Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Generalized Monte-Carlo Tree Search Extensions for General Game Playing

Hilmar Finnsson
School of Computer Science
Reykjavík University, Iceland
hif@ru.is

Abstract

General Game Playing (GGP) agents must be capable of playing a wide variety of games skillfully. Monte-Carlo Tree Search (MCTS) has proven an effective reasoning mechanism for this challenge, as is reflected by its popularity among designers of GGP agents. Providing GGP agents with the knowledge relevant to the game at hand in real time is, however, a challenging task. In this paper we propose two enhancements for MCTS in the context of GGP, aimed at improving the effectiveness of the simulations in real time based on in-game statistical feedback. The first extension allows early termination of lengthy and uninformative simulations while the second improves the action-selection strategy when both explored and unexplored actions are available. The methods are empirically evaluated in a state-of-the-art GGP agent and shown to yield an overall significant improvement in playing strength.

Introduction

General Game Playing(GGP) (Genesereth, Love, and Pell 2005) has become an established field in the AI community with annual competitions since 2005. The focus of GGP is to research and produce agents capable of playing skillfully any game presented to them without human intervention.

Every year since the 2007 GGP competition the first-place agent has been based on the Monte-Carlo Tree Search (MCTS) algorithm. This algorithm has effectively applied Monte-Carlo simulations to tree-search problems and over the past years become a fully developed search method with a clearly defined structure. The origin of MCTS is the Upper Confidence Bounds applied to Trees (UCT) algorithm (Kocsis and Szepesvári 2006).

It was success in the game of *Go* (Gelly et al. 2006; Enzenberger and Müller 2009) that brought MCTS its deserved attention. But since then it had proven itself again and again in such games as *Amazons* (Lorentz 2008), *Lines-of-Action* (Winands, Björnsson, and Saito 2010), *Chinese Checkers* (Sturtevant 2008), *Spades and Hearts* (Sturtevant 2008) and *Settlers of Catan* (Szita, Chaslot, and Spronck 2009).

One of the most alluring traits of Monte-Carlo simulations is its ability to function without heuristic knowledge. This makes it ideal for GGP as it gives the agent a chance to cope with any surprise thrown its way. In practice though MCTS must be enhanced with knowledge to be truly effective. When applied to specific games like *Go* this knowledge can be acquired directly from human experience or learned beforehand from data gathered from games played by expert *Go* human players. In GGP this is not possible as the agents must learn everything in real time. The contributions of this paper are two extensions that generalize their knowledge acquisition such that no domain specific knowledge is required a priori. Real time information is gathered from the MCTS simulations rather than making assumptions from the game rules. These extensions are *Early Cutoffs* and *Unexplored Action Urgency*. They deal with terminating simulations early if they are good enough or useless and keeping the focus on good actions in the presence of completely unexplored actions respectively.

The paper is structured as follows. We begin by giving a summary of GGP and MCTS before describing the proposed extensions. This is followed by results from various experiments and our analysis of what they represent. We then end by discussing related work and concluding.

Background

General Game Playing

The *General Game Playing(GGP)* challenge (Genesereth, Love, and Pell 2005) centers around the creation of intelligent game-playing agents that can accept and skillfully play any game that comes their way. The language used to describe games in GGP is called the *Game Description Language* (GDL) (Love et al. 2008). The games this paper focuses on are written in GDL version 1.0 which requires the games it describes to be perfect-information and deterministic. GDL-II (Thielscher 2010; Schiffel and Thielscher 2011) is a recent addition to GGP that allows imperfect information and randomness. The new version of GDL will not absorb the older one and each will have their separate track in competitions.

When a GGP agent starts to play a game it has no knowledge of what tasks it is about to undertake and from this moment on there can be no human intervention. A program dubbed the *Game Master* (GM) sends out the GDL game description to all the participating agents plus information

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

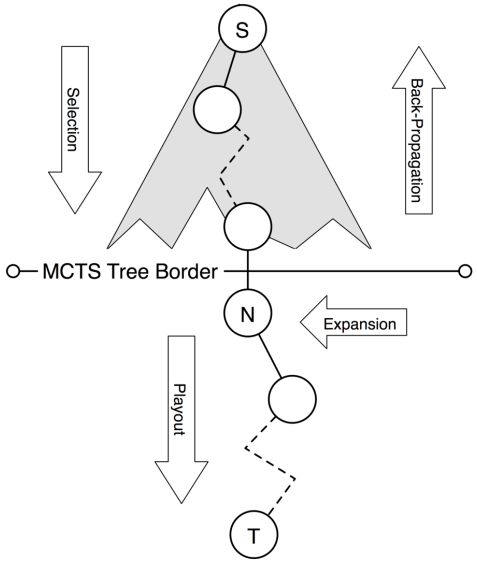

Figure 1: An overview of a single simulation

about what role they play and the time controls. The time controls are known as *startclock* and *playclock*. The former gives the time in seconds until the actual gameplay will begin relative to when the GDL was received and the latter is the think time between moves. All GDL games are at their core a simultaneous move game as agents are required to send the GM a move on every turn, but turn-taking is easily emulated with alternating ineffectual dummy moves. The GM is responsible for informing all agents about all moves made within the game until it terminates.

Monte-Carlo Tree Search

MCTS uses its deliberation time to generate information from simulation rewards. While running these simulations an in-memory tree is built to store the generated information, henceforth called the MCTS tree. This tree reflects the top part of the actual tree described by the game it is playing. Just like the simple Monte-Carlo method, the algorithm uses the data gathered at the root to decide on the next move to make.

This process has been formally separated into four consecutive tasks or phases. These are *selection, playout, expansion,* and *back-propagation*. Each of these phases can be implemented a number of ways independently but in this description we will keep it simple by focusing on the most basic version which is usually a good starting point for any MCTS application. As a visual reference Figure 1 will be used. It depicts a single simulation beginning at state S running through the MCTS tree (the shaded area) and some unknown part of the game tree before terminating at state T.

Selection

With every simulation run the MCTS tree expands and while a simulation is within this tree it has access to information on how earlier traversals of the collected states have fared. The UCT (Kocsis and Szepesvári 2006) formula does a good job balancing exploration and exploitation with regards to the information in the MCTS tree and converges to the best action given enough samples. UCT selects the most informative action which is considered to be a^* when selected from the set of available actions $A(s)$ in state s using the following:

$$a^* = argmax_{a \in A(s)} \left\{ Avg(s,a) + 2 * C_p \sqrt{\frac{\ln Cnt(s)}{Cnt(s,a)}} \right\}$$

The function Avg returns the average reward observed when taking action a in state s and Cnt returns the number of times state s has been sampled or how often action a has been selected in state s, depending on the parameters. The C_p parameter is a tunable variable which can be used to influence the exploration/exploitation balance with higher values giving more exploration. The balancing part of the formula is known as the *UCT Bonus*. In Figure 1 the selection phase lasts from state S up to but not including the state N.

Playout

At some point during the simulation it will fall out of the in-memory MCTS tree. This marks the start of the Playout phase. The basic strategy in this phase is just to play uniformly at random. This phase includes everything from state N to and including state T in Figure 1.

Expansion

The name of this phase refers to expanding the in-memory MCTS tree. The most common strategy here is expanding the tree by one state each simulation, the added state being the first state of the playout phase (Coulom 2006). This way the MCTS tree grows most where the selection strategy believes it will encounter its best line of play. For the simulation in Figure 1 this means that state N is incorporated into the MCTS tree.

Back-Propagation

The back-propagation phase controls how the end reward of the simulation is used to affect the traversed states of the current MCTS simulation path. In the basic implementation, just as with MC simulations, this keeps track of the average rewards. To distinguish state rewards at different distances from terminal states, a discounting factor can be applied to the reward as it is backed up from the terminal state to the root.

Generalizing MCTS Extensions

In GGP it is a difficult task to generate reliable knowledge and can even backfire, causing undesirable behavior in the agent. The difficulty of knowledge generation has in part contributed to the popularity of the MCTS agent in the field as in its purest form it needs no prior knowledge to play any game. Extending MCTS with knowledge can produce great results but the improvement gain relies heavily on the quality of the knowledge used. Any attempt of injecting such

extensions into an MCTS GGP agent must take into consideration at least two possible scenarios. The extension may be harmful in some games if the knowledge it utilizes is not general enough and in some cases it may not be applicable at all and should be ignored.

Following are generalizations of two extensions for MCTS. They make use of statistical information gathered in real time from the simulations rather than making static assumptions from the game rules.

Early Cutoffs

Without knowledge to guide the MCTS simulations huge amounts of computation time may be wasted on irrelevant continuations in the playout phase. There is a danger of simulations running for a long time, missing a lot of obvious good moves or even just going around in circles in the state space. When they finally end up in a terminal state the information backed up in the tree may be just noise. Also, to ensure that games are deterministic, GGP games often encode a counter into their rules to terminate game playouts that have gone beyond reasonable length and scores them as a draw without any regards to who is closer winning. If it would be possible to predict that a simulation is headed nowhere it would be better to terminate it immediately and use the time saved to run another simulation that has a better chance of generating more relevant information.

In other cases simulations may have entered a part of the state space when one player is in such a good position that it is not necessary to continue and better to score the game immediately so another simulation can run. It may even be enough to know when in the playout phase the simulation discovers a better or worse position for the agent and returning back this information.

This extension covers two cases when simulations can be terminated early. On one hand is the case when a simulation can be evaluated to an informative score before reaching a terminal state and on the other hand is the case when the simulation has gone on longer than any rational play of the game would without reaching a terminal state.

Before cutoffs can be made we first gather data on if the extension is applicable to the game being played. This is done by observing the "normal" simulations of the agent to get information on if the extension should be activated and if so how should its setting be. This data includes the depth of terminal states and therefore needs naturally terminated simulations.

To see if it is possible to evaluate the states we use the scoring mechanism of GDL, the *Goal* relation. The trouble here is that it makes no promises of returning useful information, if any at all, in non-terminal states. Still in some games it can give a good estimation on the state as a result of how the game rules are written. To make sure that the *Goal* relation can be used we check it for a property called *Stability*. The notion of stability in GGP was invented for the agent ClunePlayer (Clune 2008) and was used to determine if a feature extracted from the game description would be useful in evaluating game positions. Stability is based on the assumption that if a value calculated from a state changes gradually throughout the course of the game it is in some way correlated with what is actually happening in the game as it plays out. If this value jumps back and forth continuously or never changes it is deemed unrelated to the game progression. To calculate the stability value the variance of the changes in the score obtained from the *Goal* relation is observed throughout the simulations and then averaged over them. If the resulting stability value is lower than a predetermined threshold but not zero this score is believed to be stable and the extension is activated.

The pseudo-code in Algorithm 1 outlines the decision-making process if an early cutoff should be made and we use it as a reference throughout this section. It is called right after checking if the current state is terminal and before any actions retrieval or selection is made to transition to the next state. The variables *useEarlyCutoff* is true if the extension is active and *playoutSteps* always contains the current number of states traversed since the current simulation left the MCTS tree.

The *Goal* stability part of the extension has precedence and once it has been activated it terminates simulations and scores them with the *Goal* relation as shown with the conditional function *IsGoalStable* in Algorithm 1. The cutoff length is set as the distance from the initial state of the game to the first state where the goal changes with the number of players participating added. It is then checked relative to how many playout steps have been taken. This way the simulations should immediately start to draw attention towards the parts of the state space where something is to be gained. The player count addition is made to allow at least one round to be played beyond the goal point if there is an immediate neutralizing response, not to mention if it is a forced one. These variables are only set in the beginning at the same time the stability is determined and therefore keep the same distance from the MCTS tree fringe throughout the game.

In order to predict if a simulation is a waste of time due to its length the agent collects data on at what depth terminal states are encountered and uses it to estimate the depth interval in which they occur. If the interval is very narrow or always at the same depth, there is no use making early cuts as little or no time can be saved by doing so. The minimum interval size is set equal to the variable *minimumSteps* seen in Algorithm 1 where it is also used as a failsafe for a minimum number of playout steps that must be made for a cut to be considered. If it is found that the interval is sufficiently big the extension becomes active through the conditional function *hasTerminalInterval()* in Algorithm 1.

As before a cutoff depth is set and then checked relative to the MCTS tree fringe by keeping track of the number of steps taken in the playout phase. The cutoff depth is set relative to the initial state and matches traversing the third of the interval where terminal states have been observed. As an example lets say we have a game where terminal states have been reached from depths 20 to 50 relative to the initial state. The cutoff depth is then:

$$20 + \frac{1}{3} * (50 - 20) = 20 + 10 = 30$$

If we then exit the MCTS tree at depth 8 from the initial state the simulation will be terminated at depth 38. A terminal interval cutoff defaults to a draw (GGP score of 50).

Algorithm 1 Pseudo-code for deciding cuts for the Early Cutoff extension

```
if not useEarlyCutoff then
    return false
end if
if playoutSteps < minimumSteps then
    return false
end if
if IsGoalStable() then
    // Cutoff point has been calculated as:
    // cut ← firstGoalChange + numPlayers
    return playoutSteps ≥ cut
end if
if hasTerminalInterval() then
    // Cutoff point has been calculated as:
    // cut ← firstTerminal + 0.33 ∗ terminalInterval
    return playoutSteps ≥ cut
end if
```

The reason for one third being the fraction of the terminal state interval traversed as a minimum for a cut to be made was based on results from Finnsson and Björnsson (Finnsson and Björnsson 2011). They show that pushing back an artificial termination depth to give a UCT agent more chance to find a real terminal state does not equal better play through better information. In the example game there the agent reached its peek performance as early as only finding a real terminal state 30% of the time. From those results, traversing the top one third of the terminal state interval as an absolute minimum is a reasonable setting for this parameter.

Unexplored Action Urgency

In GGP it is commonplace to use UCT in the selection phase to control the exploration/exploitation balance in MCTS. By default UCT never exploits on the fringe of the MCTS tree, always selecting amongst the unexplored actions available. It is not until all unexplored actions have been exhausted that attention is given to exploitation in the state. When domain-specific knowledge is available unexplored actions can be evaluated and bad ones ignored. This may sound excessive but if the knowledge is accurate, the agent would never have selected the bad action anyway.

The main idea with the extension is to make it possible for the agent to exploit actions on the MCTS tree fringe if there is reason to do so. Even though no domain specific knowledge is available we can observe how the rewards of an action accumulates. Basically if the reward is consistently very good the action estimated average will stay high no matter how often we select. So if an action fits this description we are now going to prefer it over any unexplored action found in the state while the number of simulations traversing this state is low. Instead of the default rule with unexplored actions taking precedence a quick decision point is added when the agent is on the MCTS fringe. The decision of selecting an unexplored action is given a value that can be compared with the highest UCT value of the already explored actions and the higher value wins. Basically the agent is selecting between using the selection strategy on the subset of explored actions in the state or the playout strategy on the subset of the unexplored actions. We call the value assigned to the unexplored actions their *urgency*, which is a phrase used when knowledge is used to assign values to actions in (Bouzy 2005).

Algorithm 2 Pseudo-code for action selection when using the Unexplored Action Urgency extension

```
if state.explored = ∅ then
    // We are in the Playout phase
    return playoutStrategy(state.unexplored)
end if
action ← selectionStrategy(state.explored)
if state.unexplored = ∅ then
    // Fringe not reached in the Selection phase
    return action
end if
// Fringe reached in the Selection phase
exploit ← action.uctValue
discount ← state.unexplored.size() / state.actions.size()
urgency ← 50 + Cp∗ √(ln state.visits() * discount)
if exploit ≥ urgency then
    return action
else
    return playoutStrategy(state.unexplored)
end if
```

Pseudo-code for the urgency calculations is shown in Algorithm 2. The variable $state$ accesses the current simulation state and from it we get three action sets, $unexplored$ actions, $explored$ actions, and all $actions$. The $exploit$ variable stores the highest UCT value of the already explored actions in the state and the $urgency$ variable holds the urgency of the unexplored actions. The formula for the urgency is actually the UCT formula as it would look for an action that has been tried once before in this state and resulted in an immediate draw (GGP scores range from 0 to 100). The $discount$ variable has the purpose of lowering the urgency as more and more actions get explored. We have the discount because with fewer and fewer actions left unexplored, the more likely we are to have already come across the best available action. By incorporating the UCT formula into this decision we keep the guarantee of all actions getting explored in the limit.

This extension relies on the agent actually selecting a good action early on when exhausting the unexplored actions in a state on the MCTS fringe. It is likely that given a uniform random playout strategy this extension will simply get balanced out and no improvements are made. This extension has to be paired with a playout strategy that shifts the action selection in favor of selecting good actions early. Given such a playout strategy this extension should boost its effect while not causing harm if it turns out only as good as the uniform random strategy in some games. An example of such an extension is the *Move-Average Sampling Technique* (MAST) (Finnsson and Björnsson 2008) which we will use in order to evaluate this extension in the experiments section of this paper.

Empirical Evaluation

Both extensions were implemented into a GGP agent, just as they are intended to be used, and therefore all experiments were run as GGP games.

Games Used

The extensions were tried on the ten different games listed in Tables 1 and 2, many of them well-known although some need some explanations. *Amazons small* is *Amazons* on a 6×6 board. *Battle* is a simultaneous move game played on an 8×8 board where each player has 20 disks along the edges of two adjacent sides that can move one square or capture an opponent disks next to them. The goal is to be the first to capture 10 opponent disks. *Breakthrough* is a turn-taking race to the back ranks of the opponent using chess-like pawns. The initial setup is the same as in chess with all pieces replaced by these pawns in the corresponding color. The pawns can move forward or diagonally one square at a time and when moved diagonally they may capture an opponents piece. *Chinook* is a games presented at the 2010 GGP competition and is a breakthrough type game using pieces that move like *Checkers* pawns. *Chinook* also has simultaneous moves such that White alternates moving pawns on white and black squares each ply starting with a pawn on a white square while Black moves simultaneously one of its pawns that reside on the opposite colored squares. *Knight-through* is also a breakthrough type game played with chess knights that may only play knight-type moves that advance them on the board. *Runners* is a simple simultaneous game where two runners can at every ply select from standing still or taking one, two or three steps forwards or backwards. The runners then race to a goal 50 steps away and have 50 actions to reach it. This game's GDL encoding is deceptively tricky in its simplicity for the kind of statistical data simulations rely upon. *Skirmish* is a game with chess pieces where the object is simply to capture all the pieces of the opponent. In this version each player has four pawns which can only move by capturing, two knights, two bishops and two rooks. In addition the squares at *c3*, *c6*, *f3* and *f6* are off limits.

Setup

The agents ran on a single Intel Xeon 2.66GHz, 3.00GHz or 3.20GHZ CPU (competing agents always on matching CPUs) and every game type was run as a tournament between two agents with 300 game instances played, having the agents switch roles half way through. In every game the agents were given 10 seconds both for the startclock and the playclock. The number of sample simulations for the Early Cutoff extensions was set as 100 and its stability threshold as 1000. The results are displayed in Tables 1 and 2. The cells marked as -*N/A*- under the Early Cutoff extension are games where the extension decided that it should not activate. In every tournament an extended agent was pitched against the base agent it was derived from. All cell data is the winning percentage of the extended agent along with a 95% confidence interval.

The base agent used was a basic UCT/MCTS agent as described in this paper and enhanced with a transposition table. Also a UCT agent extended with MAST was used. MAST has the ability to bias the playout action selection in many games towards good actions, allowing the hypothesized consequences of Unexplored Action Urgency extension to be checked. In Table 2 the base agents are extended with both extensions to reveal the overall gain and test for any improvement overlap. The third base player in that table, the RAVE/MAST/FAST agent is the extension combination used by the GGP agent CADIAPLAYER (Finnsson and Björnsson 2008) in the GGP competition in 2011. CADIAPLAYER is a two-time winner of the competition and came in second in the 2011 competition and can therefore be considered as one of the top GGP agents. We matched this player against an extended version of itself to answer if the proposed extensions can help an agent of such caliber. Information on *Rapid Action Value Estimation* (RAVE) can be found in (Gelly and Silver 2007) and please refer to (Finnsson and Björnsson 2010) for information on *Features-to-Action Sampling Technique* (FAST). The failsafe variable for minimum steps in the Early Cutoff extension was set to 10.

Results

The results for the Early Cutoff extension show very significant improvement in the games *Battle*, *Checkers* and *Skirmish* both with and without the MAST extension. Also it is important to point out that in other games where this extension is active it shows little or no sign of having adverse effects on the agent. Regarding the games that did not activate the extension, lets take a closer look at *Othello* as an example. It has only extremely rare instances of the game ending before the board has been filled making the terminal interval just a single value when sampled. The player who has more disks wins and this is the information the *Goal* relation gives at every state of the game. This information goes back and forth in declaring which player is winning and therefore gets evaluated as unstable by the extension. This matches the generally known fact that the difference in number of disks is a bad heuristic for Othello.

The Unexplored Action Urgency extension yields significant improvements on many of the games and as expected it needs an informed playout strategy to work. This is confirmed by the fact that almost no improvement is gained when joined with the random playouts of UCT while the MAST joining shows significant improvements in six out of the 10 games. The results for *Runners* are very curious as it improves the agent in this game single handedly without the aid of MAST. It appears that for some games keeping focus on actions that seem good by relatively few random samples can help. Also it is a bit disappointing to see that *Skirmish* seems to suffer when using this extension. Because the development of MAST was inspired by problems UCT encountered when playing *Breakthrough*, it is not surprising to see the biggest improvements there.

When the extensions have been joined together, the improvements they provide do not seem to overlap except in the case of the quirky *Runners* game and *Battle*, even though still at approximately 75%, does worse than without the Unexplored Action Urgency in the UCT case. With MAST we

Table 1: Isolated Extension Tournaments

Game	Early Cutoff vs.UCT win %	Early Cutoff vs. MAST win %	Unexplored Action Urgency vs. UCT win %	Unexplored Action Urgency vs. MAST win %
Amazons Small	*-N/A-*	*-N/A-*	56.33 (± 5.62)	56.67 (± 5.62)
Battle	87.50 (± 3.62)	73.33 (± 4.97)	46.33 (± 5.52)	54.00 (± 5.50)
Breakthrough	53.00 (± 5.66)	51.67 (± 5.66)	44.33 (± 5.63)	70.67 (± 5.16)
Checkers	72.83 (± 4.79)	74.33 (± 4.71)	48.33 (± 5.41)	52.00 (± 5.33)
Chinook	46.17 (± 5.49)	52.17 (± 5.53)	50.17 (± 5.56)	63.67 (± 5.33)
Knightthrough	54.67 (± 5.64)	47.67 (± 5.66)	51.33 (± 5.67)	61.00 (± 5.53)
Othello	*-N/A-*	*-N/A-*	47.83 (± 5.54)	58.50 (± 5.46)
Pentago	53.33 (± 5.52)	51.00 (± 5.53)	45.17 (± 5.54)	42.00 (± 5.46)
Runners	58.17 (± 5.39)	62.67 (± 5.34)	59.50 (± 5.36)	57.83 (± 5.41)
Skirmish	63.67 (± 5.13)	67.67 (± 4.97)	41.50 (± 5.26)	45.17 (± 5.38)
Overall	**61.17**	**60.06**	**49.08**	**56.15**

Table 2: Combined Extensions Tournaments

Game	Early Cutoff + Unexplored Action Urgency vs. UCT win %	Early Cutoff + Unexplored Action Urgency vs. MAST win %	Early Cutoff + Unexplored Action Urgency vs. RAVE/MAST/FAST win %
Amazons Small	58.33 (± 5.57)	60.67 (± 5.54)	56.33 (± 5.62)
Battle	74.67 (± 4.80)	74.67 (± 4.82)	84.33 (± 3.99)
Breakthrough	52.33 (± 5.66)	68.33 (± 5.27)	73.33 (± 5.01)
Checkers	71.00 (± 4.87)	71.33 (± 4.82)	65.83 (± 5.12)
Chinook	44.33 (± 5.48)	63.83 (± 5.32)	54.33 (± 5.47)
Knightthrough	48.00 (± 5.66)	65.67 (± 5.38)	85.00 (± 4.05)
Othello	45.83 (± 5.58)	54.67 (± 5.57)	47.83 (± 5.52)
Pentago	47.67 (± 5.57)	47.33 (± 5.60)	45.67 (± 5.55)
Runners	62.17 (± 5.43)	59.83 (± 5.47)	56.17 (± 4.52)
Skirmish	58.50 (± 5.28)	61.67 (± 5.29)	54.33 (± 5.41)
Overall	**56.28**	**62.80**	**62.32**

have significant improvements on eight of the ten games plus Othello being close as it did go over the significance threshold in Table 1 and does not activate the Early Cutoff extension.

Regarding the question if these extensions can improve upon the state-of-the-art in GGP we see that there are significant improvements for six of the games resulting in over 12% overall improvement on this set of games.

Related Work

Early cutoffs have been used before in MCTS game playing agents in an effort to get better information through more simulations. INVADERMC (Lorentz 2008) is an MCTS agent made for the game *Amazons* that utilizes this kind of extension. It terminates a simulation when it has reached a fixed number of playout-steps, returning a heuristic evaluation of the reached state. This extension is also used in the game *Lines of Action* (Winands and Björnsson 2009), where it has the more appropriate name *Evaluation Cut-offs*. There the cutoff is not made until the evaluation function has reached a certain score instead of having the length of the simulation dictate when to cut. However, both approaches rely on pre-coded evaluation knowledge.

The Unexplored Action Urgency extension resembles extensions that include domain specific knowledge in selecting between actions in the playout phase. The notion of move urgency was used in (Bouzy 2005) when playing *Go*. There knowledge about *Go* is used to bias the selection probability distribution of playout moves from being uniform to reflect what is known, that is moves that are known to be good become more urgent. Move urgency is widely used in successful *Go* programs.

Conclusion

In this paper we described how two MCTS extensions can be generalized for practical use in GGP agents. The two extensions show genuine improvements to basic agents without, as far as we can tell, displaying any severe adverse effects when they do not help. When merged with a GGP agent that is armed with extensions that allow it to reach competition level strength, both these extensions still offer overall improvements.

References

Bouzy, B. 2005. Associating domain-dependent knowledge and Monte Carlo approaches within a Go program. *Information Sciences* 175(4):247–257.

Clune, J. E. 2008. *Heuristic evaluation functions for general game playing*. PhD dissertation, University of California, Los Angeles, Department of Computer Science.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceeding of the Fifth International Conference on Computers and Games (CG 2006), Turin, Italy*, 72–83.

Enzenberger, M., and Müller, M. 2009. Fuego - an open-source framework for board games and Go engine based on Monte-Carlo tree search. Technical Report 09-08, Dept. of Computing Science, University of Alberta.

Finnsson, H., and Björnsson, Y. 2008. Simulation-based approach to general game playing. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA*, 259–264. AAAI Press.

Finnsson, H., and Björnsson, Y. 2010. Learning simulation control in general game-playing agents. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2008, Atlanta, Georgia, USA*, 954–959. AAAI Press.

Finnsson, H., and Björnsson, Y. 2011. Game-tree properties and MCTS performance. In *The IJCAI Workshop on General Game Playing (GIGA'11)*.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In *Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA*, volume 227, 273–280. ACM.

Gelly, S.; Wang, Y.; Munos, R.; and Teytaud, O. 2006. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA.

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

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proceeding of the Seventeenth European Conference on Machine Learning (ECML 2006), Berlin, Germany*, 282–293.

Lorentz, R. J. 2008. Amazons discover Monte-Carlo. In *Proceedings of the Sixth International Conference on Computers and Games (CG 2008), Beijing, China*, 13–24. Springer-Verlag.

Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E.; and Genesereth, M. 2008. General Game Playing: Game description language specification. Technical Report March 4 2008, Stanford University.

Schiffel, S., and Thielscher, M. 2011. Reasoning about general games described in GDL-II. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11), San Francisco, California, USA*, 846–851. AAAI Press.

Sturtevant, N. R. 2008. An analysis of UCT in multi-player games. In van den Herik, H. J.; Xu, X.; Ma, Z.; and Winands, M. H. M., eds., *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, 37–49. Springer.

Szita, I.; Chaslot, G.; and Spronck, P. 2009. Monte-Carlo tree search in Settlers of Catan. In van den Herik, H. J., and Spronck, P., eds., *Advances in Computer Games*, volume 6048 of *Lecture Notes in Computer Science*, 21–32. Springer.

Thielscher, M. 2010. A general game description language for incomplete information games. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), Atlanta, Georgia, USA*, 994–999. AAAI Press.

Winands, M. H. M., and Björnsson, Y. 2009. Evaluation function based Monte-Carlo LOA. In *Twelfth International Advances in Computer Games Conference (ACG 2009), Pamplona, Spain*. 33–44.

Winands, M. H. M.; Björnsson, Y.; and Saito, J.-T. 2010. Monte Carlo Tree Search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games* 2(4):239–250.