Automatic Construction of a Heuristic Search Function for General Game Playing

Stephan Schiffel and **Michael Thielscher**
Department of Computer Science
Dresden University of Technology
{stephan.schiffel,mit}@inf.tu-dresden.de
Student Paper

Abstract

General Game Playing (GGP) is the art of designing programs that are capable of playing previously unknown games of a wide variety by being told nothing but the rules of the game. This is in contrast to traditional computer game players like Deep Blue, which are designed for a particular game and can't adapt automatically to modifications of the rules, let alone play completely different games. General Game Playing is intended to foster the development of integrated cognitive information processing technology. In this article we present an approach to General Game Playing using a novel way of automatically constructing a search heuristics from a formal game description. Our system is being tested with a wide range of different games. Most notably, it is the winner of the AAAI GGP Competition 2006.

1 Introduction

General Game Playing is concerned with the development of systems that can play well an arbitrary game solely by being given the rules of the game. This raises a number of issues different from traditional research in game playing, where it is assumed that the rules of a game are known to the programmer. Writing a player for a particular game allows to focus on the design of elaborate, game-specific evaluation functions (e.g., [Morris, 1997]) and libraries (e.g, [Schaeffer *et al.*, 2005]). But these game computers can't adapt automatically to modifications of the rules, let alone play completely different games.

Systems able to play arbitrary, unknown games can't be given game-specific knowledge. They rather need to be endowed with high-level cognitive abilities such as general strategic thinking and abstract reasoning. This makes General Game Playing a good example of a challenge problem which encompasses a variety of AI research areas including knowledge representation and reasoning, heuristic search, planning, and learning. In this way, General Game Playing also revives some of the hopes that were initially raised for game playing computers as a key to human-level AI [Shannon, 1950].

In this paper, we present an approach to General Game Playing which combines reasoning about actions with heuristic search. Our focus is on techniques for constructing a search heuristics by the automated analysis of game specifications. More specifically, we describe the following functionalities of a complete General Game Player:

1. Determining legal moves and their effects from a formal game description requires reasoning about actions. We use the Fluent Calculus and its Prolog-based implementation FLUX [Thielscher, 2005].
2. To search a game tree, we use non-uniform depth-first search with iterative deepening and general pruning techniques.
3. Games which cannot be fully searched require automatically constructing evaluation functions from formal game specifications. We give the details of a method that uses Fuzzy Logic to determine the degree to which a position satisfies the logical description of a winning position.
4. Strategic, goal-oriented play requires to automatically derive game-specific knowledge from the game rules. We present an approach to recognizing structures in game descriptions.

All of these techniques are independent of a particular language for defining games. However, for the examples given in this paper we use the Game Description Language developed by Michael Genesereth and his Stanford Logic Group; the full specification of syntax and semantics can be found at `games.stanford.edu`. We also refer to a number of different games whose formal rules are available on this website, too. Since 2005 the Stanford Logic Group holds an annual competition for General Game Players. The approach described in this paper has been implemented in the system FLUXPLAYER, which has has won the second AAAI General Game Playing competition.

2 Theorem Proving/Reasoning

Games can be formally described by giving an axiomatization of their rules. A symbolic representation of the *positions* and *moves* of a game is needed. For the purpose of a modular and compact encoding, positions need to be composed of *features* like, for example, `(cell ?x ?y ?p)` representing that `?p`

is the contents of square (?x,?y) on a chessboard.[1] The moves are represented by symbols, too, like (move ?u ?v ?x ?y) denoting to move the piece on square (?u,?v) to (?x,?y), or (promote ?u ?x ?p) denoting the promotion of a pawn to piece ?p by moving it from file ?u on the penultimate row to file ?x on the last row.

In the following, we give a brief overview of a specific Game Description Language, GDL, developed by the Stanford Logic Group and used for the AAAI competitions. For details, we refer to games.stanford.edu. This language is suitable for describing finite and deterministic n-player games ($n \geq 1$) with complete information. GDL is purely axiomatic, so that no prior knowledge (e.g., of geometry or arithmetics) is assumed. The language is based on a small set of keywords, that is, symbols which have a predefined meaning. A general game description consists of the following elements.

- The **players** are described using the keyword (role ?p), e.g., (role white).
- The **initial position** is axiomatized using the keyword (init ?f), for example, (init (cell a 1 white_rook)).
- **Legal moves** are axiomatized with the help of the keywords (legal ?p ?m) and (true ?f), where the latter is used to describe features of the current position. An example is given by the following implication:

```
(<= (legal white (promote ?u ?x ?p))
    (true (cell ?u 7 white_pawn))
    ... )
```

- **Position updates** are axiomatized by a set of formulas which entail all features that hold after a move, relative to the current position and the moves taken by the players. The axioms use the keywords (next ?f) and (does ?p ?m), e.g.,

```
(<= (next (cell ?x ?y ?p))
    (does ?player (move ?u ?v ?x ?y))
    (true (cell ?u ?v ?p)))
```

- The **end of a game** is axiomatized using the keyword terminal, for example,

```
(<= terminal checkmate)
(<= terminal stalemate)
```

  where checkmate and stalemate are auxiliary, domain-dependent predicates which are defined in terms of the current position, that is, with the help of predicate true.

- Finally, the **result** of a game is described using the keyword (goal ?p ?v), e.g.,

```
(<= (goal white 100)
    checkmate (true (control black)))
(<= (goal black 0)
    checkmate (true (control black)))
(<= (goal white 50) stalemate)
(<= (goal black 50) stalemate)
```

  where the domain-dependent feature (control ?p) means that it's player's ?p turn.

In order to be able to derive legal moves, to simulate game play, and to determine the end of a game, a general game playing system needs an automated theorem prover. The first thing our player does when it receives a game description is to translate the GDL representation to Prolog. This allows for efficient theorem proving. More specifically, we use the Prolog based Flux system [Thielscher, 2005] for reasoning. Flux is a system for programming reasoning agents and for reasoning about actions in dynamic domains. It is based on the Fluent Calculus and uses an explicit state representation and so called state-update axioms for calculating the successor state after executing an action. Positions correspond to states in the Fluent Calculus and features of a game, i.e. atomic parts of a position, are described by fluents. For the time being, the state update axioms we obtain don't take full advantage of the efficient solution provided by the Fluent Calculus and Flux for the inferential frame problem. Improvements to the translation are work in progress.

3 Search Algorithm

The search method used by our player is a modified iterative-deepening depth first search with two well-known enhancements: transposition tables[Schaeffer, 1989] for caching the value of states already visited during the search; and history heuristics[Schaeffer, 1989] for reordering the moves according to the values found in lower-depth searches. For higher depths we apply non-uniform depth first search, i.e. the depth limit is higher for moves that got a high value in a lower-depth search. This method allows for higher maximal depth of the search, especially with big branching factors. It seems to increase the probability that the search reaches terminal states earlier in the match because all games considered end after a finite number of steps. Thus non-uniform depth first search helps to reduce the horizon problem[Russell and Norvig, 2003]. The horizon problem arises often towards the end of a match and particularly in games ending after a limited number of steps. In those games the heuristic evaluation function can return a good value but the goal is in fact not reachable anymore because there are not enough steps left. A higher search depth for the moves considered best at the moment helps to avoid bad terminal states in that case. There is another rational behind non-uniform depth-limited search: It helps filling the transposition table with states we will look at again and therefore speeds up search in the future. If we search the state space behind a move we do not take, it is unlikely that we will encounter these states again in the next steps of the match.

Depending on the type of the game (single-player vs. multi-player, zero-sum game vs. non-zero-sum game) we use additional pruning techniques e.g., alpha-beta-pruning for two-player games. Our player also decides between turn-taking and simultaneous-moves games. For turn-taking games the node is always treated as a maximization node for the player making the move. So we assume each player wants to maximize his own reward. For simultaneous-moves games we make a paranoid assumption: we serialize the moves of

[1] Throughout the paper, we use KIF (the *Knowledge Interchange Format*), where variables are indicated by a leading "?".

the players and move first. Thereby we assume that all our opponents know our move. In effect of this we can easily apply pruning techniques to the search but the assumption may lead to suboptimal play. There is currently work in progress to use a more game-theoretic approach including the calculation of mixed strategy profiles and Nash-equilibria.

4 Heuristic Function

Because in most games the state space is too big to be searched exhaustively, it is necessary to bound the search depth and use a heuristic evaluation function for non-terminal states. In game playing systems for specific games these heuristic functions are often hand-made and very specific for a concrete game. In fact, much of the work done for creating computer game players is put in hand tuning the search heuristics. Because a general game playing system in our setting must be able to play all games that can be described with the GDL, it is not possible to provide a heuristic function beforehand that depends on features specific for the concrete game at hand. Therefore the heuristics function has to be generated automatically at runtime by using the rules given for the game. We developed a method for generating a heuristic function based on the rules of the game, specifically the rules for terminal and goal states.

The idea for the heuristic evaluation function is to calculate the degree of truth of the goal and terminal formulas in the state to evaluate. The values for goal and terminal are combined in such a way that terminal states are avoided as long as the goal is not fullfilled, i.e. the value of terminal has a negative impact on the evaluation of the state if goal has a low value and a positive impact otherwise.

One trivial approach would be to use fuzzy logic, i.e. assign values between 0 and 1 to atoms depending on their truth value and use some standard t-norm and t-co-norm to calculate the degree of truth of complex formulas. However this approach has undesirable consequences: With any standard t-norm the value of a conjunction is limited by the minimum of the values of the conjuncts. That means we would get a value of 0 for the whole conjunction if one of the conjuncts is false. Consider a simple blocks world domain with three blocks a, b and c and the goal $on(a, b) \wedge on(b, c) \wedge ontable(c)$. In that case we would want the value of the formula to reflect the number of subgoals solved. However if the only subgoal missing is $on(a, b)$ we would still get a value of 0 although the goal is almost reached. Another problem with this approach is that we can't differentiate between states that both fullfill a certain formula. E.g., the goal of the Othello game is to have more pieces of your own color on the board than there are opponent's pieces. A good heuristics would be to say the more pieces of your color are on the board the better. However the formula $greater(?whitepieces, ?blackpieces)$ would get a value of 1 no matter if white has just one or many more pieces than black.

To overcome these problems we use values $1 - p$ and p with $0.5 < p < 1$ for atoms that are false or true respectively. However this introduces new problems if one uses a continuous t-norm, e.g., $a * b$. Now a conjunction with a lot of conjuncts could get a value of nearly zero, although all conjuncts are true and on the other hand a big disjunction could get a value of nearly one although all disjuncts are false. But we want to use a continuous function otherwise we would still end up with the problem explained with the blocks world domain above.

Therefore we introduce a threshold t with $0.5 < t < 1$, with the following intention: values above t denote true and values below $1-t$ denote false. The truth function we use for conjunction is now defined as:

$$T'(a,b) = \begin{cases} max(T(a,b),t), & \text{if } min(a,b) > 0.5 \\ T(a,b) & \text{otherwise} \end{cases}$$

where T denotes an arbitrary standard t-norm. This function together with the associated truth function for disjunctions ($S'(a,b) = 1 - T'(1-a, 1-b)$) ensures that formulas that are true always get a value greater or equal t and formulas that are false get a value smaller or equal $1 - t$. Thus the values of different formulas stay comparable. This is necessary if there are multiple goals in the game. The disadvantage is that T' is not associative, at least in cases of continuous t-norms T, and is therefore not a t-norm itself in general. The effect of this is that the evaluation of semantically equivalent but syntactically different formulas can be different. However by choosing an appropriate t-norm T it is possible to minimize that effect.

For the t-norm T we use an instance of the Yager family of t-norms:

$$\begin{aligned} T(a,b) &= 1 - S(1-a, 1-b) \\ S(a,b) &= (a^q + b^q)^{\frac{1}{q}} \end{aligned}$$

The Yager family of t-norms captures a wide range of different t-norms. Ideally we would want a heuristic and thus a pair of t-norm and t-co-norm that is able to differentiate between all states that are different with respect to the goal and terminal formulas. In principal this is possible with the Yager family of t-norms. By varying q one can choose an appropriate t-norm for each game, depending on the structure of the goal and terminal formulas to be evaluated, such that as many states as possible that are different with respect to the goal and terminal formulas are assigned a different value by the heuristic function. Because $S(a,b) >= max(a,b)$ and S is monotonic in a and b, the more disjunctions occur in a goal and terminal formulas the smaller q has to be in order for the evaluation of the formula not to yield 1 for a lot of states, although the states are quite different with respect to the goal. On the other hand q has to be bigger the more conjunctions occur in the formulas, otherwise the value of the formula will be 0 for a lot of the states. However, at the moment we just use a fixed value for q which seems to work well with most games currently available on `http://games.stanford.edu`, i.e. the evaluation function is able to differentiate a wide variety of different states on most of the games.

Our evaluation function $eval(f, z)$ for evaluating the state z with respect to the formula f is defined as follows(a denotes

an atom, f and g are arbitrary formulas):

$$\begin{aligned} eval(a,z) &= \begin{cases} p, \text{if } a \text{ holds in the current state} \\ 1-p, \text{otherwise} \end{cases} \\ eval(f \wedge g, z) &= T'(eval(f,z), eval(g,z)) \\ eval(f \vee g, z) &= S'(eval(f,z), eval(g,z)) \\ eval(\neg f, z) &= 1 - eval(f,z) \end{aligned}$$

According the the definitions above, the evaluation function has the following property, which shows its connection to the logical formula:

$$\begin{aligned} (\forall f, z)\, eval(f,z) \geq t > 0.5 &\iff holds(f,z) \\ (\forall f, z)\, eval(f,z) \leq 1-t < 0.5 &\iff \neg holds(f,z) \end{aligned}$$

Where $holds(f,z)$ denotes that formula f holds in state z.

The heuristic function for a state z in a particular game is defined as follows:

$$\begin{aligned} h(z) &= \frac{1}{\sum_{gv \in GV} gv} * \bigoplus_{gv \in GV} h(gv,z) * gv/100 \\ h(gv,z) &= \begin{cases} eval(goal(gv) \vee term, z), \text{if } goal(gv) \\ eval(goal(gv) \wedge \neg term, z), \text{if } \neg goal(gv) \end{cases} \end{aligned}$$

$\bigoplus$ denotes a product t-co-norm sum, GV is the domain of goal values, $goal(gv)$ is the (unrolled) goal formula for the goal value gv and $term$ is the (unrolled) terminal formula of the game. That means the heuristics of a state is calculated by combining heuristics $h(gv,z)$ for each goal value gv of the domain of goal values GV weighted by the goal value gv with a product t-co-norm (denoted by $\bigoplus$). The heuristics for each possible goal value is calculated as the evaluation of the disjunction of the goal and terminal formulas in case the goal fullfilled, i.e. the heuristics tries to reach a terminal state if the goal is reached. On the other hand the heuristics tries to avoid terminal states as long as the goal is not reached.

5 Identifying Structures

The evaluation function described above can be further improved by using the whole range of values between 0 and 1 for atoms instead of assigning $1-p$ for false and p for true atoms.

The idea is to detect structures in the game description which can be used for non-binary evaluations like successor relations, order relations, quantities or game boards. The approach is similar to the one of [Kuhlmann *et al.*, 2006] but with some differences.

Static structures, i.e. structures that are independent of the state of the game, like successor and order relations are detected by checking certain properties of all relations that do not depend on any instance of $true(?x)$. E.g., binary relations that are antisymmetric, functional and injective and whose graph representation is acyclic are considered as successor relations. Order relations are binary relations that are antisymmetric and transitive. Those properties can be proved quite easily as all domains are finite. Unlike [Kuhlmann *et al.*, 2006], who use syntactical structure of the rules to detect e.g., successor relations, we use semantical properties of the predicates. In addition to [Kuhlmann *et al.*, 2006] we can also detect higher level predicates like order relations. This is difficult to do when relying on the syntax of the rules, because there are many semantically equivalent but syntactically different descriptions of a predicate.

Dynamic structures like game boards and quantities are detected in the same way as described in [Kuhlmann *et al.*, 2006]. However our definition of a game board is broader in that we don't restrict ourselves to two-dimensional boards with just one argument describing the content of the board's cells.

For us a game board is a multi-dimensional grid of cells that have a state which can change. Each fluent in the game description with at least 2 arguments is a potential board. Some of the arguments must identify the cell of the board, those are the coordinates. The remaining arguments form the cell's state. They must have a unique value for each instance of the coordinate arguments in each state of the game. The coordinates of the cell are input arguments and the remaining arguments are output arguments of the fluent describing the board. A board is ordered if the coordinate arguments are ordered, i.e. there is a successor relation for the domain of the arguments. If only some of the coordinate arguments are ordered then the fluent possibly describes multiple ordered boards. An example for this case is the fluent $cell(?b, ?y, ?x, ?c)$ in the Racetrack game (the final game in the AAAI GGP Competition 2005), a two-player racing game where each player moves on his own board. The arguments $?y$ and $?x$ are the coordinates of the cell on board $?b$ and $?c$ is the content of the cell. The domains of $?x$ and $?y$ are ordered but the domain of $?b$ is not, all three of $?x$, $?y$ and $?b$ are the input arguments of the fluent $cell(?b, ?y, ?x, ?c)$.

A unary fluent describes a quantity if it's argument is ordered and the fluent is a singleton. A singleton fluent is a fluent without input arguments, that means the fluent occurs at most once in every state of the game. A prominent example is the step counter $step(?x)$ occurring in many games, which describes a quantity, namely the number of steps executed so far in the game. Also states of board cells can be quantities if they are ordered. E.g., the fluent $money(?player, ?amount)$ of the game Farmers (a three player economic game), which describes the amount of money each player has in a certain state, is identified as a one-dimensional unordered board with one cell for each player where the content of each cell is a quantity.

For deciding if a fluent describes a board or a quantity we need to know the fluent's input and output arguments as well as the information if a certain argument of a fluent is ordered. Similar to the method described in [Kuhlmann *et al.*, 2006], input and output arguments of all fluents are approximately calculated by generating hypotheses and checking them in states generated by random play until the hypotheses are stable. We start with an empty set of input arguments as hypothesis and add one argument whenever the hypothesis turns out not to be true in a state. Usually the hypotheses are stable and correct after very few states (3 to 5). This simulation stage is also used to check other properties of the game like if the game is turn-based. For the decision if an argument of a fluent is ordered it is necessary to know the domains of the ar-

guments of fluents and if there is a successor relation for this domain. The domains, or rather supersets of the domains, of all predicates and functions of the game description are computed by generating a dependency graph from the rules of the game description. The nodes of the graph are the arguments of functions and predicates in game description and there is an edge between two nodes whenever there is a variable in a rule of the game description that occurs in both arguments. Connected components in the graph share a (super-)domain.

The dependency tree in figure 1 is the one for the following game rules describing a successor predicate and an unary fluent `step` which is incremented every step:

```
(succ 0 1) (succ 1 2) (succ 2 3)
(init (step 0))
(<= (next (step ?x))
    (true (step ?y)) (succ ?y ?x))
```

The domains of the arguments of the function `step` and the predicate `succ` are all connected in the graph and thus share the same domain, which consists of the 4 connected constants `0` to `3`. The computed set of constants is actually a superset of the real domain of the arguments of `succ`. The domain of the first argument of `succ` doesn't contain `3` and the domain of the second argument doesn't contain `0`. But we disregard this fact because we are more interested in the dependencies between different functions and predicates than in the actual domains.

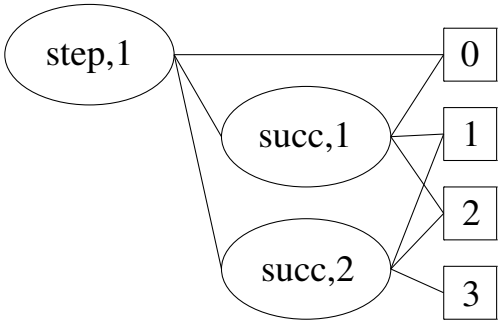

Figure 1: A dependency graph for calculating domains of functions and predicates. (Ellipses denote arguments of fluents or predicates and squares denote constants.)

6 Using identified structures for the heuristic function

The heuristic evaluation function is improved by introducing non-binary evaluations of the atoms that correspond to identified structures.

The evaluation of an order relation r is computed as

$$eval(r(a,b),z) = \begin{cases} t + (1-t) * \frac{\Delta(a,b)}{|dom(r)|}, \text{if } r(a,b) \\ (1-t) - (1-t) * \frac{\Delta(b,a)}{|dom(r)|}, \text{if } \neg r(a,b) \end{cases}$$

Where $\Delta(a,b)$ is the number of steps needed for reaching b from a with the successor function that is the basis of this order relation and $|dom(r)|$ denotes the size of the domain of the arguments of r. This evaluation function, which is used in games like Othello (the goal is to have more pieces than the opponent) or Farmers (the goal is to have more money as the opponents), is reasonably fast to compute and has advantages over a binary evaluation which just reflects the truth value of the order relation: It prefers state with a narrow miss of the goal over states with a strong miss. E.g., for the white player in an Othello game it prefers a state with 15 white pieces and 16 black ones over a state with 10 white and 21 black pieces although both are on the losing side at the moment. However the chances of white winning the game continuing with the first state are probably much higher. The same holds for states on the winning side, where the evaluation function prefers strong wins over narrow ones.

The evaluation of atoms of the form `(true f)` can be non-binary if the `f` in question describes an ordered game board or a quantity. For ordered game boards that evaluation reflects the distance (computed with a normalized city-block metrics) of the position of a piece on the board in the current state to the goal position. E.g., the evaluation of the atom `(true (cell wlane e ?x white))`, which occurs in the goal definition of the Racetrack game, in a state which contains `(true (cell wlane b 3 white))` is based on the (normalized) Manhattan distance between the coordinates of the white piece in both states. The distance for the third coordinate is assumed to be zero because it is a variable. The distances for each single coordinate are normalized by the size of the coordinates domain. If there are multiple matching pieces in a state like in Chinese Checkers where the pieces of each player are indistinguishable, the evaluation is based on the mean value of the distances for all matching pieces.

E.g., the evaluation function for a two-dimensional ordered board with the coordinates x and y is defined as follows:

$$eval(f(x,y,c),z) = \frac{1}{N} * \sum_{f(x',y',c) \in z} \frac{1}{2} * \left(\frac{\Delta(x,x')}{|dom(f,1)|} + \frac{\Delta(y,y')}{|dom(f,2)|} \right)$$

Where N is the number of occurrences of $f(x',y',c)$ in z for arbitrary x', y' and the given c, $|dom(f,i)|$ denotes the size of the domain of the i-th argument of f and $\Delta(x,x')$ ($\Delta(y,y')$) is the number of steps between x and x' (y and y') according to the successor function that induces the order for the domain. The function can easily be extended to arbitrary dimensional boards with an arbitrary number of output arguments.

If the fluent `f` of the atom `(true f)` is a quantity (or a board where the cells state is a quantity), the evaluation is based on the difference between the quantity in the current state and the quantity in the atom to evaluate. This evaluation is effectively used in e.g., many games that use a step counter. In games that end when the step counter reaches a certain limit the evaluation of the terminal rule yields higher values for higher step counters. The effect of this evaluation is that of two states that are equal except for the step counter the one with the smaller step counter is preferred as long as the goal is not fullfilled. This coincides with a quite common practice in planning problems, where shorter plans are pre-

ferred over longer ones. E.g., the evaluation function for and unary quantity fluent f like the step counter is:

$$eval(f(x), z) = \frac{\Delta(x, x')}{|dom(f, 1)|}, \text{if } f(x') \in z$$

This function can be extended easily to an arbitrary (non-zero) number of input and output arguments.

7 Evaluation

Due to the lack of freely available general game playing systems at this time it is not possible to make thorough comparisons of our approach with others. One noteworthy exception is the annual AAAI GGP Competition. In the first GGP Competition at AAAI'05 our game player with the name FLUX-PLAYER performed reasonably well and it is the winner of the AAAI GGP Competition 2006. It is also not easily possible to compare the performance of our general game player with existing game playing systems for specific games like Chess or Chinese Checkers, because the performance of our systems highly depends on the description of the game rules, which can be arbitrarily complex for the same game.

However, based on comparisons with human players and analysis of the results of the heuristic evaluation function for different games, it is possible to give an assessment which types of games our game player is able to play well and which games it has problems with. The informal notion of "playing a game well" describes that the automatically constructed heuristics captures the purpose of the game as a human understands it. That means the heuristics yields a higher value for states that seem to be advantageous as perceived by a human. It is clear that this might not be the best choice of a heuristics. Thorough experiments yielding quantitative results are needed to be able to compare our approach to others, this is work in progress.

Because our approach is highly based on the goal description our player performs well in all games where the goal description yields a meaningful heuristics. Those include goal descriptions that describe concrete goal states preferably with little interdependencies between the subgoals. Examples for this are games like 8-Puzzle or Blocks World, where certain configurations of the pieces have to be reached and actions typically change only a small set of fluents. Other examples are games with gradual goal values (goal values between 0 and 100 for reaching only part of the goal) like the 10-Queens problem where the goal value is lower the more queens attack each other or Crisscross, a two-player variant of Chinese Checkers, where the goal value depends on the number of pieces moved into their goal position.

All types of games where the goal value depends on reaching a certain location on a board or reaching a certain amount of some quantity are usually played well, if it is possible to gradually improve the location or the amount over time. Examples are Chinese Checkers, Racetrack and other racing games as well as Othello or Farmers. However some features necessary for very good performance are missing. E.g., our player is only able to recognize the worth of a corner square in Othello if he is able to search to a certain depth. The same holds for buying farms or factories in the game of Farmers, a game with the purpose of buying, producing and selling goods to make profit. That means certain advantageous moves are only made when the search is deep enough to see the advantage if they do not directly improve the goal evaluation (occupying a corner square in Othello) or even deteriorate it at first (buying a farm or factory in Farmers decreases your amount of money at first).

Like in all search based approaches the performance depends on the depth of the search, which is low if the branching factor of the game tree is high. This doesn't seem to be a problem with some games like the 10-Queens problem (branching factor $b \approx 100$), where the heuristics and the non-uniform depth first search results in good play even with a search depth of only 1 or 2 initially. High branching factors are a problem in games like Farmers ($b \approx 200$) where a deeper search is necessary for good play.

Other game descriptions that are problematic are those with complex goal descriptions, that are expensive to evaluate and, on the other hand, goal descriptions that do not help to differentiate non-terminal states. An example for the latter category is Peg Jumping a game with the goal to remove all pegs from the board by jumping over them. The goal description consists only of the goal position of the last peg and the atom `(true (pegs s1))`, which means that there must be only one peg left. That means the evaluation of a state regarding the goal depends almost exclusively on the number of pegs currently on the board, which is the same for all states of the same depth in the game tree. Two facts are the reason that our game player plays this game well nonetheless: First, the game ends if there is no possible move left. Thus the terminal evaluation corresponds to a mobility heuristics, which helps avoiding dead ends. And second, the very simple goal description can be evaluated very fast which allows deep searches.

8 Discussion

The evaluation function obtained by the method described above is directly based on the goal description of the game. The generation of the heuristics is much faster than learning-based approaches. Thus our approach is particularly well suited for games with a big state space and sparse goal states. Those are typical properties of almost all singleplayer games and some multi-player games like Chess. In both cases with learning based approaches one has to play many random matches to have significant data. Based on the same rationale our approach has advantages for games with little time for learning parameters of the heuristic function.

However depending on the game description, our evaluation function can be more complex to compute than learned features or it might not be able to differentiate between non-terminal states at all. Decomposition and abstraction techniques like they are described in [Fawcett and Utgoff, 1992] might be used to overcome the first problem. We want to tackle the second problem by formal analysis of the rules of the game for discovering relations between different properties of a state and thereby detecting features that are important for reaching the goal.

Another advantage of our approach regarding possible fu-

ture extensions of the general game playing domains is, that it is directly applicable to domains with incomplete information about current position. Flux, which is used for reasoning, as well as the heuristic evaluation function are in principle able to handle incomplete information games.

9 Related Work

A work in the same setting is [Kuhlmann *et al.*, 2006]. The authors work on the same problem but use a different approach for heuristics construction. A set of candidate heuristics are constructed, which are based on features detected in the game, but don't take the goal description into account. The process of selecting which of the candidate heuristics is leading to the goal remains an open challenge.

Previous work on general game playing includes Barney Pell's Metagamer [Pell, 1993] which addresses the domain of Chess-like board games. Because the domain representation differs from the GDL it is unclear if the techniques can be directly applied to GGP.

Fawcett [Fawcett, 1993] applies a feature discovery algorithm to a game of Othello. Features are generated by inductively applying transformation operators starting with the goal description. Discovered features need to be trained. The method uses a STRIPS-style domain but may possibly be applied to GGP.

Much research has been done on heuristic-search planning [Bonet and Geffner, 2001]. However, most of the work depends on STRIPS-style domain descriptions. It is not clear how much of the techniques can be applied to GGP. It is also unknown if the methods can be adapted for multi-player games.

10 Conclusion

We have presented an approach to General Game Playing which combines reasoning about actions with heuristic search. The main contribution is a novel way of constructing a search heuristics by the automated analysis of game specifications. Our search heuristics evaluates states with regard to the incomplete description of goal and terminal states given in the rules of the game. It takes advantage of features detected in the game like boards, quantities and order relations.

Our General Game player performed well in a wide variety of games, including puzzles, board games, and economic games. However there is ample room for improvement: We are currently working on enhancing the translation of the game description to Prolog in order to make reasoning and especially the calculation of successor states more efficient and therefore the search faster. There is also work in progress directed at an improved evaluation of non-leaf nodes of the search tree, which uses methods from game theory and allows better opponent modeling.

A number of problems of the heuristics need to be addressed. This includes a more efficient evaluation of complex goal descriptions which might be achieved by using abstraction and generalization techniques to focus on important features and disregard features which are less important but expensive to compute.

We plan to use formal analyses of the game description to get a more thorough picture of the function of each part of the game description and their connections. This information can then be used to increase the efficiency of the theorem prover as well as detecting new features of the game which might improve the heuristics.

For directing future research we need to analyse the impact of the different evaluation functions on the game playing performance. For this analysis it is necessary to define realistic opponent models and problem sets. Ideally, we would want to have a range of benchmark domains and opponents such that objective comparisons between different general game playing systems are possible.

References

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Fawcett and Utgoff, 1992] Tom Elliott Fawcett and Paul E. Utgoff. Automatic feature generation for problem solving systems. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Conference on Machine Learning*, pages 144–153. Morgan Kaufmann, 1992.

[Fawcett, 1993] T. Fawcett. Feature discovery for inductive concept learning, 1993.

[Kuhlmann *et al.*, 2006] Gregory Kuhlmann, Kurt Dresner, and Peter Stone. Automatic heuristic contruction in a complete general game player. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, July 2006. To appear.

[Morris, 1997] Robert Morris, editor. *Deep Blue Versus Kasparov: The Significance for Artificial Intelligence*. AAAI Press, 1997.

[Pell, 1993] B. Pell. Strategy generation and evaluation for meta-game playing, 1993.

[Russell and Norvig, 2003] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice-Hall, 2003.

[Schaeffer *et al.*, 2005] Jonathan Schaeffer, Yngvi Björnsson, Neil Burch, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Solving checkers. In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 292–297, Edinburgh, UK, August 2005.

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

[Shannon, 1950] Claude Shannon. Programming a computer for playing chess. *Philosophical Magazine 7*, 41(314):256–275, 1950.

[Thielscher, 2005] Michael Thielscher. *Reasoning Robots: The Art and Science of Programming Robotic Agents*, volume 33 of *Applied Logic Series*. Kluwer, 2005.