Online Adjustment of Tree Search for GGP

Jean Méhat & Jean-Noël Vittaut
Université de Paris 8 Vincennes–Saint Denis
France
jm@ai.univ-paris8.fr jnv@ai.univ-paris8.fr

Abstract

We present an adaptative method that enables a General Game Player using Monte-Carlo Tree Search to adapt its use of RAVE to the game it plays. This adaptation is done with a comparison of the UCT and RAVE prediction for moves, that are based on previous playout results. We show that it leads to results that are equivalent to those obtained with a hand tuned choice of RAVE usage and better than a fit-for-all fixed choice on simple ad'hoc synthetic games. This is well adapted to the domain of General Game Playing where the player can not be tuned for the characteristics of the game it will play before the beginning of a match.

1 Introduction

It this introduction, we present the General Game Playing (GGP) and the Monte-Carlo Tree Search with UCT and RAVE used by the General Game player we use for this study.

1.1 General Game Playing

Since its definition by the Logic Group of the Stanford University in 2005 [Genesereth *et al.*, 2005], GGP has aroused research in the field of computer playing programs that are able to play a large class of different games without modification.

In GGP, games are described with the General Description Language (GDL); it allows the description of any finite deterministic game with complete information [Genesereth, 2006]. GDL is based on first order logic with negation as failure; most notably it does not include arithmetic that has to be defined as needed in the game description. It is supplemented with a few keywords (see table 1). Based on the *Knowledge Interchange Format* (KIF), GDL has a syntax reminiscent of Lisp and is semantically very similar to Datalog.

An extension of GDL, named GDL2, allows the description of games with incomplete information [Thielscher, 2010]. It adds only a keyword *(sees p x)* to describe the percepts of each player. The player name *random* is also reserved for a source of non-determinism.

(role p)	p is a player
(legal p m)	move m is legal for player p
(does p m)	player p played the move m
terminal	the match is finished
(goal p n)	player p got n points ($0 \leq p \leq 100$)
(init x)	x is true in the initial position
(true x)	x is true in the current position
(next x)	x will be true after the current move

Table 1: GDL keywords used to describe a game in the context of first order logic.

These extensions to GDL have been proved to be sufficient to make it universal [Thielscher, 2011].

Each year since 2005, there is an international GGP competition hosted by the AAAI or IJCAI conferences where different teams pit their players against each other on new games designed by the organizers.

1.2 Monte-Carlo Tree Search

Since 2008, most of the players participating in the GGP competition use some kind of Monte-Carlo Tree Search (MCTS). The base of MCTS is to combine a stochastic sampling of the search space and the buildup of a tree of game positions linked by possible moves.

An exploration is made of four phases: selection of a tree leaf, tree growth, playout to the end of the game and update of the tree nodes.

The *selection* of a tree leaf is done with a descent in the tree. During this descent, previous sampling results are used to select the parts of the tree where exploration is promising. When this descent reaches a node with unplayed moves, a move is selected and a new leaf is built and added to the tree, and a *playout* is performed: successive moves are selected according to a given policy and played until a terminal situation is reached. The result, as described by the game rules, is used to update the nodes and/or edges forming the path built during the descent in the tree.

MCTS methods have been used with great success in the game of Go, where they allow programs to reach the level of the best human players on small boards. They are now applied to many fields of Artificial Intelligence with many variants [Browne *et al.*, 2012]. An attractive characteristic of MCTS is that it does not rely on a heuristic evaluation of a game situation. In GGP there is no general method known to build a reliable heuristic.

Upper Confidence bound applied to Trees
During the descent phase, one has to make a compromise between *exploration*: the selection of less visited branches and *exploitation*: accumulating visits in parts of the tree where previous samplings gave good results. This dilemma is frequently solved using *Upper Confidence bound applied to Trees* (UCT) [Kocsis and Szepesvári, 2006].

With UCT, the move selected during the descent in the tree is one that maximizes

$$\mu_i + C \times \sqrt{log(t)/s_i}$$

where μ_i is the mean result of the playouts starting with the move, t is the total number of playouts played in the current node and s_i is the number of playouts starting with this move. The constant C, named the *UCT constant* is used to adjust the level of exploration of the algorithm: high values favor exploration and low values favor exploitation.

Rapid Action Value Estimates
When using bare UCT, the first move selections in a given node of the tree have to be made according to a statically encoded heuristic (not possible in GGP) or at random, as long as there is not enough playout results to guide the selection. To alleviate this inconvenience, most Go playing program use some variant of *All Moves As First* (AMAF): the back-propagation of the playout results takes into account the move played *and the subsequent moves.*

The currently most common variant of AMAF is *Rapid Action Value Estimates* (RAVE): a node contains a table associating legal moves with the results of all the playouts where these moves were selected during the playout. This table is used to obtain a RAVE estimation that is combined with the UCT estimation in a way that depends on the number of playouts starting with this move: the move choice is principally based on the RAVE estimation when there are few playouts; the importance of the UCT estimation grows with the number of playouts starting with this move.

2 Implementation of RAVE in our General Game Player

We detail here the precise implementation of RAVE in our General Game Player that we used for the experiments.

Each edge in the built part of the game tree contains the mean result of the explorations that went through this edge. Each node contains a RAVE table associating each legal move with the mean result of all the playouts that went through this node where this move was played later on.

In the back-propagation phase, a *RAVE estimation* is updated in each node with the mean playout results for each legal move that was selected during the playout.

In the tree selection part, a *move score* is computed as follows:

- if there was some playout starting with this move, its score is the mean of the results of the playouts that started with this move; if no playout has been played starting with this move, it receives a *default mean score* that we fixed to the maximum possible result after informal tests. This ensures that unexplored moves are preferred over sub-optimal explored moves.
- when the move was used at the beginning of a playout, the *mean score* is replaced by an *UCT score* u using the upper confidence bound computed with the usual UCT formula. This ensures that a move giving always good results in a few playouts will receive a better score than a never explored move. As the number of experiments on this move grows, the upper confidence bound on its mean value will decrease and other moves will be explored. This property is particularly desirable in GGP where the number of playouts can remain small with the usual time settings, due to GDL interpretation time.
- if the move was used later in some playouts, a *RAVE score* r is computed as the mean of these playout results; a *RAVE influence factor* α is computed as $\alpha = \sqrt{k/3t + k}$ and the move final score is $(1 - \alpha)u + \alpha r$; k is the *RAVE equivalence constant* balancing the weight of UCT and RAVE. RAVE will influence the selection more when there are few playouts while UCT will have the greatest influence when the number of playouts grows.

Finally, the move is selected pseudo-randomly between those having the maximum score.

3 Online adjustments of RAVE usage

Finsson et al. show in the context of General Game Playing that RAVE can bring an advantage on some games (Checkers, Othello) while it can be detrimental for some others (Skirmish) [Finnsson and Björnsson, 2010].

What we are interested in is whether it is possible to dynamically adapt the RAVE usage to the characteristics of the game. The player has no knowledge of the characteristics of the game and the static analysis of its properties based on its description is difficult.

One would like to use online learning on the information gathered in the first playouts to deduces some properties of the game. Rave would be used only when it is profitable.

When the tree is built and playout results are used in the back-propagation phase to update the moves characteristics in the nodes, data is accumulated on RAVE and UCT estimations. We investigate how these data can be used to adjust the RAVE usage.

4 Games we use

In this section, we present three games specifically designed to present characteristics where RAVE gives a significant advantage or disadvantage. They are tweaked versions of Sum Of Switches games (SOS).

There are at least two kinds of situations where RAVE is known to hinder the results of explorations. First when a move is good if played as first move but bad if played later; as the move played later leads to bad results, its exploration as a first move is not encouraged by RAVE; it occurs usually in Go in the context of *semeai* or *tsumego* problems where the first move is crucial and has to be played first to be of some value.

RAVE is detrimental in a second kind: when moves are good when played later on but bad if played as first moves. RAVE evaluation is so good that it favors their exploration. It leads to bad choices of the part of the game tree to explore. When the number of explorations is used for the selection of the move to play actually it can lead to the choice of a bad move. This typically occurs in Go with *ko* threats: a *ko* threat is a good move when played in the right time but is silly if played before the beginning of the *ko* fight or when the *ko* can be taken back. We are less interested by this class of games as it does not seem to be a challenge in the current context of GGP.

We use synthetic games to specifically embody these situations: *Blind Cashing Checks* that was also studied in [Tom and Müller, 2010] under the generic name *Sum Of Switches* and another tweaked version *Cashing Stale-dated Checks* with characteristics of the first kind that makes RAVE detrimental. We also present another tweaked version of *Cashing Checks* that we call *Cashing Post Dated Checks* that belongs to the second kind.

4.1 Sums of switches: *Cashing Checks*

Berlekamp and al. present the family of games named *Cashing Checks* [Berlekamp *et al.*, 1982, p. 120]. The material is a set of bearer checks for certain amounts; a player move consists in taking one of the checks for his own. At the end of the game, the sum of each player checks are summed up and the winner is the player that holds the largest amount.

This game is a *Sum Of Switches* (SOS); in the context of Combinatorial Game Theory it is a sum of sub-games the values of which are either $+n$ or $-n$ depending on the player who takes the check; these are *switches*, noted $\pm n$.

The best strategy is naturally to take the check for the largest amount that remains on the board. In a game starting with k checks with amounts n_i, the first player will score $\sum_{1<2i<k} n_{2i}$ and the second $\sum_{0<2i+1<k} n_{2i+1}$ when $n_i \geq n_{i+1}$.

4.2 *Blind Cashing Checks*

The game becomes more interesting when the players are *innumerate*, i.e. are not able to read or compare numbers: they have to select the checks without knowledge of the amount that is written on it. At the end of the game, an arbiter sums the amounts gained by each player and announces the winner. This final result is the only clue the players get on the value of the checks. We call this game *Blind Cashing Checks*. This game was already used to study properties of RAVE [Tom and Müller, 2010].

RAVE works well for a MCTS player at *Blind Cashing Checks*: the player who takes the checks with the largest amounts wins this playout, without consideration of the turn where she took the check, so RAVE will promote the choice of these checks in the first turns, leading to the winning strategy.

The difficulty of the game can be adjusted by varying either k the number of turns or N the number of checks as long as $N > k$. As Tom et al., we use checks referring to the first N non null positive integers and compensate for the first player advantage by setting a *komi* of $k/2$. If both players play optimally the sums of the amounts written on checks held by both players (plus *komi* for the second player) are equal and the game is declared a draw.

4.3 *Cashing Stale-Dated Checks*

We tweak the game of *Cashing Checks* to have a game where RAVE will be detrimental: on each check we add a limit of validity under the form of a turn of the match; if a player takes a check before this turn, she cashes the amount written on the check; if she takes it after this turn, the date is stale and it gives no point at all.

The date of each check is the turn where it is taken when both players play the optimal strategy at *Blind Cashing Checks*. The check with the biggest value amounts to zero after the first move, the second one after the second turn and so on. This modification to the game allows to modelize the situations where RAVE is a disadvantage because a move is good if played as first move but bad if played later.

The limit of validity of the checks introduces another winning strategy: a winning move for a player is either to take the valid check with the largest amount *or* to take the check with the next amount, as the check with the largest amount will give nothing after this move. This way, the first player can force the second player to take a check that amounts to nothing at the last turn if the number of checks and the number of turns are equal. To avoid this issue we use at least one more check than there are turns in a match (i.e. $N > k$).

4.4 *Cashing Post Dated Checks*

Another variation is *Cashing Post Dated Checks*. In this game, each check is dated with a turn; it amounts to nothing if taken before this turn and for the sum written on it if taken on this turn or any subsequent turn.

We set the date for each check according for this amount: a check for a sum of $N - k + i$ is valid only on turn i and subsequent turns. The optimal strategy for both players is then to take the checks in the reverse order, starting with the one that has the least amount and finishing with the one with the biggest amount. A reverse *komi* compensates the advantage of the second player.

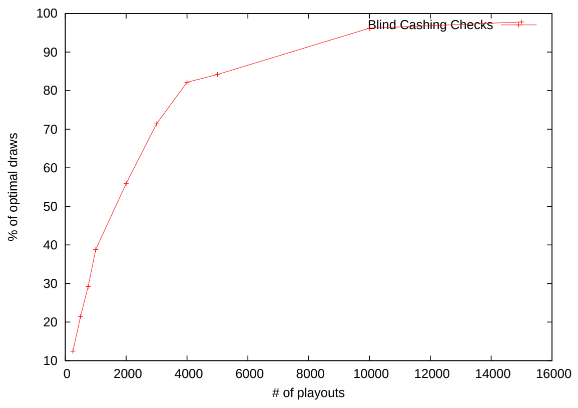

Figure 1: The percentage of matches with optimal draws at *Blind Cashing Checks* as a function of the number of playouts per move.

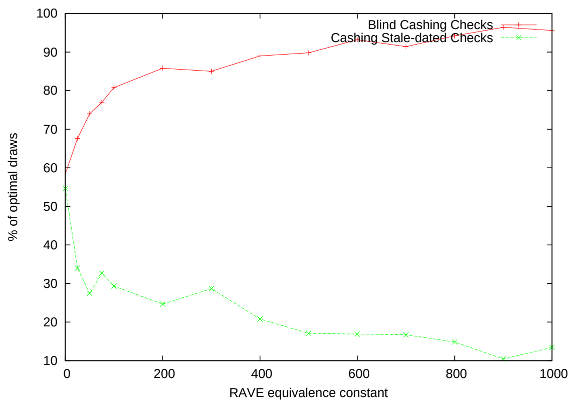

Figure 2: The percentage of matches with optimal moves of both players at *Blind Cashing Checks* and *Cashing Stale-dated Checks* as a function of the RAVE equivalence constant.

This allows to modelize the situation when a move is good if played at the right time but bad if played sooner. We did not use this game in the experiments as this does not seem to be a crucial issue in the current common General Game Playing settings.

5 Experimental settings

For the experiments we have used a GDL description of *Blind Cashing Check* using twenty checks and ten turns with a *komi* of five points. For *Cashing Stale-dated Checks* the number of turns was also set to ten but the number of checks was limited to twelve to give comparable results.

Both players were instances of our General Game Player Ary [Méhat and Cazenave, 2010] in its usual setting: UCT with an exploration constant of 0.4 (actually 40 since the reward of a player can vary between 0 and 100) and transposition tables.

For the experiments, two players with the same parameters played together for 500 complete matches and the percentage of draws with optimal moves by both players (*optimal draw*) was counted; a larger percentage indicates that the players played well, so the value of the parameter is well suited to the task at hand.

To fix the number of playouts used by the players to select a move, we studied the results obtained when this number varies. The results are presented in figure 1: the results are better as the number of playouts grows but not linearly. Later on, we set the number of playouts to 2000 for the experiments, where the number of optimal draws was over 50% in order to leave some space for improvements when RAVE is used.

With the previous settings, the value of the RAVE equivalence constant was made to vary for the games *Blind Cashing Checks* and *Cashing Stale-dated Checks* (see figure 2). As the value of the equivalence constant becomes greater, the level of play gets better at *Blind Cashing Checks* and gets worse at *Cashing Stale-dated Checks*. Here also the influence on the results is more obvious for the small values of the constant.

Given these results, we choose to set the RAVE equivalence constant to 700 for the following experiments, as this value appears in the floor where small modifications of the constant do not modify significantly the results.

6 Using RAVE only for some choices

We first explore what happens when a player uses RAVE for some choices during the descent and does not use it for others. With the RAVE equivalence constant set to 700, we vary the percentage of children selections in the descent phase of UCT where RAVE is used.

As it descends in the built tree, a (pseudo-)random number is used to decide if the next edge will be chosen using only UCT or if RAVE is to be used as described in section 2. The results give a measure of the importance of using RAVE or not using it systematically on every choice.

The results are summarized in figure 3: as expected, the level of play, reflected by the number of draws, diminishes for *Blind Cashing Checks* as RAVE is used more often, while it gets better for *Cashing Stale-dated Checks*.

7 Comparing observed results and RAVE predictions

As the tree is built, the results of the playouts are accumulated in the edges and the nodes to be used as a basis for the predictions of UCT and RAVE. We propose to compare observed results with the RAVE predictions to get an evaluation of when to use RAVE. This evaluation will not be precise, but as shown by the previous experiment, one can expect to get a result that is better than a fit for all setting and that is proportional to the precision of the evaluation.

As a measure of the precision, we sum at the root node the number of playouts for the moves where RAVE gave

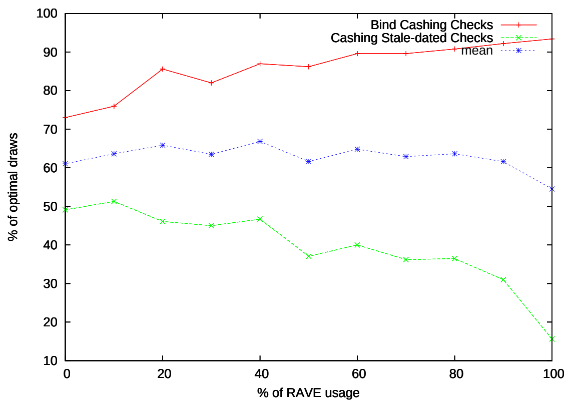

Figure 3: The percentage of draw games with optimal play of both players at *Cashing with Stale-dated Checks* and their mean as a a function of the RAVE usage.

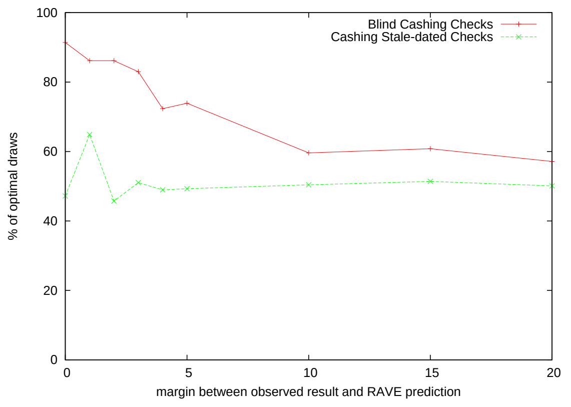

Figure 4: The percentage of optimal draws at *Blind Cashing Checks* and *Cashing Stale-dated Checks* as a a function of the margin used to consider RAVE move prediction for to be too optimistic.

an estimation that was higher than the observed mean result and the number of moves where it is lower, with an error margin. When the number of playouts starting with moves where RAVE was optimistic is greater than the number of moves where it is pessimistic, RAVE is used for the choices of children of the next descent in the built tree.

The results are presented in figure 4. The level of play at *Cashing Stale-dated Checks* stays about the same when the margin augments, while it decreases at *Blind Cashing Checks*.

When the margin is set to 0, the percentage of optimal draws is 91.40 % at *Blind Cashing Checks* and 47.20 % at *Cashing Stale-dated Checks*. This figures are to be compared with those obtained with the best setting for a game: 93.39 % at *Blind Cashing Checks* when always using RAVE and 49.10 % at *Cashing Stale-dated Checks* when never using RAVE: the player adapts successfully its use of RAVE to the game at hand.

8 Conclusion and perspectives

We have shown that it is possible to use the comparison of the mean results observed during playouts with the results predicted by RAVE. We can then the use of RAVE to the characteristics of the game at hand and get an overall result that is equivalent to what one can obtain with an usage of RAVE tuned before the match beginning.

The results presented here were obtained on synthetic games; the observations have to be extended to real games whose characteristics regarding RAVE are less bold and in realistic playing situations where the number of playouts can be much smaller due to slowness of the GDL interpretation.

There are many other ways to measure the correlation between observed playout results and RAVE predictions that have be explored. We intend to investigate if this correlation can be used to adapt the value of the *RAVE* equivalence constant.

The method used here would not work in another synthetic game built by summing the two games used, for example alternating a move in *Blind Cashing Checks* and a move in *Cashing Stale-dated Checks* because the comparison between observed results and RAVE prediction was always calculated on the root node. It would be possible to observe this correlation at every node when it stores enough playouts results.

More generally, the method presented here uses information that is already present in the tree built by the MCTS to determine characteristics of the game it plays. It could also be interesting outside of GGP for games when characteristics vary depending on the position.

References

[Berlekamp *et al.*, 1982] Elwyn R Berlekamp, John Horton Conway, and Richard K Guy. *Winning ways for your mathematical plays. Volume 1.* Academic Press, 1982.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[Finnsson and Björnsson, 2010] Hilmar Finnsson and Yngvi Björnsson. Learning simulation control in general game-playing agents. In *Proc. 24th AAAI Conf. Artif. Intell., Atlanta, Georgia*, pages 954–959, 2010.

[Genesereth *et al.*, 2005] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI magazine*, 26(2):62, 2005.

[Genesereth, 2006] Michael Genesereth. General game playing: Game description language specification, 2006.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–293, 2006.

[Méhat and Cazenave, 2010] Jean Méhat and Tristan Cazenave. Ary, a general game playing program. In *13th board game studies colloquium*, volume 165, pages 168–170, 2010.

[Thielscher, 2010] Michael Thielscher. A general game description language for incomplete information games. In *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI 2010)*, pages 994–999, 2010.

[Thielscher, 2011] Michael Thielscher. The general game playing description language is universal. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pages 1107–1112. AAAI Press, 2011.

[Tom and Müller, 2010] David Tom and Martin Müller. A study of uct and its enhancements in an artificial game. *Advances in Computer Games*, pages 55–64, 2010.

Annex: *Blind Cashing Checks* GDL Description

```
(nstep 10)    ; ten steps
(maxvalue 20) ; twenty checks
(komi 5)      ; komi is 5 points

;; two players
(role left)
(role right)

;; legal moves
(<= (legal ?p ?v )
    (true (control ?p))
    (value ?v)
    (not (true (played ?v ?s))))
(<= (legal ?p noop)
    (true (notcontrol ?p)))

;; alternate play
(init (control left))
(init (notcontrol right))
(<= (next (control ?p)) (true (notcontrol ?p)))
(<= (next (notcontrol ?p)) (true (control ?p)))

;; turns
(init (step 0))
(<= (next (step ?n+1))
    (true (step ?n))
    (+ 1 ?n ?n+1))
(<= terminal (nstep ?nstep) (true (step ?nstep)))

;; maintain sum for each player
(init (sum left 0))
(init (sum right 0))
(<= (next (sum ?p ?n+m)) ; play valid check
    (role ?p)
    (true (sum ?p ?n))
    (does ?p ?m)
    (+ ?m ?n ?n+m))

(<= (next (sum ?p ?n))
    (role ?p)
    (true (sum ?p ?n))
    (does ?p noop))

;; do not take the same check twice (with transpo)
(<= (next (played ?v ?s)) (true (played ?v ?s)))
(<= (next (played ?v somestep)) (does ?p ?v))

;; goal
(<= l>r
    (true (sum left ?l))
    (true (sum right ?r))
    (komi ?k)
    (+ ?k ?r ?r+k)
    (gt ?l ?r+k))
(<= r>l
    (true (sum left ?l))
    (true (sum right ?r))
    (komi ?k)
    (+ ?k ?r ?r+k)
    (gt ?r+k ?l))

(<= (goal left 100) l>r)
(<= (goal left 0) r>l)

(<= (goal right 0) l>r)
(<= (goal right 100) r>l)

(<= (goal ?p 50)
    (role ?p)
    (not l>r)
    (not r>l))

; values of the checks
(<= (value ?n)
    (gt ?n 0)
    (maxvalue ?max)
    (not (gt ?n ?max)))

;; arithmetic: addition, comparison
(<= (+ 0 ?x ?x))
(<= (+ ?a ?b ?a+b)
    (++ ?a-1 ?a)
    (++ ?b ?b+1)
    (+ ?a-1 ?b+1 ?a+b))
(<= (gt ?a ?b) (++ ?b ?a))
(<= (gt ?a ?b) (++ ?a-1 ?a) (gt ?a-1 ?b))

;; integers
(++ 0 1) (++ 1 2) ...
```