same color in a row. The partial solution interpretation of the conjunction results in a number that is proportional to the fraction of conjuncts satisfied (0.75 for three in a row). This is similar to Schiffel & Thielscher's degree of goal attainment.

Table 1: Abstract Model Parameters

Parameter	Meaning
$P : \Omega \to [0, 100]$	Approximates payoff function.
$C : \Omega \to [-1, 1]$	Degree of control player has.
$T : \Omega \to [0, 1]$	Probability that state is terminal.
$S_P : (1, \infty)$	Stability of P.
$S_C : (1, \infty)$	Stability of C.

For each candidate expression, we create corresponding features by applying each of the possible interpretations to the expression. Finally, some additional features are generated by observing and exploiting symmetry in the game description. Utilizing the domain information extracted earlier, we can observe that red_piece and black_piece appear in the same number of relations the same number of times and that each have an initial cardinality of twelve. From this, we can hypothesize that the solution cardinality interpretations of (cell ?x ?y red_piece) and (cell ?x ?y black_piece) are symmetric to each other and introduce a *relative* feature for the difference between the two. In this case, the feature represents the material piece advantage of red over black.

Once we have generated a set of candidate features, we need some way of determining which features are most likely to be relevant to state evaluation. The intuition behind our criteria will be that quantities which wildly oscillate do not provide as good a basis for assessing the value of a state as quantities that vary only incrementally. We quantify this idea by introducing a measure called the *stability*.

To compute the stability of a feature, we first generate a set of sample states in the game through random exploration of the game tree. We compute the value of the feature for each sample state. Next, we compute the variance of the feature's values over the sample states. This is the *total variance*. Two pairs of states are *adjacent* if one is the immediate successor of the other in some path through the game tree. We calculate another quantity which we will call the *adjacent variance* by summing the squares of the difference in feature values for adjacent sample states and dividing by the number of adjacent state pairs. We define the *stability quotient* to be the ratio of the overall variance to the adjacent variance. We also refer to the stability quotient as simply the *stability*. If the feature wildly oscillates from state to state, the stability will be low (≈ 1), whereas if the feature value changes only incrementally, the stability will be significantly greater than one.

Abstract Model

The abstract model reduces the game to five parameters: P (payoff), C (control), T (termination), S_P (payoff stability), and S_C (control stability). The ranges and intuitive meanings of the various game parameters are summarized in Table 1, where Ω denotes the set of legal game states.

The intention of the payoff value is to provide a function from states to numbers that evaluates to the role's payoff values for the terminal states and is approximately continuous

over the topology of the game tree. To construct this, we start with our list of stable features. We eliminate some features through a dependency analysis of the rules in the game description. We exclude features based on expressions involving relations that do not influence goal expressions.

Next, we categorize the remaining stable features by their correlation to the payoff function. The correlation is either positive, negative, or not correlated. To determine the correlation, we construct a pair of pseudo-states that are identical except that one has the expression associated with the feature and the other does not. We compute the payoff value of the pseudo-states based on the game's payoff rules as if the pseudo-states were terminal states. If the payoff values are the same, the feature is considered uncorrelated with payoff. If the payoff is higher on the state with the higher feature value, then the correlation is positive and if the payoff is lower on the state with the higher feature value, then the correlation is negative. Of the features having non-zero correlation, we exclude absolute features that are subsumed by relative features. In the case of cylinder checkers, the features correlated with payoff are the red piece count, the black piece count, and the difference in piece counts, so the difference in piece counts is retained. When there are multiple features that are correlated with payoff and not subsumed by other features, the features are weighted according to their stability, with the coefficient being positive for positively correlated features and negative for negatively correlated features. Finally, the overall coefficient and offset are set such that the values of the resulting payoff function for all the sample states falls in the range of [0, 100].

The control function is intended to quantify differences in the number of moves available to each role. Let the moves available to role k at state $\omega \in \Omega$ be denoted $m_k(\omega)$. We define the control at state ω to be:

$$\frac{m_{red}(\omega) - m_{black}(\omega)}{\max_{\omega' \in \Omega}(m_{red}(\omega') + m_{black}(\omega'))} \quad (1)$$

Positive numbers indicate red has more moves, negative numbers indicate black has more moves. In games with more than two roles, m_{black} is replaced with the sum of the number of moves of the adversaries. The denominator cannot be measured directly because the state space is too large, so we approximate by taking the maximum quantity over the sample states.

To compute the control function, we begin with the stable features. We eliminate features associated with expressions that do not influence *legal* expressions, as these features do not impact the moves available to the various roles. We remove absolute features subsumed by relative features. In cylinder checkers, we end up with two features: the number of red pieces minus the number of black pieces and the number of red kings minus the number of black kings. To quantify the relative contribution of these features, we take a statistical approach. We generate a collection of sample states by simulating game play with random moves. We perform least squares regression with the control as the dependent variable and the features as the independent variables to find the best fit for control values in terms of the features.

The third function that we use in constructing our heuristic evaluation function is termination. It is used to determine the relative importance of the payoff and control functions, the intuition being that in the opening it may make sense to attempt to gain more control of the game's trajectory, but that in the endgame focusing on the payoff is paramount. Treating termination as probabilistic is perhaps counter-intuitive, given that the termination of each state is in fact deterministic. The probabilistic treatment enables us to abstract over a region of states with similarly valued stable features. The termination function is computed statistically by least squares regression, with the target values 1 for terminal states and 0 for non-terminal states.

Heuristic Evaluation Function

To determine how to combine the payoff, control, and termination into an overall heuristic evaluation function, we consider the game as a compound lottery. With probability T, the game terminates and red is awarded payoff P. With probability $1 - T$, the game continues and has an outcome determined by a second lottery. The second lottery has two possible outcomes. The first outcome, with probability S_C is a payoff to red of $100 * C$. The second outcome, with probability $S_P = 1 - S_C$ is a payoff to red of P. The values P, C, and T are the values of the payoff, control, and termination functions evaluated at the given state. The values S_P and S_C are the payoff stability and control stability, respectively, computed as the stability of functions P and C just as the stability of the primitive features were computed in the previous section.

The result of running the analyzer on cylinder checkers for red for 10 minutes is as follows. Although the analysis program prints the features in terms of GDL syntax, we present the features here in their English translations for improved readability.

Payoff = 50 + 10 (# total red - # total black)
Control =
0.087 (# red kings - # black kings)
+0.042 (# red pieces - # black pieces)
Terminal = -0.002 (# steps before we reach step 50) + 0.083
Payoff Stability = 0.341, Control Stability = 0.659

Given the values for the parameters in Table 1, the expected outcome v of the lottery follows directly according to probability calculus:

$$v = T * P + (1 - T)((50 + 50 * C) * S_C + S_P * P) \quad (2)$$

A key parameter influencing both the quality of the results and the computational cost of the analysis is the number of sample states generated. More samples generally result in better results at the expense of longer analysis time. It is difficult to predict how long the model construction will take for a given number of samples for a given game, so the analyzer implements an anytime algorithm that produces better quality game models given more time. The anytime algorithm simply starts with a small number of sample states (25), and computes the model. It then doubles the number of sample states and recomputes the model. It repeats this until there is no analysis time remaining, at which point it returns the most recently completed model.

To utilize these functions to play games, we perform an iterative-deepening minimax search with well-known enhancements (alpha-beta pruning, transposition tables, and aspiration windows). When evaluating nodes at the frontier, terminal nodes are valued according to their actual payoff values as determined by the game description. To evaluate non-terminal nodes, we first compute values for U, T, C, S_U, S_C by evaluating the functions constructed in the analysis phase. We plug the resulting values into Equation 2. These values are propagated according to the minimax algorithm and are used as the basis for move selection.

Preliminary Results

The operations for automatically constructing heuristic evaluation functions as described in the previous section have been implemented in OCaml. This section reports results so far in running the heuristics constructor on a sampling of games. The results were obtained by running our single-threaded program on a MacBook Pro with a 2.16 GHz Intel Core Duo. Game descriptions for each of the games were provided by the Stanford General Game Playing group. Different analysis times were used on the various games because larger games take longer for the analysis to converge to a consistent evaluation function.

Racetrack Corridor

Racetrack corridor was introduced in the championship match of the First Annual General Game Playing Competition. It was inspired by the game of Quorridor, which involves moving a piece across a board and placing walls to impede the opponent's progress. Racetrack corridor is played on 3 x 5 grids as shown in Figure 1.

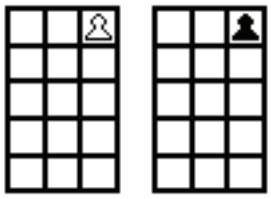

Figure 1: Racetrack Corridor (initial position)

Each player controls a piece that starts at the top of one of the grids and must race lengthwise to the other side. In each move, a player may choose to move a piece forward or sideways or may place a horizontal wall that blocks the middle and one side blocks one side of the opponent's track. Moves are made simultaneously. Each player begins the game with four walls. The game ends when either player reaches the bottom or after 20 moves, whichever comes first. Though the board looks small, the initial branching factor is one-hundred because each player simultaneously chooses from 10 legal moves.

The following results were obtained for white in 10 minutes of analysis:

Payoff = 50 + 10 (black goal distance - white goal distance)
Control =
0.091 (# walls on right side: white lane - black lane)
+0.090 (# walls on left side: white lane - black lane)
+0.006 (# pieces in top row: white - black)
Terminal = 0.014 (# steps into the game) -0.088
Payoff Stability = 0.729, Control Stability = 0.271

The payoff function values proximity to the goal, an intuitively obvious heuristic in this game. The likelihood of termination increases with each step of the game. The control function is non-intuitive, but its lower stability dictates that it will be weighed less heavily than the payoff function.

Othello

An interesting challenge in constructing heuristic evaluation functions for Othello is that the intuitive strategy of greedily maximizing one's disks throughout the game results in poor performance. Machine learning techniques have been effectively utilized to construct heuristic evaluation functions for Othello, such as in (Buro 2002). However, these techniques have relied on self-play on the order of hundreds of thousands of games. Thirty minutes of analysis on Othello produced the following result:

Payoff =
3.361 (lower right corner: # white - # black)
+2.015 (upper right corner: # white - # black)
+1.379 (upper edge: # white - # black)
+1.221 (lower left corner: # white - # black)
+1.114 (lower edge: # white - # black)
+0.789 (upper left corner: # white - # black)
+0.646 (right edge: # white - # black)
+0.592 (left edge: # white - # black)
+50.000

No stable features were found to be predictive of control, so the heuristic evaluation function reduces to the payoff function. The payoff function is piece differential, but consists of only the four corners and four edges. We are encouraged to note that the naive approach of maximizing the number of total pieces throughout the game is not advocated by the above evaluation function.

Chess

Chess is a challenging game with a long history of AI work, most of which has utilized highly tuned heuristic evaluation functions formulated by chess experts. The GDL game description for chess holds fairly closely to the official game rules, including special moves such as castling and en passant. It does not, however, contain the rules for a draw after repeating a state three times. Instead, it calls a game a draw if no checkmate or stalemate is reached after 200 moves.

Two hours of analysis yielded the following result:

Control =
0.097 (# white queens - # black queens)
+0.092 (# white rooks - # black rooks)
+0.042 (# white bishops - # black bishops)
+0.039 (# white knights - # black knights)
+0.022 (# empty squares: rank 1 - rank 8)
+0.004 (# white pawns - # black pawns)

It did not find any stable features correlated with payoff, which is unsurprising given the conditions for checkmate. The heuristic evaluation function reduces to the con-

trol function, which is dominated by a valuation of material. Among material features, it valued queens highest, followed by rooks, then bishops, then knights, then pawns. This relative ordering is consistent with the classical valuation of chess pieces (9 for queens, 5 for rooks, 3 for bishops, 3 for knights, and 1 for pawns). The non-material feature indicates an advantage in clearing one's first rank. This appears to be due to the increased mobility achieved by moving the major pieces out from behind the row of pawns.

Chinese Checkers

Chinese Checkers is a multi-player game played with marbles on a star-shaped board. The program analyzed a GDL description of a small version of Chinese Checkers for six players, each of which controls three marbles on a star-shaped board. Ten minutes of analysis for red yielded:

Payoff =
1.201 (fraction of red marbles in goal)
-3.000 (total distance red marbles need to travel)
+80.0

No stable features were found to correlate with control, so the evaluation function is strictly the payoff function. In this six-player game, the analyzer was unable to deduce relative features, so only features relevant to the player's own goals appear. The function primarily values minimizing distance to the goal and secondarily attempts to maximize the fraction of marbles actually in the goal.

Empirical Trials

We performed empirical trials to test the effectiveness of the heuristics based on the above models. We pitted two versions of our player against each other that were identical except for the heuristic evaluation functions. One version used the evaluation functions described above, and the other used a simple heuristic that we call *early payoff*. The early payoff heuristic simply computes what the game description indicates the payoff of the given state would be if that state were terminal. The quality of this heuristic varies from fairly good for cylinder checkers to admittedly poor for chess. All games were run multiple times, switching roles between each game and averaging the payoffs for the mean score. In the case of Chinese Checkers, games were run with three players using the model-based heuristics and three players using the early payoff heuristic.

The results are shown in Table 2. The model-based heuristics outperformed the early payoff heuristic in every case. The closest margins were in Chinese Checkers, which is a little difficult to interpret because it is a six-player game, and cylinder checkers, where the early payoff heuristic works quite well. This is evidence that the heuristic evaluation functions are indeed effective.

Table 2: Mean Scores from Empirical Trials

Game	Model-Based	Early Payoff
Cylinder Checkers	59	41
Racetrack Corridor	100	10
Othello	75	25
Chess	100	0
Chinese Checkers	42	30

Discussion

A core idea in the approach to heuristics for general game playing outlined here is the abstraction of a specific game to a simple model that provides a quantitative summary of three aspects: payoff, control, and termination. An enabling technique is the introduction of stability as a quantitative method of distilling the features most relevant to this summary.

The concept of stability is deceptively simple, but its significance is demonstrated by the specificity with which it identifies relevant features. For example, both chess and Othello are played on an 8x8 board described by successor relations. Stability considerations enable the analyzer to ascertain the importance of corners in Othello while rejecting them as features in chess. Conversely, stability concerns lead the analyzer to reject overall material as a heuristic in Othello, while adopting it in chess and cylinder checkers.

The results so far suggest that the techniques described here construct heuristic evaluation functions that are both comprehensible and effective.

Acknowledgments

We thank Rich Korf for insightful advice throughout the project. Thanks also to Alex Dow, Alex Fukunaga, and Eric Huang for comments on an earlier version of this paper.

References

Buro, M. 2002. The evolution of strong Othello programs. In *IWEC*, 81–88.

Genesereth, M., and Love, N. 2005. General game playing: Game description language specification. Technical report, Computer Science Department, Stanford University, Stanford, CA, USA. http://games.stanford.edu/gdl_spec.pdf.

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

Kuhlmann, G.; Dresner, K.; and Stone, P. 2006. Automatic Heuristic Construction in a Complete General Game Player. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, Boston, Massachusetts, 1457–62.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, Massachusetts: Addison-Wesley.

Pell, B. D. 1993. *Strategy Generation and Evaluation for Meta-Game Playing*. Ph.D. Dissertation, University of Cambridge.

Schiffel, S., and Thielscher, M. 2007. Automatic Construction of a Heuristic Search Function for General Game Playing. In *Seventh IJCAI International Workshop on Nonmontonic Reasoning, Action and Change (NRAC07)*.

von Neumann, J., and Morgenstern, O. 1953. *Theory of Games and Economic Behavior*. Princeton: Princeton University Press.