A GGP Feature Learning Algorithm

Mesut Kirci Nathan Sturtevant Jonathan Schaeffer

This paper presents a learning algorithm for two-player, alternating move GGP games. The Game Independent Feature Learning algorithm, GIFL, uses the differences in temporally-related states to learn patterns that are correlated with winning or losing a GGP game. These patterns are then used to inform the search. GIFL is simple, robust and improves the quality of play in the majority of games tested. GIFL has been successfully used in the GGP program Maligne.

1 Introduction

Agents with the ability to demonstrate intelligence over a wide variety of domains remains an elusive goal for the field of artificial intelligence. Even restricting the set of domains to a subset of game playing, arguably an area with "simple" scope, is quite challenging. The state of the art in this area is to develop game-specific solutions that provide insights but do not necessarily generalize to other games. The annual General Game Playing (GGP) competitions have encouraged researchers to explore developing programs that can play a game given only the set of rules [1]. Playing a legal game is easy; having the system play these games at a high level of skill is challenging.

A general game player accepts a game description as input at runtime, analyzes it, and then plays the game without human intervention. GGP programs have to play many classes of games, varying parameters such as the number of players (one or more), move constraints (alternating or simultaneous), and game space (such as board games and card games). Thus, GGP programs cannot use game-specific algorithms or knowledge.

The current state of the art in GGP algorithms, adopted by most of the leading competitors, is to use the UCT search algorithm [5]. The algorithm's appeal is its simplicity of implementation and, more importantly, it achieves good performance with no domain knowledge (other than the game rules). There are two portions to the overall UCT search: a growing tree which is held in memory storing the current value of various moves, and random sampling that proceeds from the leaves of the tree to terminal states in the game. Given enough samples, UCT will provide an educated guess as to the "best" move to play. The advent of UCT represented an improvement in the playing abilities of the top GGP programs (from very weak to weak, as evidenced by play against humans). However, if one wants to achieve high performance, one of the goals of GGP research, some form of application-specific knowledge will have to learned by the game-playing software.

Domain-independent knowledge extraction is a very hard problem. There have been a few attempts to use machine learning in a GGP program but none of them have had significant competitive success. The CLUNE PLAYER and University of Texas programs [6], both frequent GGP competitors, use automatically extracted features to calculate the evaluation function. These are simple features like the number of pieces on the board and the number of legal moves. Sharma et al. use temporal difference learning to build a domain-independent knowledge base that is used to guide the UCT search [8]. This method has not been used by any competitive GGP program.

This paper presents the Game Independent Feature Learning (GIFL) algorithm. It learns features and uses them to guide the search in two-player, alternating-move games. Similar to the well-known history heuristic [7], GIFL uses state differences in 2-ply game trees to identify "good" and "bad" features. The learned features are used to guide the otherwise random move selection in the random sampling portion of the UCT search.

In GGP, the game rules are defined using the Game Description Language (GDL) [2]. A game state is defined as a set of predicates that are true (facts). Predicates present in a state are called state predicates. In the special case of a goal state they are referred to as terminal predicates. GIFL is given a line of play from the starting position to a terminal position, and then does a retroactive analysis. The algorithm identifies states and actions which might be associated with winning or losing, and then attempts to extract general patterns (sets of predicates) that are necessary for an action to be useful. An *offensive feature* is a pattern that is correlated with success and suggests a move that can be used to direct a search towards a positive outcome. A *defensive feature* is a pattern that is correlated with failure avoidance and suggests a move (if legal in the current position) that may prevent a bad outcome. Each feature is assigned a value which measures the distance from the said outcome.

These features could be used by many different algorithms, but we focus on the application to UCT here. During the random sampling portion of the UCT search all available features are evaluated to see if they are applicable in the current state. Immediate wins and losses are taken or avoided. Otherwise, the feature (and associated move) to apply is chosen via a Boltzmann distribution over the value of each valid feature.

This paper reports experimental results for 15 games used in previous GGP competitions. GIFL-enhanced UCT search outperforms standard UCT in nine games, does not effect the results in three games, loses slightly in two games, and loses badly in one game. As well, GIFL was able to improve the performance of the GGP program MALIGNE in five of the games in the 2010 GGP Championship, where MALIGNE took second place. Although the results are strong, one must keep in mind that these are still early days for machine learning in the GGP framework; performance of the programs is still at a relatively weak level of play.

Some of the results in this paper have previously been published [3, 4].

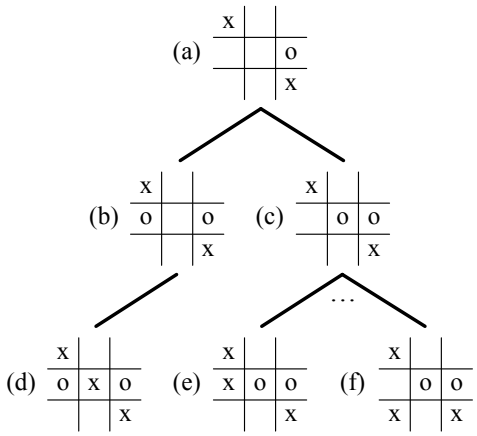

Abbildung 1: 2-ply game tree at the end of the game sequence.

2 Feature Learning

The goal of GIFL is to learn generalized features which can be used to improve the quality of play. Given a move sequence ending in a terminal state, there are two stages to the learning process, which we describe in detail here. In the first stage, a 2-ply game tree is built that leads to a terminal state, and states are identified for learning. In the second stage, general features are extracted from states to be used during game play. We will demonstrate these using examples from the game tic-tac-toe.

2.1 Identifying States for Learning

GIFL identifies states for learning by performing random walks in a game until a terminal state is reached. Then, a 2-ply tree is built around the terminal state to analyze whether learning can occur. We demonstrate this in Abbildung 1. This shows a small portion of a tic-tac-toe tree. The left branch was randomly sampled and ended because the state labelled (d) is a terminal state and a win for the 'x' player. We would like to generalize that in states that are similar to (b), the 'x' player should move in the center to win the game – an offensive feature. Thus, the state (b) is sent to a function which extracts a general feature from the state. We describe a method to do this in Section 2.2.

This tree also shows us, however, that the 'o' player had a better move at state (a). If the 'o' player plays in the center in states like (a), player 'x' will no longer have a winning move at state (c), as evidenced by the successors of (c). We call this a defensive feature. We similarly will then send state (a) to the function which builds a generalized feature from this state.

2.2 Learning Generalized Features

Abbildung 2 illustrates the feature learning process which takes place once interesting states have been identified. The generalization process takes as input a GGP state, an action, and a functional test which must be preserved during the generalization process.

A GIFL feature includes (1) predicates for identifying interesting states, (2) an action to take when the predicates from (1) are found in the current state, and (3) the relative value of the

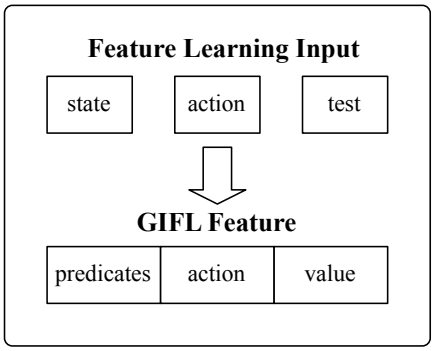

Abbildung 2: General feature-learning process.

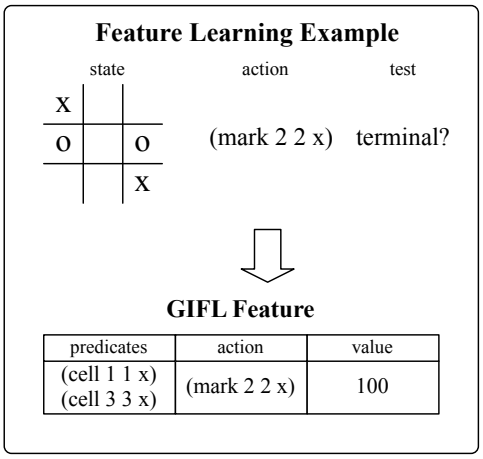

Abbildung 3: Building GIFL features: Only predicates which make the test true when applying the provided action are maintained in the GIFL feature.

feature. The main task in learning GIFL features is to generalize from full states found from the 2-ply trees built in the previous section to a small set of predicates which can possibly match many different states.

The generalization process occurs as follows. Predicates are removed from the input state one at a time. After removing each predicate, the action is applied followed by the test. If removing a predicate makes the action invalid or the test false, then that predicate becomes part of the GIFL feature. If removing the predicate has no effect on the action or test, then it is not required and does not become part of the generalized GIFL feature.

We illustrate two examples of this generalization. The first, in Abbildung 3, is an example of learning an offensive feature. The input state is state (b) from Abbildung 1. The action that was applied at that state to win the game was to mark the middle position in the board with an 'x'. This is an appropriate move as long as applying it will result in the successor state being a terminal or goal state.

To find the generalized predicates, all predicates in the state are removed one at a time, the action is applied, and the test is performed. The predicates in this state are {*(cell 1 1 x) (cell 3 3 x) (cell 1 2 o) (cell 3 2 o)*}. Removing *(cell 1 2 o)* and *(cell 3 2 o)* does not make *(mark 2 2 x)* illegal, and does not

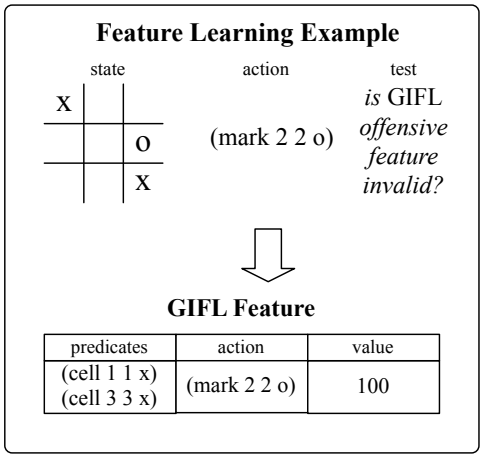

Abbildung 4: Learning a defensive feature higher in the 2-ply tree.

change whether the subsequent state is terminal. Thus, these predicates are not part of the GIFL feature. However, removing either *(cell 1 1 x)* or *(cell 3 3 x)* will cause the test to be false, as the subsequent state will no longer be terminal. Therefore, these are part of the generalized GIFL feature.

Because this action leads directly to the goal, it is given a value of 100. This is an offensive feature, because it directs the 'x' player towards a winning move.

An example of learning a defensive feature is illustrated in Abbildung 4. This state corresponds to the root of the 2-ply tree, Abbildung 1 (a). In this example, it has been discovered that *(mark 2 2 o)* prevents the 'x' player from winning the game, and the state must be generalized. The test here is more complicated. Instead of testing for a win, we must test to see if the previously learned offensive feature becomes invalid.

The predicates *(cell 1 1 x)* and *(cell 3 3 x)* are already known to be required in the GIFL defensive feature, as they have been identified in the offensive feature. Removing the predicate *(cell 3 2 o)* from the state does not prevent the supplied action from successfully blocking the offensive feature, so it is unnecessary. The final defensive feature then states that if the 'x' player has marks in opposite corners, the 'o' player should move in the middle between them. Because this prevents an immediate loss, it also gets a value of 100.

Note that if the 'o' player had an action that erased one of the 'x' players' mark, then this would also be learned as a defensive feature, because the offensive feature would then no longer be applicable in the subsequent state.

2.3 Extending GIFL Features Up The Tree

We have presented here a simple method for learning from a 2-ply tree at the terminal state of a game. Building the tree identifies possible states where offensive or defensive features could be learned, and builds generalized GIFL features from these states. But, ideally, a learner would also learn elsewhere in the tree.

This learning can be performed using the same procedure by looking at moves higher up in the random walk. Instead of looking for a move which leads to a goal, GIFL looks for a move which contributes to the offensive feature. A 2-ply tree is then built around this move, and similar learning takes place. A key difference is that the value of the GIFL features found higher up in the move sequence are discounted the farther away they are from the leaf state.

The value of the move is computed according to the formula $100 \cdot V^{level-1}$ where V is a constant between 0-1 and the *level* is the level of the 2-ply tree where the feature was learned. Trees built from terminal states have level 0. Trees built above the root of a terminal state have a level of 1, and so on. We used $V = 0.9$ in all the results presented here.

An even more detailed presentation with pseude-code for the full feature learning process is available [3, 4].

2.4 Limitations and Further Generalization

What has been described so far is a simple approach to generalizing and building GIFL features. It is possible that some necessary predicates are not properly generalized using this method; there are likely a wide variety of methods for generalizing offensive and defensive features which have yet to be explored.

For instance, suppose that there are three stones placed vertically in the game of Connect4. If the stone in the middle is removed and then a stone is placed on top of that column, the game description dictates that a stone is placed on top of every stone that has an empty space on top. Therefore, the empty place in the middle is replaced even though it is not supposed to be. This will result in the stone in the middle not being a part of the offensive-feature predicates even though it should be.

To solve this problem, GIFL uses another method to find the offensive-feature predicates. If a predicate which was removed during the generalization process is found back in the state after applying the sample action, then this predicate is added to the GIFL feature.

Another example which our implementation of GIFL does not handle is a goal which requires any two of three predicates to be satisfied. If all three predicates are in the state identified for learning, then none of them will be include as predicates in the GIFL feature, as it is only the combination of features that is necessary. While removing pairs of states would reveal such dependencies, higher level logical analysis of a game could lead to stronger generalization rules to handle cases such as these.

3 Using Features

GIFL features are used to guide the random simulation in a UCT search. The program checks each state during a simulation to see whether or not a feature can be applied. A feature can be applied if the predicates of a feature are matched, and if the move associated with the feature is legal. If there is a feature match, then the move associated with that feature is marked with the value from the GIFL feature. Moves with a value of 100 lead to immediate wins or losses and are taken immediately, with preference given to offensive features.

After all of the applicable features are found, the program selects a move according to probabilities calculated with a Boltzmann distribution:

$$p(a) = \frac{e^{V(a)/\tau}}{\sum_{b=1}^{n} e^{V(b)/\tau}}$$

where there are n actions and $V(a)$ is the value of an action. This will bias the random simulation towards known offensive and defensive moves, improving the quality of the simulation. This provides a good exploration-exploitation balance to the move selection. Even though a higher valued feature leads to a win in fewer moves, the outcome depends on the opponent's response. Therefore, other possible moves are explored. We used $\tau = 0.5$ in all the results presented here.

There is one final step used in the application of GIFL features. If both players use GIFL features equally during the random UCT simulations, they will improve simulation for each player equally well, and the performance may not improve. Instead, each player has an independent probability of using the GIFL features at each step, and the probability for the opponent is lower than for the learning player. This incorporates some form of opponent modeling, causing the learning player to attempt to exploit the non-learning player. In all results reported here the opponents were modeled as only being able to use the learned features 50% of the time.

Pseude-code for how to use the features in UCT search can be found in [3].

4 Experiments

The experiments were prepared using the game definitions in the Stanford GGP repository [2] and those used in the 2008 and 2009 GGP competitions. Some game from the 2008 competition are named arbitrarily like game1, game2, etc. All games are 2-player, alternating move and prefect information. Also, in some games being first player or second player may be advantageous. Therefore, experiments are conducted so that this does not effect the results.

The player that uses the features to guide the random simulation is called the learning player, and is compared against a UCT player with purely random simulation. Therefore, the only difference between the learning player and the non-learning player is that learning player uses the learned features to guide the random simulation phase during the UCT search.

We present four sets of results here. First, we look at the number of learned features and the performance of the learning player when playing against a non-learning player with the same number of UCT simulations. We then analyze the speedup and slowdowns that GIFL introduces before comparing performance with a fixed amount of time per play.

The number of training runs is limited to 500 unless specified otherwise. The learning time may vary between 100 training runs per minute in breakthrough and 20 training runs per minute in checkers. The level of 2-ply tree in which the learning is occurring is limited to 3. This reduces the number of the features and the time spent in random simulations as too many features increases the cost of feature mapping.

4.1 Number of Learned Features

We begin by looking at the number of learned features for a variety of games. As two minutes is a common start clock, we show what can be learned in this time frame in Tabelle 1.

For most games a significant number of features can be learned during the (short) time limit. This is important, but perhaps less important than it seems, as learning can take place while UCT is already beginning to explore the game tree. The dual use of simulations reduces the effective overhead of GIFL learning.

The total number of offensive features learned is always greater than the total number of defensive features, as a defensive feature can only be learned in response to an offensive feature.

4.2 Fixed UCT Simulations

Given that a significant number of features can be learned, we then measure the effectiveness of these features on play. Results are in Tabelle 2. The scores are the average score when playing two games, one as the first player and one as the second player. Thus, a score of 193-7 in game2 results from always winning and getting 100 points as player one, and averaging 93 points as player two. Games that are constant-sum have scores that add up to 200. Chess and game5 are not constant-sum.

Of the 15 games that were used, the learning player defeats the non-learning player in nine of the games. The knowledge does not significantly affect the results in three games. In two games, the learning player loses by a small margin. Using learned knowledge decreases the quality of play significantly in only one game, checkersbarrelnokings.

In seven of the nine games for which the learning player has the advantage over the non-learning player, the results are statistically significant. Thus, in the GGP competition setting, where games start from the same initial state, the learning player is expected to beat a UCT player in these games with 95% confidence. This shows that GIFL features improve the performance of UCT search.

Starting from the same initial state could result in test games that were played identically. However, because the UCT search does random simulations the test games were not repetitions of the same game. In 12 of the 15 test domains all the games differentiated in less than five moves, and in the other three domains up to 10 moves were needed.

It should be noted that the games in which the learning does not affect the results are not very interesting: the first player always wins in pentago, all games are tied in game4, and all games end in less then 10 moves in quarto.

Usage of GIFL seems to degrade performance in checkersbarrelnokings. Although this game is similar to the original checkers, at which the learning player has a clear advantage, the learning player loses badly in checkersbarrelnokings. There are a number of reasons for this.

In checkersbarrelnokings, due to lack of kings and forced capture moves, the number of legal moves per step is low. Therefore, the non-learning player does less unnecessary exploration. For example, there are 20 legal moves in a state in breakthrough. When the learning player uses a GIFL feature, this gives an advantage over the non-learning player because the non-learning player explores all of the 20 moves. However, the average number of legal moves for checkersbarrelnokings is low and the advantage gained from using GIFL features is also lower than gained from the breakthrough.

In addition, learning capture moves, which are the most important moves to win the game, is not useful because they are forced moves. Therefore, GIFL can only make a difference by learning defensive feature moves. Learning defensive features is

name	source	n. of features	n. of offensive features	n. of defensive features
game2	2008 competition	135	75	60
pawn whopping	2009 competition	328	185	143
knightthrough	2008 competition	134	79	55
game1	2008 competition	1869	1014	855
breakthrough	2007 competition	120	63	57
checkers	[2]	895	458	437
connect4	[2]	2187	1108	1079
chess	[2]	25	13	12
game5	2008 competition	1357	686	671
pentago	[2]	679	405	274
game4	2008 competition	1101	558	543
quarto	[2]	8527	5254	3273
game6	2008 competition	1456	813	643
game3	2008 competition	1555	812	743
checkersbarrelnokings	[2]	5262	2830	2432

Tabelle 1: Number of features that learned by GIFL in two minutes.

name	n. of simulations	n. of games	learning-uct	point percentage	95% confidence
game2	1000	20	193-7	97.5 %	√
pawn whopping	1000	20	190-10	95.0 %	√
knightthrough	1000	20	184-16	92.0 %	√
game1	1000	20	170-30	85.0 %	√
breakthrough	1000	20	165-35	82.5 %	√
checkers	150	20	156-44	78.0 %	√
connect4	1000	100	115-85	57.5 %	√
chess	25	40	102-84	54.8 %	
game5	1000	40	111-94	54.1 %	
pentago	1000	100	100-100	50.0 %	
game4	1000	40	100-100	50.0 %	
quarto	1000	100	98-102	49.0 %	
game6	1000	100	96-104	48.0 %	
game3	1000	100	91-109	45.5 %	
checkersbarrelnokings	1000	100	61-139	30.5 %	√

Tabelle 2: Effectiveness of using GIFL with a fixed number of simulations for each player.

also harder in checkersbarrelnokings because escaping an imminent capture is often done by capturing the opponent piece (forced move). GIFL learns features that help the player to avoid getting captured in the next turn. However, GIFL features do not contain information about other pieces of the opponent. Thus, when a GIFL feature suggests how to avoid capture from one piece, it may inadvertently put itself in position to be captured by an alternate piece.

4.3 Cost of GIFL

Running GIFL incurs cost overheads which we measure in this section. The biggest overhead is that of matching GIFL features with each state to see if they are applicable. However, there is an alternate benefit from using these features, as they decrease the length of the random UCT simulations. We measure both effects in Tabelle 3, predicting the expected speedup or slowdown.

The second column reports the ratio of the learner's random walk length to the length of a regular UCT random walk. In most games using GIFL decreases the length of the random walk, sometimes significantly. However, in checkersbarrelnokings the length of random walks is increased significantly, a factor in the poor performance in that game.

The third column reports the ratio of simulations performed by the learner to the regular UCT player. This measure already takes into account the shorter simulation length. Despite the shorter lengths, the learner is performing fewer simulations in all but two games.

game	simulation length (learner/uct)	n. of simulations (learner/uct)	GIFL Overhead (uct/learner)	Effective Overhead
game2	51 %	46 %	4.3	2.2
pawn whopping	82 %	65 %	1.9	1.5
knightthrough	45 %	93 %	2.4	1.1
game1	53 %	104 %	1.8	1.0
breakthrough	61 %	79 %	2.1	1.3
checkers	73 %	36 %	3.8	2.8
connect4	95 %	20 %	5.3	5.0
chess	99 %	74 %	1.4	1.4
game5	101 %	32 %	3.1	3.1
pentago	68 %	156 %	0.9	0.6
quarto	100 %	34 %	2.9	2.9
game6	97 %	58 %	1.8	1.7
game3	52 %	99 %	1.9	1.0
checkersbarrelnokings	178 %	38 %	1.5	2.6
game4	116 %	47 %	1.8	2.1

Tabelle 3: Average length of simulations.

Column four is the inverse of the product of columns two and three. This gives the factor by which GIFL slows down the GGP player. However, because the simulations are of shorter length, the effective cost is less, as shown in the last column (the inverse of column three).

For example, for game2 the simulations are, on average, half the length with GIFL. Because the cost of the GIFL analysis slows down the program by a factor of 4.3, the program runs 2.2 times slower than with regular UCT.

name	learning-uct	win percentage
game2	140-30	70.0 %
pawn whopping	190-10	95.0 %
knightthrough	150-50	75.0 %
game1	160-40	80.0 %
breakthrough	170-30	85.0 %
checkers	110-90	55.0 %
connect4	90-110	45.0 %
chess	75-125	35.0 %
game5	40-160	15.0 %
pentago	100-100	50.0 %
quarto	80-120	50.0 %
game6	70-130	35.0 %
game3	100-100	50.0 %
checkersbarrelnokings	30-170	15.0 %
game4	100-100	50.0 %

Tabelle 4: Effectiveness of using GIFL with 30 seconds per move.

4.4 Fixed Time

We complete our experiments in games with a fixed time limit, shown in Tabelle 4. The games are still ordered from best to worst performance given a fixed number of UCT simulations. Although the numbers are not as favorable as before, there are still significant gains in many different games.

These are not the best possible results using GIFL, as it has not been tuned for maximal performance. Most of the work of GIFL could be integrated into the inference engine, thereby reducing the overhead.

5 Conclusion

The learning algorithm learns GIFL features and uses them to guide random UCT simulation. The concepts are simple and domain independent which is essential for GGP algorithms. Up until the 2008 GGP competition, learning algorithms have not been an essential part of a successful GGP program because domain-independent learning is a very hard problem. However, this paper presents a simple but effective method that shows very promising results in some of the games that are frequently used in GGP competitions.

The algorithm shows promising results in GGP, but the learning concepts are heavily depended on the terminal conditions. If the goal conditions of a game is too specific, the features may not be encountered frequently. Thus, GIFL may not be effective. For instance, the terminal conditions of chess has many variations depending on the position, number and type of pieces. GIFL learns one of these variations at each step of the algorithm. The occurrence of that specific terminal position during a simulation is necessary for the learned feature to be effective. However, most of the GGP games in which GIFL is successful, have less number of different possible terminal conditions. In conclusion, the effectiveness of GIFL depends on how many variations terminal conditions of a game can have.

In addition, the computation overhead of using features is an important area for the future work. The primary focus of GIFL is the effectiveness of the features, therefore time has not been spent to develop more efficient ways of feature matching and feature pruning. Some of the learned features may not be effective and can be removed. We believe that significant performance gains are possible.

The algorithm has room for improvements. First, the features can be used as a part of an evaluation function. A minimax approach can be tried with this evaluation function instead of the UCT search.

Second, the algorithm can only learn features from a game sequence if the player that wins the game makes the last move. The learning algorithm cannot be applied to games when the losing side makes the last moves. Lose Checkers is an example of these types of games. The players aim to lose all the pieces instead of trying to capture them. This problem may be solved by changing the leaf of the 2-ply tree where the learning occurs.

In addition, the frequency of features seen in the learning process can be included when the values for the feature moves are calculated. Right now, all of the features have the same importance.

The learning algorithm presented in this paper is relatively simple, yet we have shown it to be quite successful. There are certainly more complex approaches which could be even more successful. We look forward to future competitions encouraging even more learning with longer start clocks which would allow more learning to take place before a game begins.

Literatur

[1] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[2] Stanford Logic Group. `http://logic.stanford.edu`.

[3] Mesut Kirci. Feature learning using state differences. Master's thesis, Computing Science, University of Alberta, 2009.

[4] Mesut Kirci, Nathan Sturtevant, and Jonathan Schaeffer. Feature learning using state differences. In *IJCAI Workshop on General Game Playing*, 2009.

[5] Levante Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

[6] Gregory Kuhlmann and Peter Stone. Graph-based domain mapping for transfer learning in general games. In *Proceedings of the 18th European Conference on Machine Learning*, September 2007.

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

[8] Shiven Sharma, Ziad Kobti, and Scott Goodwin. Knowledge generation for improving simulations in UCT for general game playing. In *AI 2008: Advances in Artificial Intelligence*, pages 49–55. Springer-Verlag, 2008.

Kontakt

Mesut Kirci
Email: kirci@ualberta.ca

Nathan Sturtevant
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6M 2K9 Email: nathanst@cs.ualberta.ca

Jonathan Schaeffer
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6M 2K9 Email: jonathan@ualberta.ca

Bild

Mesut Kirci has a M.Sc. degree from the Department of Computing Science at the University of Alberta. He is currently working for Taleworlds, a game-development company in Turkey.

Bild

Nathan Sturtevant is an Assistant Professor in the Department of Computer Science at the University of Denver, however the work in this paper was completed while he was a assistant adjunct professor at the University of Alberta. Nathan's research focuses on heuristic search, with contributions in single-player and multi-player games. His work in single-player search was incorporated in the game Dragon Age, which has sold over one million copies.

Bild

Jonathan Schaeffer is a Professor of Computing Science at the University of Alberta. He is the iCORE Chair for High-Performance AI Systems. For over 30 years he has been using games and puzzles as experimental testbeds for his AI research. He is best known for developing CHINOOK, the first program to win a human world championship in any game.