Delete Relaxation and Traps in General Two-Player Zero-Sum Games

Thorsten Rauber and **Denis Müller** and **Peter Kissmann** and **Jörg Hoffmann**
Saarland University, Saarbrücken, Germany
{s9thraub, s9demue2}@stud.uni-saarland.de, {kissmann, hoffmann}@cs.uni-saarland.de

Abstract

General game playing (GGP) is concerned with constructing players that can handle any game describable in a pre-defined language reasonably well. Nowadays, the most common approach is to make use of simulation based players using UCT. In this paper we consider the alternative, i.e., an Alpha-Beta based player. In planning, delete relaxation heuristics have been very successful for guiding the search toward the goal state. Here we propose evaluation functions based on delete relaxation for two-player zero-sum games.

In recent years it has been noted that UCT cannot easily cope with shallow traps, while an Alpha-Beta search should be able to detect them. Thus, a question that arises is how common such traps are in typical GGP benchmarks. An empirical analysis suggests that both cases, relatively few traps and a high density of traps, can occur. In a second set of experiments we tackle how well the Alpha-Beta based player using the proposed evaluation function fares against a UCT based player in these benchmarks. The results suggest that (a) in most games with many traps Alpha-Beta outperforms UCT, (b) in games with few traps both players can be on par, (c) the evaluation functions provide an advantage over a blind heuristic in a number of the evaluated games.

Introduction

Game playing has always been an important topic in artificial intelligence. The most well-known achievements are likely the successes of specialized game players such as DeepBlue (Campbell, Hoane, and Hsu 2002) in Chess or Chinook (Schaeffer et al. 1992) in Checkers, defeating the human world-champions in the respective games. However, these specialized players have deviated far from the original idea of a general problem solver (Newell and Simon 1963).

In 2005 this idea was picked up again, by introducing a new competition for promoting research in general game playing (Genesereth, Love, and Pell 2005). Here the players are not supposed to play only a single game on world-class level, but rather to be able to handle any game that can be described in a given language and play it reasonably well. Most research in this area has been invested in deterministic games of full information.

After early successes of players based on Alpha-Beta search with automatically generated evaluation functions (e.g., (Clune 2007; Schiffel and Thielscher 2007)), a new trend dominates the field: the use of UCT (Kocsis and Szepesvári 2006). This is a simulation-based approach, i.e., lots of games are simulated and the achieved rewards propagated toward the root of a partial game-tree, in order to decide on the best move to take. Since 2007 all winners of the international competition have made use of this technique (e.g., (Björnsson and Finnsson 2009; Méhat and Cazenave 2011)). However, for certain games it has been shown that UCT is not always the best choice. One property that is difficult to handle by that approach is the presence of shallow traps (Ramanujan, Sabharwal, and Selman 2010), i.e., states from which the opponent has a winning strategy of short length. While Alpha-Beta can identify such traps, UCT typically can not, at least if the branching factor is high enough or the possible playouts within the trap are long enough.

The basic setting of general game playing is comparable to that of action planning. There the aim also is to implement solvers that can handle any planning task describable in the given language. The current trend in planning is in heuristic search, where the heuristics are automatically generated at run-time. One successful approach is based on delete relaxation, e.g., the FF heuristic (Hoffmann and Nebel 2001). In the delete relaxed setting, anything that once was true remains true. The length of a plan (i.e., a solution) for a delete relaxed planning task can then be used as an estimate for the length of a plan in the original setting.

In this paper we propose new evaluation functions for general game playing based on the length estimates derived by delete relaxation heuristics and apply these evaluation functions in an Alpha-Beta implementation. Furthermore, we empirically evaluate a number of games to get an idea of their trap density. In the experimental results we will see that our Alpha-Beta based player indeed outperforms a UCT based player in most tasks that contain a large amount of shallow traps and is on-par in several of the games with fewer traps. Additionally, the use of the evaluation function brings a real advantage over a blind heuristic in a number of the evaluated games.

Background

In this section we provide the necessary background on general game playing, UCT search, traps in games and delete relaxation heuristics as they are used in planning. We assume the reader to be familiar with the basics of Alpha-Beta search, so that we skip an introduction.

General Game Playing

The main idea of general game playing (GGP) is to implement players that play any game that can be described by the given language reasonably well. The current setting as it was introduced for the first international competition in 2005 (Genesereth, Love, and Pell 2005) allows for a wide range of games: single-player puzzles or two- and multi-player games, which can be, among others, turn-taking or with simultaneous moves, zero-sum or more general rewards, cooperative, etc. In all settings the goal for each player is to maximize its own reward. Furthermore, all these games are finite, discrete, deterministic, and all players have full information.

In this paper we consider only the case of strictly turn-taking two-player zero-sum games. The two players are denoted Max (the starting player) and Min. As possible outcomes we allow only win (here denoted 1), loss (denoted -1), and draw (denoted 0) from the Max player's point of view. Basically, our definition of a game is an extension of the multi-agent STRIPS setting (Brafman and Domshlak 2008) to adversarial agents:

Definition 1. *A game is a tuple* $\Pi_G = \langle V, A_{Max}, A_{Min}, I, G, R\rangle$*, where* V *is the set of state variables or facts,* A_{Max} *and* A_{Min} *are the actions of the Max and Min player, respectively,* I *is the initial state in form of a complete assignment to* V*,* G *is the termination criterion in form of a partial assignment to* V*, and* R *is a function assigning a reward in* $\{-1, 0, 1\}$ *to each terminal state. Similar to planning, an action* a *is of the form* $\langle pre_a, add_a, del_a\rangle$*, where* pre_a *is the precondition,* add_a *the list of add-effects, and* del_a *the list of delete-effects.*

Note that this definition deviates from the commonly used game description language GDL (Love, Hinrichs, and Genesereth 2008), where the effects of an action specify all facts that are true in the successor state, which together with the closed world assumption results in a full state specification.

Definition 2. *For a game* $\Pi_G = \langle V, A_{Max}, A_{Min}, I, G, R\rangle$*, the semantics are defined by means of a transition system* $\Theta_G = \langle S, L, T, I, S_g^{-1}, S_g^0, S_g^1\rangle$*, where* $S = S_{Max} \cup S_{Min}$ *is the finite set of all states,* S_{Max} *the set of states where Max has to move and* S_{Min} *the set of states where Min has to move.* $L = L_{Max} \cup L_{Min}$ *is a finite set of labels with* $L_{Max} = A_{Max}$ *and* $L_{Min} = A_{Min}$*.* $T = T_{Max} \cup T_{Min}$ *is a set of transitions consisting of* $T_{Max} \subseteq S_{Max} \times L_{Max} \times S_{Min}$ *and* $T_{Min} \subseteq S_{Min} \times L_{Min} \times S_{Max}$*. Precisely,* $(s, l, s') \in T_{Max}$ *if* l *is applicable in* s *(i.e., if* $s \in S_{Max}$*,* $l \in L_{Max}$*,* $pre_l \subseteq s$*), and* $s' \in S_{Min}$ *is the resulting successor state (i.e., if* $s' = (s \setminus del_a) \cup add_a$*); similar for* T_{Min}*.* $I \in S_{Max}$ *is the initial state.* $S_g^{-1} \subseteq S$ *is the set of terminal states lost for Max,* $S_g^0 \subseteq S$ *the set of terminal draw states, and* $S_g^1 \subseteq S$ *the set of terminal states won by Max, i.e.,* $s \in S_g^r$ *if* $G \subseteq s$ *and* $R(s) = r$*. The Max player tries to maximize the reward while Min tries to minimize it.*

Upper Confidence Bounds Applied to Trees

The upper confidence bounds (UCB1) algorithm (Auer, Cesa-Bianchi, and Fischer 2002) is used in the area of multi-armed bandits and aims at maximizing the expected reward. The upper confidence bounds applied to trees (UCT) algorithm (Kocsis and Szepesvári 2006) is an extension of this to tree based searches. It treats every internal node as a multi-armed bandit (where the different arms correspond to the possible actions to take) and tries to learn which actions are preferable. UCT consists of four phases: selection, expansion, simulation, and backpropagation.

In the selection phase nodes stored in the UCT tree are evaluated and a path is followed until a leaf of that tree is reached. The evaluation works as follows. Let s be the state represented by a node, $a_1, \ldots, a_n$ the actions applicable in state s, $s_1, \ldots, s_n$ the corresponding successor states, $n(s)$ the number of times state s was reached, $n(s, a)$ the number of times that action a was chosen in state s, and $\delta(s)$ the average reward achieved when starting in state s. The UCT value for the different actions is defined as

$$UCT(s, a_i) = \delta(s_i) + C\sqrt{\frac{\log n(s)}{n(s, a_i)}}\,. \qquad (1)$$

In our two-player setting, if $s \in S_{Max}$, we can use this directly and select the action achieving the highest UCT value; if $s \in S_{Min}$, we use the negation of $\delta(s_i)$ in equation (1) and still select the action achieving the highest UCT value. The constant C is used to set the amount of exploration or exploitation: With a small value the algorithm will tend to exploit the knowledge already generated by mainly following the most promising actions, while with a high value the algorithm will tend to explore different areas, often selecting actions that lead to less promising successors.

If a state contains some unexplored successors, instead of evaluating the UCT formula one of the unexplored successors will be selected randomly. This assures that the formula is evaluated only if initial values for all successors are set.

The expansion phase starts when a leaf node of the UCT tree has been reached. In that case the leaf node will be expanded, the successor added to the tree, and the simulation phase starts.

In the simulation phase a Monte-Carlo run is started, which chooses among the applicable actions of the current state at random until a terminal state is reached.

When this happens, the backpropagation starts. This updates the average rewards and counters of all states visited during the selection phase. As soon as all nodes are updated the selection phase starts over at the root of the UCT tree.

When an actual action is to be performed and we are the Max (Min) player, the action leading to the successor with highest (smallest) average will be selected and the corresponding successor node will become the new root of the UCT tree. If it is not our turn to move in the current state, we wait for the action chosen by the opponent and take the corresponding successor as the new root. Afterwards the search starts over at the new root node.

Traps in Games

In some games such as Go traditional Alpha-Beta based players are clearly inferior to UCT based players like MoGo (Gelly and Silver 2008). In others like Chess, however, the Alpha-Beta based players are on a level beyond any human

world-champion and clearly outperform UCT based players. One explanation for this behavior was provided by Ramanujan, Sabharwal, and Selman (2010), who noticed that Chess contains shallow traps while Go does not.

A state s is called at risk if for the current player p there is a move m so that the corresponding successor s' is a state in which the opponent of p has a winning strategy. If the winning strategy has a maximum of k moves, we call state s' a level-k search trap for p.

A trap is considered to be shallow if it can be identified by Alpha-Beta search. Due to its exhaustive nature, this is the case if its depth-limit is at least $k+1$. Such a player can avoid falling into the trap by using a move different from m in state s. Typically, traps of level 3–7 are considered to be shallow. UCT often cannot identify such traps as it spends a lot of time exploring states much deeper than the level of the trap, in areas it considers promising.

In a subsequent study Ramanujan, Sabharwal, and Selman (2011) created a synthetic game in which they could manually set the density of traps. With this they found that without any traps, UCT was much better than Minimax. With only few traps in the state space UCT was still better than Minimax. However, the higher the density of traps the worse UCT performed in comparison to Minimax.

Delete Relaxation Heuristics

The setting in planning is similar to ours, with the exception that planning allows only for a single agent. Additionally, the aim of this agent is to reach a terminal state in as few steps as possible. Other than that, especially the handling of actions is the same in both formalisms.

The delete relaxation corresponds to the idea of ignoring the delete effects. That means, everything that once was true will remain true forever. This allows the calculation of a fixpoint of those facts that can become true at any point in the future and of those actions that may be applicable at some point in the future.

One way to do so is by means of the relaxed planning graph (RPG). The RPG consists of alternating layers of facts and actions. The first layer contains all those facts currently true. Then follows a layer with all the actions that are applicable based on those facts. The next layer contains all facts true in the previous layers and the ones added by the actions of the previous layer. This continues until a fixpoint is reached or all facts of a specified goal state are present in the last generated layer. Instead of generating the full RPG, it often suffices to store, for each action and each fact, the first layer it appeared in.

The optimal relaxation heuristic h^+ gives the minimal number of actions needed to reach a given goal state from the current state in the relaxed setting, which is an admissible (i.e., not overestimating) heuristic for the original non-relaxed search problem. As this is NP-hard to calculate approximations are used, e.g., the FF heuristic (Hoffmann and Nebel 2001). After generating the RPG, it marks all facts of the specified goal. Then it works in a backpropagation manner through the RPG, starting at the last generated layer. For each marked fact newly added in this layer it marks an action that adds it. Given these newly marked actions it marks all facts in their preconditions. This continues until the first layer of the RPG is reached. At that point, the marked actions correspond to a solution plan of the relaxed task, and their number is returned as an approximation of the h^+ heuristic.

Delete Relaxation in GGP

In order to evaluate non-terminal states we propose the following new approach based on delete relaxation heuristics. Similar to automated planning, we can define the delete relaxation of a game:

Definition 3. *For a game $\Pi_G = \langle V, A_{Max}, A_{Min}, I, G, R\rangle$, we denote its delete relaxation as $\Pi_G^+ = \langle V, A_{Max}^+, A_{Min}^+, I, G, R\rangle$ where $A_{Max}^+ = \{\langle pre_a, add_a, \emptyset\rangle \mid \langle pre_a, add_a, del_a\rangle \in A_{Max}\}$ (similar for A_{Min}^+).*

Given a state s, we use the FF heuristic (Hoffmann and Nebel 2001) operating on the full set of actions $A^+ = A_{Max}^+ \cup A_{Min}^+$ to estimate the number of moves needed to reach a state with reward 1, denoted as $l_{win}(s)$, and to estimate the number of moves needed to reach a state with reward -1, denoted as $l_{lose}(s)$. Each of these values is set to ∞ if no corresponding terminal state is reachable anymore. We define the evaluation function $h_1(s)$ of state s as

$$h_1(s) = \begin{cases} 1 & \text{if } l_{win}(s) \neq \infty \text{ and } l_{lose}(s) = \infty \\ -1 & \text{if } l_{win}(s) = \infty \text{ and } l_{lose}(s) \neq \infty \\ 0 & \text{if } l_{win}(s) = \infty \text{ and } l_{lose}(s) = \infty \\ \frac{l_{lose}(s) - l_{win}(s)}{\max(l_{lose}(s), l_{win}(s))} & \text{otherwise.} \end{cases}$$

If only one player's winning goal states cannot be reached anymore we treat the state as being won by the opponent. Otherwise the quotient results in a value in $[-1, 1]$. If it takes more moves to reach a lost state the Max player seems to have a higher chance to win, so that the evaluation will be greater than 0; otherwise the Min player seems to have a better chance, resulting in a value smaller than 0.

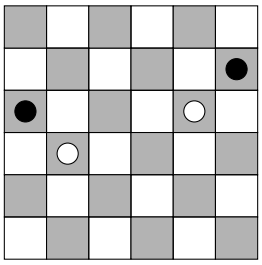

Figure 1: Example state of the game Breakthrough.

Example 1. *As an example we take the game Breakthrough. In this game, white starts with two rows of pawns at the bottom and black with two rows of pawns at the top. The pawns may only be moved forward, vertically or diagonally, and can capture the opponent's pawns diagonally. The goal of each player is to reach the other side of the board with one of their pawns. Consider the state given in Figure 1. Assume that white is the Max player and black the Min player, and in the current state it is white's turn to move.*

In the following we will evaluate the states reached by applying the two different capturing moves. The first one captures with the most advanced white pawn, resulting in state s_1, the second one captures with the least advanced white pawn, resulting in state s_2. In s_1 white needs at least one more move, while black needs at least 3 more moves, so that $h_1(s_1) = (3-1)/3 = 0.67$, indicating an advantage for the white player. In s_2 the white player will need at least two moves and the black player at least four moves to win the game, so that $h_1(s_2) = (4-2)/4 = 0.5$, which indicates a smaller advantage for the white player compared to s_1.

An alternative is to take the mobility into account. A previous approach (Clune 2007) used the mobility directly: The author compared the number of moves of both players, normalized over the maximum of possible moves of both players. While this seems to work well in several games, it bears the danger of sacrificing own pieces in games like Checkers where capturing is mandatory: In such games, bringing the mobility of the opponent to as small a value as possible typically means restricting the opponent to a capture move.

Thus, we do not inspect the mobility in the current state but rather try to identify how many moves remain relevant for achieving a goal. In order to do so we first calculate a full fixpoint of the RPG, i.e., we generate the RPG until no new facts or no new actions are added to it. Only the actions in that graph can become applicable at some time in the future. Now, starting at the facts describing a player's won states, we perform backward search in the RPG, identifying all actions of that player that may lie on a path to those facts. These actions we call relevant.

Let $n_{Max,rel}(s)$ be the number of relevant actions of the Max player in state s and $n_{Min,rel}(s)$ the number of relevant actions of the Min player in state s. Similarly, let n_{Max} be the total number of actions of the Max player and n_{Min} the total number of actions of the Min player. Then we define another evaluation function h_2 for state s as follows:

$$h_2(s) = \frac{n_{Max,rel}(s)}{n_{Max}} - \frac{n_{Min,rel}(s)}{n_{Min}}$$

This assumes that a player has a higher chance of winning if the fraction of still relevant actions is higher for this player than for the opponent.

Example 2. *Consider again the Breakthrough example from Figure 1 and the two successor states s_1 and s_2. In Breakthrough, all moves advance a pawn to the opponent's back row, so that any move that can still be performed at some point is relevant. We distinguish between the (left/right) border cells, where the players have two possible moves, and the inner cells, where the players have three possible moves. Initially, both players have 80 relevant actions (both can reach 10 border cells and 20 inner cells). In s_1, the white player can reach three border cells, and five inner cells, resulting in a total of 21 relevant moves. The black player can still reach three border cells and three inner cells, resulting in a total of 15 relevant moves. Thus, $h_2(s_1) = 21/80 - 15/80 = 6/80$, giving a slight advantage for white. In s_2, the white player can reach three border cells and four inner cells, resulting in a total of 18 relevant moves. The black player can still reach four border cells and six inner cells, resulting in a total of 26 relevant moves. Thus, $h_2(s_2) = 18/80 - 26/80 = -8/80$, indicating a slight advantage for black.*

As a third option, we combine these two evaluation functions to a new function $h_{1+2}(s) = w_1 h_1(s) + w_2 h_2(s)$. Learning weights w_1 and w_2, especially ones optimized for the game at hand, remains future work; in this paper we use a uniform distribution, i.e., $w_1 = w_2 = 0.5$.

Implementation Details

We implemented an Alpha-Beta based player and a UCT based player as well as the new evaluation functions on top of the FF planning system (Hoffmann and Nebel 2001). In this section, we provide some details on the extensions over basic Alpha-Beta and UCT players that we additionally implemented.

Alpha-Beta

In our Alpha-Beta implementation we make use of iterative deepening (Korf 1985), searching to a fixed depth in each iteration and evaluating the non-terminal leaf nodes based on our evaluation function. Between iterations we increase the depth limit by one.

In addition we implemented several extensions found in the literature, among them the use of a transposition table and approaches to order the moves based on results in the transposition table and from previous iterations, which is supposed to result in stronger pruning.

As soon as the evaluation time is up and the player must decide which action to take, the current iteration stops. If it is not this player's turn, nothing has to be done. Otherwise, in case of being the Max (Min) player the action leading to the successor with highest (smallest) value is chosen. The successor reached by the chosen action is taken as the new root of the graph and Alpha-Beta continues.

Quiescence Search On top of this we implemented quiescence search, which tries to circumvent the so called horizon effect. The basic idea is to distinguish between noisy and quiet states, where a state is considered to be noisy in case of tremendous changes in the game with respect to the previous state. As soon as our normal Alpha-Beta search reaches the depth limit we check whether the current state is noisy, and if so, we will switch into quiescence search and continue until we reach a terminal state, a quiet state, or the predefined depth limit of quiescence search.

Our idea for deciding if a state is noisy is to check if the number of moves has drastically changed. Thus, we defined and tested the following criteria:

Applicable actions In each state compute, for the current player, the number of the currently applicable actions and compare it to the value of the previous state where it was this player's turn.

Possible actions In each state compute, for the current player, the number of actions that might still be possible to take later in the game (found by building the RPG

fixpoint, similar to our evaluation function h_2) and compare it to the value of the previous state where it was this player's turn.

In preliminary tests we found that the applicable actions criterion works better than the possible actions criterion. An explanation for this is the overhead induced by computing the RPG fixpoint.

For deciding if a state is noisy, apart from the criterion to check we also need a threshold for deciding if that change corresponds to a noisy state. If the change in the corresponding criterion is greater than the given threshold we consider it to be noisy. We tested threshold values between 5% and 50% and came to the conclusion that 30% is the best value wrt. the applicable actions criterion.

While for some games other values would be better, recall that we consider domain-independent approaches here, so that we cannot choose the most appropriate value for each game in advance. It remains future work to find intelligent ways for adapting this value at run-time depending on the properties of the currently played game.

UCT

For UCT, instead of a tree we generate a graph by using a hash function, similar to the transposition table in Alpha-Beta. In the expansion phase, if we generate a successor state that is already stored in the hash table we take the corresponding existing search node as the child node. While some implementations might make use of parent pointers, thus effectively updating nodes not really visited, we propagate the reached results only along the path of actually visited nodes.

Another extension concerns the use of a Minimax-like scheme in the UCT graph. Similar to the approach proposed by Winands and Björnsson (2011), we mark a node in the UCT graph as solved if it corresponds to a terminal state. During the backpropagation phase we check for every encountered node whether all successors have already been solved. If that is the case, we can mark this node as solved as well and set its value in the Minimax fashion based on the values of its successors. Furthermore, if we are in control in a node and at least one successor is marked as solved and results in a win for us, we mark this node as solved as well and set its value to a win for us. In the selection phase we go through the UCT graph as usual, but stop at solved states and can start the backpropagation phase immediately. Overall, this approach is supposed to bring us the advantage that the values converge much faster and that the runs can become shorter and only within the UCT graph, which prevents the numerous expansions in the simulation phase.

Experimental Results

In this section we start by describing the games we considered in our experiments. Next we point out some insights on traps in those games, along with an empirical evaluation of the trap densities. Finally, we present results of running our Alpha-Beta versions against UCT on that set of games.

Benchmark Games

In the following we will outline the games we used in our experiments.

Breakthrough consists of a Chess-like board, where the two rows closest to a player are fully filled with pawns of their color. The moves of the pawns are similar to Chess, with the exception that they can always move diagonally. The goal is to bring one pawn to the opponent's side of the board or to capture all opponent's pawns.

Chomp consists of a bar of chocolate with the piece in the bottom left corner being poisoned. The moves of the players are to bite at a specified position that still holds a piece of chocolate. The result is that all pieces to the top-right of this are eaten. The player eating the poisoned piece loses the game.

Chinese Checkers is normally played on a star-like grid. In the two-player version we can omit the home bases of the other four players, so that the board becomes diamond-shaped. In each move the players may only move forward (or do nothing), and perform single or double jumps. Each player has three pieces and must move them to the other side of the board, consisting of 5×5 cells. If no player is able to do so in 40 moves the game ends in a draw.

Clobber is played on a rectangular board. The pieces are initially placed alternatingly, filling the entire board. A move consists of moving a piece of the own color to a (horizontally or vertically) adjacent cell with a piece of the opponent's color on it. That piece is captured and replaced by the moved piece of the active player. The last player able to perform a move wins.

Connect Four is a classical child's game. The players take turns putting a piece of their color in one of the columns, where it falls as far to the bottom as possible. The goal is to achieve a line of four pieces of the own color. If the board gets fully filled without one player winning, the game ends in a draw.

Gomoku is played on a square board, where the players take turns placing pieces on empty cells. The first player to achieve a line of five or more pieces of the own color wins the game; if the board is fully filled without any player winning it is a draw.

Knightthrough is very similar to Breakthrough, but here the pieces are knights instead of pawns. The moves are the same as in Chess, with the exception that they may only advance toward the opponent, never move back.

Nim consists of a number of stacks of matches. In each move, a player may remove any number of matches from one of the stacks. The player to take the last match wins the game.

Sheep and Wolf is played on a Chess-like board, where, similar to Checkers, only half the board is used. The sheep start on every second cell on one side, the wolf in the middle on the other side. The wolf moves first. The sheep may move only forward to a diagonally adjacent cell, while the wolf may move forward or backward to a diagonally adjacent cell. The goal of the sheep is to surround the wolf so that it cannot move any more; the goal of the wolf is to either block the sheep or to get behind them.

Game	without traps	trap depth 0	1	2	3	4	5	6	7
Breakthrough (8x8)	662 (119)	0	83	0	114	0	141	?	?
Chinese Checkers	904 (101)	3	5	6	7	4	17	7	47
Chomp (10x10)	14 (14)	4	0	687	0	32	0	263	0
Clobber (4x5)	121 (121)	515	0	23	0	64	0	277	0
Connect Four (7x6)	625 (85)	0	263	0	50	0	28	0	34
Nim (11,12,15,25)	469 (32)	0	40	0	102	0	44	0	345
Nim (12,12,20,20)	435 (41)	0	50	0	98	0	32	0	385
Sheep & Wolf (8x8)	882 (193)	0	11	0	11	0	22	0	74

Table 1: Trap search results for 1000 randomly sampled states, searching for traps of depth up to 7 (exception: Breakthrough only up to 5). The numbers are the depths of the deepest traps found in each state. Additionally, we give the number of states without traps, and for how many of those we can prove that they already are lost anyway (given in parantheses).

Traps in the Benchmark Games

We implemented an algorithm that evaluates games for getting an idea of the density of traps in those games. To do so, we first randomly choose the depth in which to find a state, and then perform a fully random game until this depth. The reached state will be the root of a Minimax tree, which we use to decide whether or not the state is at risk. If we can prove that the state is already lost anyway, there cannot be any trap. Otherwise, if we find some successor state that is provably lost, the root node is at risk, and the lost successor states correspond to traps. The depth of a trap is then the depth of the Minimax tree needed for proving it a lost state.

Table 1 displays the results of performing this approach for 1000 different sampled states and searching for traps of a depth of at most 7. For some games generating the full Minimax subtrees is not feasible. This is true for Gomoku and larger versions of Clobber. For Breakthrough this holds as well, but the algorithm finished when searching only for traps of depth 5 or less. In some games a state is at risk by several traps of different depths; the table gives only the depth of the deepest trap the algorithm identified.

Even though the algorithm did not work out for **Gomoku**, we assume it to contain a large number of shallow traps of at least depth 3. Whenever a player achieves a situation with a line of three pieces in the own color and the two cells on both sides of that line are empty, the opponent is in a state at risk. If the next move is not next to the line of three, a trap is reached as the player may then place a fourth piece adjacent to the existing line so that the adjacent cell on both sides is empty, which is an obvious win. Due to the large branching factor (the default board size is 15×15) and the possibility to continue playing for a long time without actually playing one of the finishing moves these traps are hard to detect by UCT, even though they are rather shallow.

In **Connect Four** the number of shallow traps is likely much smaller than in Gomoku. While it is enough to have a line of two pieces to create a state at risk it is further required that the two cells to each side of such a line must be immediately playable. As such, the surrounding board must be sufficiently filled with pieces. Additionally, a vertical line can not be seen to create a serious threat, as only one side of such a line remains open and due to the small branching factor UCT should have no trouble identifying it.

Situations of *zugzwang*, for which Connect Four is known, might also be considered as traps. However, these traps are not shallow as they typically result in filling several columns until the actual move to end the game can be played.

From the results in Table 1 we can see that most traps are of depth 1, which means that there is a line of 3 pieces of the opponent which it can finish in its next move – this can hardly be considered a serious trap, as it will be easily identified by UCT.

In **Breakthrough** we expected to be confronted with a large number of shallow traps. In a situation where an opponent's pawn is three cells from the current player's side and it is the last chance to take that pawn clearly is a state at risk. While Alpha-Beta will have no trouble identifying this as the game will be lost in five more steps, UCT again has to cope with a rather large branching factor and the fact that the game can continue for a long time if the simulations do not move the pawn that threatens to end the game. However, from the gathered results it seems that traps of depth 5 or less are not as common as we expected, at least in the 8×8 version of the game; only a third of all evaluated states contained such a trap.

For **Knightthrough** we expect that it contains a rather high density of shallow traps. Here a knight may be six cells from the opponent's side in order to need only three more own moves to reach it, so that states at risk can occur much earlier in the game.

The branching factor of Knightthrough is a bit higher than that of Breakthrough, but the length of the game typically is shorter, as the pieces can move up to two cells closer to the opponent's side. As such, the difficulty to identify traps for UCT might be similar to that in Breakthrough played on a board of the same size.

The game **Nim** is easily solved by mathematical methods (Bouton 1901). A winning strategy consists of reacting directly to the opponent's moves. The idea is to encode the stacks as binary numbers and then calculate the bitwise exclusive or of these numbers. If the result is different from $0 \ldots 0$ in the initial state the game is won for the starting player. In fact, the winning player can always counter an opponent's move in such a way that the result will be $0 \ldots 0$. This means that each state is at risk for the supposedly winning player, as a wrong move immediately means that the opponent can follow the same strategy and then ensure a win. However, this results in arbitrarily long games, so that we cannot expect to find many shallow traps easily identified by Alpha-Beta search.

From the results we can see that slightly more than half of the explored states contain traps of a depth of 7 or less. The somewhat surprisingly large number of shallow traps may be explained by the fact that in the tested cases we have only four stacks with relatively few matches, so that the endgame can be reached after only few steps in case of random play.

In **Chomp** every state is at risk: The player to move may choose to take the poisoned piece and thus immediately lose

the game, which corresponds to a trap of depth 0. Alternatively a player may decide to take the piece horizontally or vertically adjacent to the poisoned one. In such a situation the opponent can then take all remaining non-poisoned pieces. This corresponds to a trap of depth 2. However, both situations can hardly be considered as serious threats: In the second case, the branching factor and the maximal depth are rather small. From the evaluation results we see that these are the most common traps and they are the deepest ones for nearly 70% of the evaluated states.

For **Clobber** it is hard to find a general criterion for the presence of traps, so we used only our evaluation of sampled states. In our implementation of this game, a player can always give up and thus lose the game. This explains why we have so many traps of depth 0, which we can disregard as no player should fall for them. Traps of depth 2 are uncommon, starting with depth 4 they become much more common again (though in half the cases the states at risk that contain such a trap also contain one of depth 6). Finally, a quarter of all evaluated states contains a trap of depth 6. This high density of shallow traps might be due to the fact that the game is rather short (it typically ends after slightly more than ten moves); for larger boards (e.g., 5×6) we expect the situation to change and the number of shallow traps to decrease significantly in the early game (the first 8–10 moves).

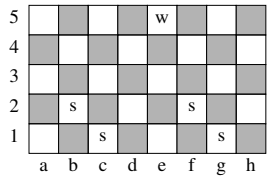

Figure 2: Relevant part of a state at risk in Sheep and Wolf.

For **Sheep and Wolf** consider the situation depicted in Figure 2. The sheep are to move next and the state is at risk. If the sheep on b2 is moved to a3 the wolf cannot be stopped from reaching cell c3. From there only one sheep is left that might stop it from going to b2 or d2, so that the wolf will win. If instead the sheep on c1 would have been moved to d2, the sheep could still win the game. Similar situations also exist with the wolf being closer to the sheep. However, our evaluation of sample states shows that such shallow traps are rather rare throughout the game. In total we found only 118 states at risk, and 74 of those had traps of depth 7.

Chinese Checkers requires to have all own pieces on the other side of the board in order to win. This means that for a trap of depth 7 or less all pieces must be placed in such a way that at most four own moves and/or jumps are required to reach the goal area. Thus, for most parts we are not confronted with any shallow traps.

Results for the Alpha-Beta Based Player

Here we provide results for running our Alpha-Beta players against the UCT player. All experiments were conducted on machines equipped with two Intel Xeon E5-2660 CPUs with 2.20 GHz and 64 GB RAM. Both processes were run on the same machine using one core each. We allowed a fixed amount of 10s for each move and performed a total of 200 runs for each game: 100 runs with UCT playing as Min player, and 100 runs with UCT as Max player.

Game	$\alpha\beta(0)$ vs. UCT	UCT vs. $\alpha\beta(0)$	$\alpha\beta(h_1)$ vs. UCT	UCT vs. $\alpha\beta(h_1)$	$\alpha\beta(h_{1+2})$ vs. UCT	UCT vs. $\alpha\beta(h_{1+2})$	$Q\alpha\beta(h_1)$ vs. UCT	UCT vs. $Q\alpha\beta(h_1)$
Breakthrough (6x6)	-0.22	-0.10	0.92	-1.00	0.80	-0.80	0.72	-0.90
Breakthrough (8x8)	0.30	-0.24	0.92	-1.00	0.94	-0.96	0.94	-0.94
Chinese Checkers	0.00	0.00	0.08	-0.18	0.03	0.00	0.32	-0.05
Chomp (10x10)	-1.00	0.64	-1.00	0.70	-1.00	0.50	-1.00	0.62
Clobber (4x5)	0.64	0.88	0.92	0.88	0.94	0.88	1.00	0.74
Clobber (5x6)	-0.92	0.94	-0.88	0.86	-0.88	0.94	-0.86	0.90
Connect Four (7x6)	-1.00	1.00	-0.86	0.75	-1.00	0.48	-0.91	0.70
Gomoku (8x8)	-0.98	0.98	0.44	0.33	0.69	0.21	0.50	0.49
Gomoku (15x15)	0.46	-0.24	1.00	-1.00	1.00	-0.98	1.00	-1.00
Knightthrough (8x8)	0.82	-0.82	0.72	-0.74	0.76	-0.82	0.70	-0.56
Nim (11,12,15,25)	0.70	-0.68	0.00	0.00	-0.04	0.06	-0.18	-0.12
Nim (12,12,20,20)	0.76	-0.78	0.04	-0.14	0.24	0.00	0.24	-0.34
Sheep & Wolf (8x8)	0.18	-1.00	0.04	-0.98	0.32	-1.00	0.12	-0.98

Table 2: Average rewards for the tested games using Alpha-Beta (six left columns) and Alpha-Beta with quiescence search (last two columns).

Table 2 shows the average rewards achieved when running Alpha-Beta with heuristic h_1 (denoted $\alpha\beta(h_1)$) and h_{1+2} (denoted $\alpha\beta(h_{1+2})$), as well as quiescence search with heuristic h_1 (denoted $Q\alpha\beta(h_1)$). As the results of quiescence search with heuristic h_{1+2} are very similar we omit those. Additionally, we used a blind heuristic (denoted $\alpha\beta(0)$), assigning each non-terminal state a value of 0, in order to show that our heuristics actually provide additional information over the basic trap detection inherent in Alpha-Beta search.

From these results we can make some observations. First of all, for the two evaluation functions and the two Alpha-Beta versions, the differences are surprisingly small. For the heuristics this might be explained by the fact that h_1 is a part of h_{1+2}. For quiescence search a possible explanation might be that the benefit of increased depth in some parts results in shallower depth in others due to the fixed time-out, so that overall both searches perform similar.

Second, the additional information of the new evaluation function provides a significant advantage in the games Breakthrough and Gomoku. While the players using the evaluation functions consistently win especially on larger boards, the player with the blind heuristic performs much worse. Obviously our heuristics are good enough for these games to prevent creating situations from which the player cannot recover, i.e., falling into traps deeper than the depth of the Alpha-Beta search tree. For the smaller version of Clobber and for Connect Four the advantage of using our evaluation functions is not as big but still noticeable. However, in the game of Nim the blind heuristic performs much better – here the evaluation functions are clearly misleading.

For Gomoku we note that while Alpha-Beta using the evaluation functions and UCT achieve similar results on the small 8×8 sized board, on the traditional 15×15 board UCT fails completely. Here we see that a likely higher number of

shallow traps together with a large branching factor and the possibility of long playouts results in an immense decrease in performance of UCT. An inverse observation can be made for Clobber: while Alpha-Beta fares reasonably well on a board of size 4×5, the density of shallow traps is likely smaller on the larger board of size 5×6, resulting in an advantage for UCT.

Considering Connect Four, we note that even though the game is closely related to Gomoku, Alpha-Beta fares much worse. As pointed out before, in Connect Four the number of shallow traps is rather small, so that the chances of UCT falling for one are decreased. Concerning Chomp, even though every state is a state at risk, we can ignore traps of depth 0 and 2. Other than these, the trap density is rather small. In the end, this results in bad performance of the Alpha-Beta players compared to the UCT player.

Not all games with few shallow traps are bad for our Alpha-Beta players with the evaluation function: In Chinese Checkers and Nim they are still on-par with UCT. Finally, Sheep and Wolf gives a rather surprising result. The number of shallow traps is not overly high, the branching factor is comparatively small and the length of the game is clearly limited by the size of the board (at worst, all sheep must be moved to the other side). It is quite easy to come up with a strategy where the Min player (the sheep) wins the game. Obviously, our UCT player cannot identify such a strategy while the Alpha-Beta player can, so that the UCT player wins less than half the games when playing as Min, while Alpha-Beta consistently wins.

Related Work on Evaluation Functions for GGP

While most state-of-the art players nowadays make use of UCT, there has been some research in the use of evaluation functions for GGP.

When the current form of GGP was introduced in 2005, the first successful players made use of Alpha-Beta with automatically generated evaluation functions. The basic idea was to identify features of the game at hand (e.g., game boards, cells, movable pieces). By taking order relations into account, it is possible to evaluate distances of pieces to their goal locations (where the order relations describe the connection of the cells of a game board) or the difference in number of pieces of the players (where the order relations describe the increase/decrease of pieces, e.g., when one is captured) (see, e.g., (Kuhlmann, Dresner, and Stone 2006; Clune 2007)).

Another way to evaluate states was used by Fluxplayer (Schiffel and Thielscher 2007): This uses fuzzy logic to evaluate how well the goal conditions are already satisfied. In a setting with a simple conjunction of facts, as we assume in this paper, this pretty much corresponds to a goal-counting heuristic. Additionally they also took identified features and order relations into account to improve this evaluation function. They do so by using different weights, e.g., taking a fact's distance to its goal value into account instead of only a satisfied/unsatisfied status.

A more recent approach (Michulke and Schiffel 2012) considers a so-called fluent graph, which captures some conditions for a fact to become true, but for each action considers only one of the preconditions as necessary for achieving one of its effects. Based on this graph an estimate on the number of moves needed for achieving a fact is calculated, which is again used for weighing the fuzzy logic formulas, similar to the previous approach in Fluxplayer. A similar graph, the so called justification graph, has been used in planning for calculating the efficient LM-Cut heuristic (Helmert and Domshlak 2009), though there the graph is used to calculate disjunctive action landmarks.

While in principle it should be possible to use our proposed heuristics (or other planning based distance estimates) in a similar way, it is not clear how useful this might be. The fuzzy logic based approach of Fluxplayer makes sense when applied in the original GDL setting, which allows for arbitrary Boolean formulas with conjunctions and disjunctions, at least when rolling out axioms. In our setting, however, we allow only conjunctions of variables in the goal descriptions. One way to emulate the Fluxplayer approach in our setting would be to calculate the required distance for each of the goal variables, and then combine those results to calculate an actual value of the evaluation function. If this improves the results is not immediately clear and remains as future work.

Conclusion

In this paper we have proposed new evaluation functions for general two-player zero-sum games inspired by successful heuristics used in automated planning, which are based on ignoring delete lists. By taking the difference in plan lengths for reaching won/lost states and the factor of still relevant actions into account, we ended up with a heuristic with which an Alpha-Beta based player is able to consistently defeat a basic UCT player on games with a large amount of traps. It also copes rather well in some of the games having only few shallow traps, where UCT typically is expected to work well.

In addition to these new evaluation functions we also provided some insight into the presence of traps in a set of GGP benchmarks. The observation here is that basically all those games contain some shallow traps, though for several games the density is rather small, which is a factor explaining the success of UCT players in the GGP setting.

In the future we will adapt further heuristics to two-player games. One approach that comes to mind is the use of abstractions. For some extensive games such an approach has yielded pathological behavior (Waugh et al. 2009), i.e., worse play when refining an abstraction, and it will be interesting to see if such behavior can also occur in our setting.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2–3):235–256.

Björnsson, Y., and Finnsson, H. 2009. Cadiaplayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games* 1(1):4–15.

Bouton, C. L. 1901. Nim, a game with a complete mathematical theory. *Annals of Mathematics* 3(2):35–39.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, 28–35. AAAI Press.

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

Clune, J. 2007. Heuristic evaluation functions for general game playing. In Howe, A., and Holte, R. C., eds., *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI-07)*, 1134–1139. Vancouver, BC, Canada: AAAI Press.

Gelly, S., and Silver, D. 2008. Achieving master level play in 9 x 9 computer go. In Fox, D., and Gomes, C., eds., *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI-08)*, 1537–1540. Chicago, Illinois, USA: AAAI Press.

Genesereth, M. R.; Love, N.; and Pell, B. 2005. General game playing: Overview of the AAAI competition. *AI Magazine* 26(2):62–72.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169. AAAI Press.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, volume 4212 of *Lecture Notes in Computer Science*, 282–293. Springer-Verlag.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Kuhlmann, G.; Dresner, K.; and Stone, P. 2006. Automatic heuristic construction in a complete general game player. In Gil, Y., and Mooney, R. J., eds., *Proceedings of the 21st National Conference of the American Association for Artificial Intelligence (AAAI-06)*, 1457–1462. Boston, Massachusetts, USA: AAAI Press.

Love, N. C.; Hinrichs, T. L.; and Genesereth, M. R. 2008. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford Logic Group.

Méhat, J., and Cazenave, T. 2011. A parallel general game player. *KI* 25(1):43–47.

Michulke, D., and Schiffel, S. 2012. Distance features for general game playing agents. In Filipe, J., and Fred, A. L. N., eds., *Proceedings of the 4th International Conference on Agents and Artificial Intelligence (ICAART'12)*, 127–136. Vilamoura, Algarve, Portugal: SciTePress.

Newell, A., and Simon, H. 1963. GPS, a program that simulates human thought. In Feigenbaum, E., and Feldman, J., eds., *Computers and Thought*. McGraw-Hill. 279–293.

Ramanujan, R.; Sabharwal, A.; and Selman, B. 2010. On adversarial search spaces and sampling-based planning. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 242–245. AAAI Press.

Ramanujan, R.; Sabharwal, A.; and Selman, B. 2011. On the behavior of UCT in synthetic search spaces. In *Proceedings of the ICAPS Workshop on Monte-Carlo Tree Search: Theory and Applications (MCTS'11)*.

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

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A successful general game player. In Howe, A., and Holte, R. C., eds., *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI-07)*, 1191–1196. Vancouver, BC, Canada: AAAI Press.

Waugh, K.; Schnizlein, D.; Bowling, M. H.; and Szafron, D. 2009. Abstraction pathologies in extensive games. In Sierra, C.; Castelfranchi, C.; Decker, K. S.; and Sichman, J. S., eds., *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'09)*, 781–788. Budapest, Hungary: IFAAMAS.

Winands, M. H. M., and Björnsson, Y. 2011. $\alpha\beta$-based play-outs in monte-carlo tree search. In Cho, S.-B.; Lucas, S. M.; and Hingston, P., eds., *Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games (CIG 2011)*, 110–117. Seoul, South Korea: IEEE.