Meta Monte-Carlo Tree Search for Automatic Opening Book Generation

Guillaume M.J-B. Chaslot[1], **Jean-Baptiste Hoock**[2], **Julien Perez**[2], **Arpad Rimmel**[2], **Olivier Teytaud**[2] **and Mark H.M. Winands**[1]
[1]Department of Knowledge Engineering, Faculty of Humanities and Sciences, Maastricht University, The Netherlands
[2]TAO (INRIA), LRI, Université Paris-Sud, France

Abstract

Modern game playing programs use opening books in order to perform better. Generating opening books automatically in combination with an $\alpha\beta$ program has been well studied. A challenge is to generate automatically an opening book for the new Monte-Carlo Tree Search (MCTS) algorithms. In this article, we tackle this issue by combining two level of MCTS. The resulting algorithm is called Meta-MCTS. Instead of applying a simulation strategy, it uses an MCTS program to play a simulated game. We describe two Meta-MCTS algorithms: the first one, Quasi Best-First, favors exploitation. The second one, Beta Distribution Sampling, favors exploration. Our approach is generic and can be used for general game playing. It will be particularly useful when there is off-line time available. In order to evaluate the performances of these algorithms, we generated and tested 9×9 Go opening books.

1 Introduction

Monte-Carlo Tree Search (MCTS) is a new best-first search that appeared in 2006 [Chaslot *et al.*, 2006b; Coulom, 2007; Kocsis and Szepesvári, 2006]. It caused a revolution in the field of Computer Go: whereas non-MCTS programs were not able to defeat most of the amateurs, MCTS programs are now able to defeat professionals for the first time on the 9×9 board. MCTS also led to state-of-the-art programs in several games, such as Amazons [Lorentz, 2008], Production Management Problems [Chaslot *et al.*, 2006a], SameGame [Schadd *et al.*, 2008], LOA [Winands *et al.*, 2008], and general game playing [Finnsson and Björnsson, 2008]. MCTS programs need, just like $\alpha\beta$ programs, an opening book to perform better. There have been a number of attempts to create opening books for $\alpha\beta$ based programs [Buro, 1999; Karapetyan and Lorentz, 2006; Lincke, 2002]. Because the proposed methods so far are designed for programs based on a static evaluation function, it is a challenge to generate an opening book for an MCTS program. In this article we propose to tackle this issue by combining two levels of MCTS. The algorithm is called Meta Monte-Carlo Tree Search. Instead of using a weak simulation strategy, it uses an entire MCTS program (MoGo) to play a simulated game.

This approach is particularly useful when off-line computations can be performed. For instance, it can be applied in a general game playing competition when there is some time between the release of the rules and the actual competition. Given an extensive start-clock, the best use of that time in a simulation based agent - an approach quite popular in contemporary GGP agent [Finnsson and Björnsson, 2008] - might actually be to build an opening book.

As a test domain we will use 9×9 Go. This game is interesting, since the MCTS programs are reaching the level of professional players.

The structure of the article is as follows. In Section 2, we describe the MCTS mechanism. Section 3 presents previous research on creating opening books. Next, in Section 4 we discuss Meta-MCTS and propose two algorithms, Quasi Best-First and Beta Distribution Sampling. Subsequently, Section 5 presents the experimental results. Finally, Section 6 concludes and gives insights for future research.

2 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [Chaslot *et al.*, 2006b; Coulom, 2007; Kocsis and Szepesvári, 2006] is a best-first search method that does not require a positional evaluation function. It is based on randomized explorations of the search space. Using the results of previous explorations, the algorithm gradually grows a game tree in memory, and successively becomes better at accurately estimating the values of the most promising moves.

MCTS consists of four strategic phases, repeated as long as there is time left. The phases are as follows. (1) In the *selection step* the tree is traversed from the root node until it selects a leaf node L that is not added to the tree yet. (2) Subsequently, the *expansion strategy* is called to add the leaf node L to the tree. (3) A *simulation strategy* plays moves in a self-play mode until the end of the game is reached. The result R of such a "simulated" game is $+1$ in case of a win for Black (the first player in Go), 0 in case of a draw, and -1 in case of a win for White. (4) A *backpropagation strategy* propagates the results R through the tree, i.e., in each node traversed the average result of the simulations is computed.

3 Automatic Opening Book Generation

For programs using $\alpha\beta$ search, there are quite a few methods for generating opening books automatically. The most popular is based on the concept of the drop-out mechanism [Karapetyan and Lorentz, 2006; Lincke, 2002]: it explores the move that maximizes the $\alpha\beta$ score minus a certain depth penalty. The depth penalty increases with the distance to the most-promising leaf node. It enables that a player drops out the book quickly only when the position is quite advantageous for him. The application of this mechanism to 9×9 Go raises a problem: there is no fast and efficient evaluation function available in Go for using an $\alpha\beta$ search. It would be possible to replace the $\alpha\beta$ search by MCTS. However, the score of a MCTS search is quite stochastic (instable), in contrast to the stable minimax score of an $\alpha\beta$ search. This could have negative effect when constructing the opening book. Using an MCTS variant to generating an opening book appears to be more natural and elegant. We will discuss this further in the next section. Finally, we remark that in some games, programs evolve so fast that a good opening book may becomes out-dated quickly. Some programs have therefore shifted to online verification of the book moves [Donninger and Lorenz, 2006].

4 Meta Monte-Carlo Tree Search

In this section, we first give the general structure of Meta Monte-Carlo Tree Search (Subsection 4.1). In Subsection 4.2, we describe the Quasi Best-First algorithm. In Subsection 4.3, we introduce the Beta Distribution Sampling algorithm.

4.1 General Idea

An MCTS program uses a weak simulation strategy in order to derive a good policy from it. The idea of Meta-MCTS consist of replacing the weak simulation strategy at the lower part of the search by an entire MCTS program (e.g., our Go program MoGo, or a general game playing program based on MCTS, such as [Finnsson and Björnsson, 2008]). This program is the lower level of the Meta-MCTS. Cazenave got the world record in the one-player game "Morpion Solitaire" by using an online Meta-MCTS composed of two UCT algorithms [Cazenave, 2007]. However, this approach is designed for one-player games and gives poor results when applied to two player games, as for instance Go. We show in this paper that using a Meta-MCTS off-line for generating an opening book is possible for these games.

We call the part of the search where the selection strategy decides which move will be explored further, the upper level. This selection strategy has to be adapted as well. The standard UCT [Kocsis and Szepesvári, 2006] algorithm requires an exploration constant to be tuned. Tuning such a constant for a two-level MCTS would take quite an amount of time. Therefore, we propose two alternatives: Quasi Best-First and Beta Distribution Sampling. They are described in the following subsections.

4.2 Quasi Best-First

MCTS is often emphasized as a compromise between exploration and exploitation. Nevertheless, many practitioners have seen that in the case of deterministic games, the exploration constant, when optimized, has to be set close to zero. A small exploration is given to every move using a specific strategy, for instance RAVE [Gelly and Silver, 2007] or Progressive Bias [Chaslot *et al.*, 2008]. In both cases the exploration term will converge fast to zero. The consequence of using such a small exploration is that, after a few games, a move is further analyzed as long as it is the move with the highest winning rate. Hence, most MCTS programs can be qualified as being greedy. We already proposed the algorithm Quasi Best-First (QBF), previously called MVBM in [Audouard *et al.*, 2009]. It usually selects the child with the highest winning rate. However, if a move's wining rate drops below a certain threshold K, QBF will ask MoGo to chose a move. The pseudo-code is given in Algorithm 1.

Algorithm 1 The "Quasi Best-First" (QBF) algorithm. λ is the number of machines available. K is a constant. g is a game, defined as a sequence of game states. The function "MoGoChoice" asks MoGo to choose a move.

```
QBF(K, λ)
while True do
  for l = 1..λ, do
    s =initial state; g = {s}.
    while s is not a final state do
      bestScore = K
      bestMove = Null
      for m in the set of possible moves in s do
        score =percentage of won games by playing the
        move m in s
        if score > bestScore then
          bestScore = score
          bestMove = m
        end if
      end for
      if bestMove = Null then
        bestMove = MoGoChoice(s) // lower level
        MCTS
      end if
      s = playMove(s, bestMove)
      g = concat(g, s)
    end while
    Add g and the result of the game in the book.
  end for
end while
```

4.3 Beta Distribution Sampling

Each node of the tree has a game-theoretic value, which is either 0 in case it corresponds to a won position for White, or 1 in case it corresponds to a won position for Black. But in practice, the convergence to the game-theoretic value is slow. We observe that the value of a node may get stuck in a local optima for a long time. From this observation, we make an hypothesis of stability H_s: each position P has a stationary average value $\mu_{s,P}$ that only depends on P and on the simulation strategy s that is used. For instance, the standard version of MoGo uses $\mu_{fastPattern,P}$, where

$fastPattern$ is a fast simulation strategy that uses 3×3 patterns to make its decision. The upper level of the Meta-MCTS uses $\mu_{MoGoGames,P}$, where $MoGoGames$ is a simulation strategy that uses MoGo to make its decision.

Let $w_{s,P}$ be the number of victories from the games made by the simulation strategy s, which went through the position P. Let $l_{s,P}$ be the number of defeats from the games made by the simulation strategy s, which went through the position P. Under the hypothesis H_s, the probability that the game is a victory for the player to move in position P, is $\mu_{s,P}$. The number of victories and defeats obeys a Bernoulli distribution. Hence, the probability distribution of $\mu_{s,P}$ knowing $w_{s,P}$ and $l_{s,P}$ is given by the conjugate prior of the Bernoulli distribution, which is:

$$p(\mu_{s,P} = x | w_{s,P}, l_{s,P}) = x^{w_{s,P}} \cdot (1-x)^{l_{s,P}}$$

We propose the following selection strategy, called Beta Distribution Sampling (BDS), which consists of sampling a random number r_i from each beta distribution for each child i.[1] The child selected is the one with the best r_i. The pseudo code is provided in Algorithm 2. According to this selection strategy, each node is selected with the probability that it is the best node, assuming the hypothesis H_s. This concept is similar to the idea of the selection strategy developed in [Chaslot *et al.*, 2006b; Coulom, 2007]. The benefit of our new approach is that there are less approximations.

Algorithm 2 The "Beta Distribution Sampling" (BDS) algorithm. λ is the number of machines available. g is a game, defined as a sequence of game states. The function "MoGoChoice" asks MoGo to choose a move.

```
BDS(λ)
while True do
  for l = 1..λ, do
    s =initial state; g = {s}.
    while s is not a final state do
      bestScore = −∞
      bestMove = Null
      for m in the set of possible moves in s do
        score =draw from distribution:
        x → x^{w_{MoGoGames,m}} · (1 − x)^{l_{MoGoGames,m}}
        if score > bestScore then
          bestScore = score
          bestMove = m
        end if
      end for
      if bestMove = Null then
        bestMove = MoGoChoice(s) // lower level MCTS
      end if
      if random_int modulo s.visit_count = 0 then
        bestMove = MoGoChoice(s) // lower level MCTS
      end if
      s = nextState(s, bestMove) (transition operator)
      g = concat(g, s)
    end while
    Add g and the result of the game in the book.
  end for
end while
```

5 Experiments

In this section, we generate several 9×9 Go opening books using QBF and BDS. We evaluated their performances and provide statistics that help understanding the structure of these books. All these opening books were generated on a grid.[2] For all experiments the symmetry of the board positions was taken into account.

Subsection 5.1 present experiments on QBF, and subsection 5.2 present a comparison of QBF and BDS.

5.1 Experiments on QBF

In this subsection we show the performances of QBF for 9×9 Go. First, we perform experiments with $K = 0.5$ in QBF. Next, experiments with different time settings for the lower level are presented. Finally, we present tests in self-play and with an expert opening book. In order to give more insight on QBF, we present some of the experiments from [Audouard *et al.*, 2009].

Experiment with a QBF constant of 0.5

In the following series of experiments we tested the quality of the QBF generated opening book with a constant *K* of 0.5. When generating the book the program MoGo used 10 seconds for choosing a move at the lower level. The generated QBF book contained 6,000 games. For evaluating the quality of the QBF book we matched two versions of MoGo against each other. One was using the QBF book and the other one did not use a book at all. Both programs received 10 seconds thinking time per move and played on an 8-core machine. Moreover, we also matched the program using the QBF book against one using an "expert book". This expert opening book has been designed specially for MoGo by Pierre Audouard.[3] The results are given in Table 1.

The first column gives an average success rate of 50%, since it is self-play. The second column shows the results for White (resp. Black) with the QBF book against no book. We see that the one using an opening book performs significantly better. In the third column we see that the QBF book also outperforms the expert book. However, in both cases we observe that the Black player does not improve when using the QBF book. This can be explained as follows: as long as Black has not found a move with success rate $> 50\%$, it always asks MoGo for a move to play. Therefore, White improves its results by choosing moves with a high success rate, but not Black. This is why that in the remainder of the paper, we will use $K = 10\%$ for Black. Table 2 shows that this setting also improves the level as Black).

[1] We used the scientific library [Blitz++, 2006] to draw random numbers according to a beta distributions.

[2] The grid was Grid5000, well-suited for large scale scientific experiments.

[3] Pierre Audouard was the French Champion in 19×19 Go and is the current World Champion in 9×9 Go for people with a physical handicap.

Table 1: Performance of the QBF algorithm with 10 seconds per move and $K = 0.5$. The confidence interval is $\pm$ 1.9 %

	No book vs. no book	QBF book vs. no book	QBF vs. expert book
White	51.5 %	64.3 %	64.1 %
Black	48.5 %	48.0 %	46.1 %
Average	50.0 %	56.2 %	55.1 %

Time Settings

The results above look promising, so we tested what happens if we use this QBF book, constructed with 10 seconds a move at the lower level, in a stronger version of MoGo (i.e., a version using more time). We matched again a program with the QBF book against a program with the expert book. Both programs had now 60 seconds thinking time for each move. The success rate of the QBF book against expert book was: 30.6 % as Black, 40.7 % as White. The QBF book constructed with 10 seconds a move, is not sufficient strong enough when MoGo was thinking 60 seconds per move. However, the expert book is more robust when the programs used more time. In the case where the thinking time was set to 60 seconds, the success rate of the expert book against no book was: 50 % as Black, 63 % as White. Moreover, even in the case the thinking time was set to 300 seconds, the success rate of the expert book against no book was: 49.6 % as Black, 57.0 % as White.

The QBF results show that a weak player cannot estimate efficiently an opening sequence. Whenever this weak player plays hundreds of games - the opening sequence might improve the weak-player, but this opening sequence will become a handicap when the player will become stronger. This result explains why we may not use a fast MCTS player to build the opening book. The conclusion is that the time setting at the lower level when constructing the book should be of the same order as the time setting of the program that will use the book.

QBF in Self-Play and against Expert Opening Book

In this section we generate a QBF by using more time at the lower level. Instead of using 10 seconds a move we used 12 hours for the complete game (six hours for each side) on a quad-core machine. The final book consisted of 3000 games.

We tested the quality of the QBF book by matching two versions of MoGo against each other. One version was using the book, the other was not. The time setting was six hours for each side. The results are presented in Table 2. For comparison reasons we also tested in a similar way the quality of the expert book. We observe that MoGo performs better when using the QBF book than when using the expert book. Finally, we see that QBF book improves the performance of both colors.

5.2 Comparison between QBF and BDS

We considered QBF within self-play experiments. However, it should be remarked that self-play experiments favor the greedy strategies [Lincke, 2002]. Hence, a comparison of QBF and BDS on this basis would be biased. We propose

Table 2: Success rate of the QBF book and expert book against the default MoGo using 6 hours for each side

	QBF opening book	Expert opening book
White	74.5% $\pm$ 2.3 %	62.9 % $\pm$ 3.8 %
Black	64.6 % $\pm$ 2.4 %	49.7 % $\pm$ 3.8 %
Average	69.6 % $\pm$ 2.4 %	56.3 % $\pm$ 3.8 %

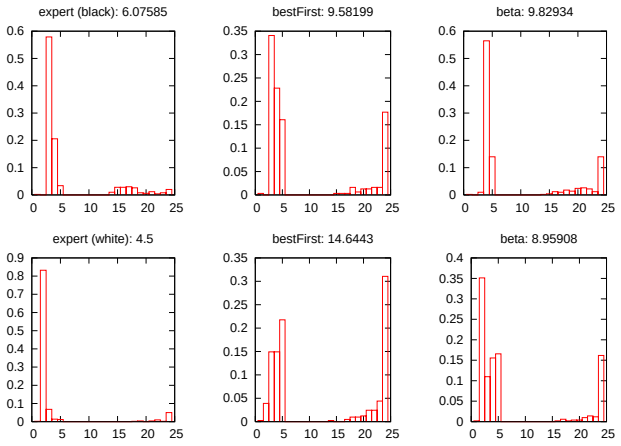

Figure 1: Number of book moves when playing against FUEGO with the opening book. The x-axis is the depth and y-axis is the density of probability. First row: playing as Black. Second row: playing as White. First column: expert opening book. Second column: QBF opening book of $12,000$ games. Third column: BDS opening book of $12,000$ games. Each label contains the average length of the book. All histograms are estimated on 500 games.

a different way of comparison. First, we measure the length of play against other opponents. Next, we compare QBF and BDS on the computer Go server CGOS.

Comparison of length of play against different opponents.

We compare the QBF book consisting of $18,000$ games to BDS book consisting of $12,000$ games. We remark that QBF benefits from a larger set.

Figure 1 shows the distribution of the length of staying in the opening book when playing 500 games as Black and 500 games as White against FUEGO [Fuego, 2009]. This program is quite similar to MoGo. In the figure we see that as Black QBF has an average length of 9.58 and BDS has an average length of 9.83. As White, QBF has an average length of 14.64 whereas BDS has only a length of 8.96. As FUEGO is quite close to MoGo, the opening book generated by QBF is a good predictor; yet it missed some moves for Black. We may conclude that QBF builds a book that is asymmetrical in considering Black and White. Because Black has the disadvantage, its best move value will go below K more often than it would be for White.

In the following series of experiments, we played against

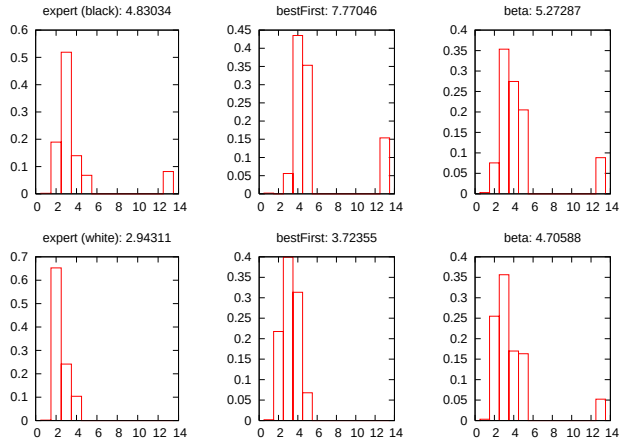

Figure 2: Number of book moves when playing against GNU GO with the opening book. The x-axis is the depth and y-axis is the density of probability. First row: playing as Black. Second row: playing as White. First column: expert opening book. Second column: QBF opening book of $12,000$ games. Third column: BDS opening book of $12,000$ games. Each label contains the average length of the book. All histograms are estimated on 500 games.

GNU GO [Gnu Go, 2004]. This program is not MCTS-based and therefore much more different from MOGO than FUEGO. Figure 2 shows the distribution of the length of staying in the opening book when playing 500 games as Black and 500 games as White. It is clear that as White BDS stayed longer in the opening book than QBF, 4.7 and 3.7 moves, respectively. However, as Black BDS stayed shorter in the opening book than QBF, 5.27 and 7.77, respectively.

In the next series of experiments we compared the number of moves staying in the opening book against expert human opponents. All the responses that are found in the classic 9×9 Go book[4] are also found in the QBF book.

Unfortunately, an in-depth analysis showed that when QBF was generating the book it soon selected always **e5** as the first move. Hence, all other opening moves in the initial position were only explored a small number of times. This was a clear drawback of the QBF approach when playing against human players. This was observed in the game against Prof. Tsai (6D), who played against the QBF book consisting of $18,000$ games.

- First, second, third games: Prof. Tsai played in a classic way. Four moves were in the book. Prof. Tsai won these two games as White.
- Fourth game: after three moves in the book, MOGO gets a good position and wins.
- Fifth, sixth, seventh games and eight: only two book moves, but a good position for MOGO. MOGO won.
- Ninth game: Prof. Tsai played the sequence **g3-d6-d4**; MOGO had no more than one book move. MOGO played correctly this opening of the game, but made a mistake later that would have been less likely if MOGO had saved some time by using more book moves. Interestingly, the BDS approach, with a smaller set of games, would have played five book moves instead of only one. QBF had missed an important opening that BDS would not have missed.

[4] See `http://senseis.xmp.net/?9x9Openings`

Table 3: Results on the computer Go server CGOS

QBF	BDS	CGOS rating	Games
No	No	2216	371
Yes	No	2256	374
No	Yes	2268	375
Yes	Yes	2237	373

Another example can be found in the official match played against Motoki Noguchi, 7 dan, with 30 minutes sudden death per side. The result of the match was 2-2. This was the first time that a computer program was able to draw against a player of that calibre. These games were played with the QBF and expert book, with $6,000$ games. In the games won by MOGO, the opening book gave an advantage to MOGO and continuously increased its advantage. In both lost games, Motoki Noguchi went out of the opening book quite fast. However, the book later developed by BDS would have contained more moves. The sequence **e5-c4-g3-e3-e4-d4** was explored only 22 times by the QBF opening book, but 424 times by the BDS. In the other game lost by MOGO, the sequence **e5-e7-g6-f3** has been explored 13 times by QBF and 18 times by BDS.

This shows that, despite the quite long sequence in the opening book against computers, QBF does not explore enough promising moves for playing against humans. BDS appears to be a better alternative against human play.

Comparison on the Go Server CGOS.

We used the Go server CGOS in order to assess the different algorithms. In order to perform a fair comparison between the algorithms, we created two dedicated opening books. The first was created by QBF and the second by BDS. Each opening book was created by using 32 cores in parallel, 1 second per move, with a total of $5,120$ games. We launched four versions of MOGO on CGOS, one without a book, one with a QBF book, one with a BDS book, and one with a combined book. Subsequently, we compared the ELO rating that they obtained by playing against a pool of different opponents. In order to make the comparison as fair as possible, we launched both program simultaneously on the server. Moreover, to avoid that the different MOGO versions played too many games against each other, we only launched them when there were enough other programs available. Hence, the different versions of MoGo played around 70% of their games against non-MOGO versions. The results can be found in Table 3. The conclusion of this experiment is that QBF and BDS present a significant improvement on the version without opening book, and that merging directly the two opening books is counter-productive. The two books were not built to be combined with each other, so negative side effects may appear.

6 Conclusion

In this paper, we used a Meta Monte-Carlo Tree Search (MCTS) in order to build opening books. Meta-MCTS is similar to MCTS, but the random player is replaced by a standard MCTS program. We described two algorithms for Meta-MCTS: Quasi Best-First (QBF) and Beta Distribution Sampling (BDS). The first algorithm, QBF is an adaptation of greedy algorithms that are used for the regular MCTS. It therefore favors more exploitation. During this events we noticed that despite the good performances of the opening book, some branches were not explored sufficiently. The second algorithm, that we call BDS, favors exploration more. BDS does not need an exploration/exploitation coefficient to be tuned. This approach resulted in an opening book which is shallower and larger. The book has the drawback to be less deep against computers, but the advantage is that it stayed longer in the book in official games against humans. Experiments on Go server CGOS server revealed that both QBF and BDS were able to improve MoGo. In both cases the improvement was more or less similar. We presented different results to show the efficiency of our algorithms on the game of 9×9 Go. However, our methodology is independent of the game and can be easily used for general game playing. Furthermore, Meta-MCTS does not rely on any domain-dependent static evaluation function, which is a major advantage for general game playing. We believe that Meta-MCTS will be particularly useful for games with a small branching factor in the opening, and when offline computational time is available.

As future research, we will experiment with different ways to build an opening book. In particular, adapting classic techniques derived from $\alpha\beta$ search to MCTS constitutes an interesting challenge. Moreover, we want to try our Meta-MCTS approach in combination with a GGP program.

Acknowledgments.

We first would like to thank the Go experts who helped us to build and test the opening book, in particular Pierre Audouard, Prof. Tsai, Motoki Noguchi, Jérôme Hubert, Frédéric Donzet, and François Mizessyn. We would like to thank the providers of computer time, Grid5000 and NCF/SARA. We are very grateful to the following persons for giving their advice: B. Bouzy, T. Cazenave, A. Cohen, R. Coulom, C. Fiter, R. Munos, Z. Yu, the computer-go mailing list, the KGS community and the the CGOS server.

References

[Audouard *et al.*, 2009] P. Audouard, G. Chaslot, J-B. Hoock, A. Rimmel, J. Perez, and O. Teytaud. Grid coevolution for adaptive simulations; application to the building of opening books in the game of Go. In *EvoGames*, volume 5484 of *Lecture Notes in Computer Science (LNCS)*, pages 323–332. Springer, 2009.

[Blitz++, 2006] Blitz++. Library for scientific computing. http://www.oonumerics.org/blitz/, 2006.

[Buro, 1999] M. Buro. Toward opening book learning. *ICCA Journal*, 22(2):98–102, 1999.

[Cazenave, 2007] T. Cazenave. Reflexive monte-carlo search. In *Proceedings of the CGW*, pages 165–173, 2007.

[Chaslot *et al.*, 2006a] G.M.J-B. Chaslot, S. De Jong, J-T. Saito, and J.W.H.M. Uiterwijk. Monte-Carlo Tree Search in Production Management Problems. In P.-Y. Schobbens, W. Vanhoof, and G. Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, pages 91–98, 2006.

[Chaslot *et al.*, 2006b] G.M.J-B. Chaslot, J-T. Saito, B. Bouzy, J.W.H.M. Uiterwijk, and H.J. van den Herik. Monte-Carlo Strategies for Computer Go. In P.-Y. Schobbens, W. Vanhoof, and G. Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence*, pages 83–90, 2006.

[Chaslot *et al.*, 2008] G.M.J-B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, and B. Bouzy. Progressive strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, 4(3), 2008.

[Coulom, 2007] R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H.J. van den Herik, P. Ciancarini, and H.H.L.M. Donkers, editors, *Proceedings of the 5th International Conference on Computer and Games*, volume 4630 of *Lecture Notes in Computer Science (LNCS)*, pages 72–83. Springer-Verlag, Heidelberg, Germany, 2007.

[Donninger and Lorenz, 2006] C. Donninger and U. Lorenz. Innovative opening-book handling. In H.J. van den Herik, S c. Hsu, T s. Hsu, and H.H.L.M. Donkers, editors, *ACG*, volume 4250 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2006.

[Finnsson and Björnsson, 2008] H. Finnsson and Y. Björnsson. Simulation-based approach to general game playing. In D. Fox and C.P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 259–264. AAAI Press, 2008.

[Fuego, 2009] Fuego. Games group of the university of alberta. http://fuego.sourceforge.net/, 2009.

[Gelly and Silver, 2007] S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM Press.

[Gnu Go, 2004] Gnu Go. Free software foundation. http://www.gnu.org/software/gnugo, 2004.

[Karapetyan and Lorentz, 2006] A. Karapetyan and R.J. Lorentz. Generating an opening book for amazons. In *Computers and Games (CG 2004)*, volume 3846 of *Lecture Notes in Computer Science (LNCS)*, pages 13–24, 2006.

[Kocsis and Szepesvári, 2006] L. Kocsis and C. Szepesvári. Bandit Based Monte-Carlo Planning. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Machine Learning: ECML 2006, Lecture Notes in Artificial Intelligence 4212*, pages 282–293, 2006.

[Lincke, 2002] T. Lincke. Strategies for the automatic construction of opening books. In *CG '00: Revised Papers from the Second International Conference on Computers and Games*, pages 74–86, London, UK, 2002. Springer-Verlag.

[Lorentz, 2008] R.J. Lorentz. Amazons discover monte-carlo. In H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands, editors, *Computers and Games (CG 2008)*, volume 5131 of *Lecture Notes in Computer Science (LNCS)*, pages 13–24, 2008.

[Schadd *et al.*, 2008] M.P.D. Schadd, M.H.M. Winands, J.W.H.M. Uiterwijk, and H.J. van den Herik. Single-Player Monte-Carlo Tree Search. In H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, editors, *Computers and Games*, pages 1–12, Beijing, China, 2008. Springer.

[Winands *et al.*, 2008] M.H.M. Winands, Y. Björnsson, and J-T. Saito. Monte-carlo tree search solver. In H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands, editors, *Computers and Games (CG 2008)*, volume 5131 of *Lecture Notes in Computer Science (LNCS)*, pages 25–36, 2008.