Monte-Carlo Simulation Balancing in Practice

Shih-Chieh Huang[1], Rémi Coulom[2], and Shun-Shii Lin[1]

[1] National Taiwan Normal University, Dept. of CSIE, Taiwan, R.O.C
[2] Université de Lille, CNRS, INRIA, France

Abstract. Simulation balancing is a new technique to tune parameters of a playout policy for a Monte-Carlo game-playing program. So far, this algorithm had only been tested in a very artificial setting: it was limited to 5×5 and 6×6 Go, and required a stronger external program that served as a supervisor. In this paper, the effectiveness of simulation balancing is demonstrated in a more realistic setting. A state-of-the-art program, Erica, learned an improved playout policy on the 9×9 board, without requiring any external expert to provide position evaluations. Evaluations were collected by letting the program analyze positions by itself. The previous version of Erica learned pattern weights with the minorization-maximization algorithm. Thanks to simulation balancing, its playing strength was improved from a winning rate of 69% to 78% against Fuego 0.4.

1 Introduction

The standard approach to writing Go-playing programs is now Monte-Carlo tree search. This idea was introduced about 20 years ago [1,2], but it is only recently that it became successful and popular [3,4,5]. The basic idea of Monte-Carlo algorithms consists in evaluating positions by averaging the outcome of random continuations.

Monte-Carlo evaluation of a position depends on the choice of a probability distribution over legal moves. A uniform distribution is the simplest choice, but produces poor evaluations. It is often better to play good moves with a higher probability, and bad moves with a lower probability. Playout policy has a lot of influence on playing strength, and several methods have been proposed to optimize it.

The simplest approach to policy optimization is trial and error. Some knowledge is implemented in playouts, and its effect on playing strength is estimated by measuring winning rate against other programs [5,6,7,8]. This approach is often slow and costly, because measuring winning rate by playing games takes a lot of time, and a lot of trials fail. It is difficult to guess what change in playout policy will make the program stronger, because making playouts play better often causes the Monte-Carlo program to get weaker [9,10].

In order to avoid the difficulties of crafting a playout policy manually, some authors tried to establish principles for automatic optimization. First, it is possible to directly optimize numerical parameters with generic stochastic optimization

algorithms such as the cross-entropy method [11]. Such a method may work for a few parameters, but it still suffers from the very high cost of measuring strength by playing games against some opponents. This cost may be overcome by methods such as reinforcement learning [9,10,12], or supervised learning from good moves collected from game records [13]. Supervised learning from game records has been very successful, and is used in some top-level Go programs such as Zen or Crazy Stone.

Among the reinforcement-learning approaches to playout optimization, a recent method is simulation balancing (SB) [12]. It consists in tuning continuous parameters of the playout policy in order to match some target evaluation over a set of positions. This target evaluation is determined by an expert. It may be obtained by letting a strong program analyze positions deeply, for instance. Experiments reported by Silver and Tesauro indicate that this method is very promising: they measured a 200 Elo improvement over previous approaches.

SB experiments were promising, but not completely convincing, because they were not run in a realistic setting. They were limited to 2×2 patterns of stone configurations, on the 5×5 and 6×6 Go boards. Moreover, they relied on a much stronger program, Fuego [14], that was used to evaluate positions of the training database. Anderson [15] failed to replicate the success of SB for 9×9 Go, but may have had bugs, because he did not improve much over uniform-random playouts. So it was not clear whether this idea could be applied successfully to a state-of-the-art program.

This paper presents the successful application of SB to Erica, a state-of-the-art Monte Carlo program. Experiments were run on the 9×9 board. The training set was made of positions evaluated by Erica herself. So this learning method does not require any external expert supervisor. Experiment results demonstrate that SB made the program stronger than its previous version, where patterns were trained by minorization-maximization (MM) [13]. Besides playing strength, another interesting result is that pattern weights computed by MM and SB are very different from each other. SB patterns may want to play some very bad shape, that MM evaluates very badly, but that helps to get a correct playout outcome.

2 Description of Algorithms

This section is a brief reminder of the MM [13] and SB [12] algorithms. More details about these algorithms can be found in the references.

2.1 Softmax Policy

Both MM and SB optimize linear parameters of a Boltzmann softmax policy. Such a policy is defined by the probability of choosing action a in state s:

$$\pi_\theta(s,a) = \frac{e^{\phi(s,a)^T\theta}}{\sum_b e^{\phi(s,b)^T\theta}} \ ,$$

where $\phi(s,a)$ is a vector of binary features, and θ is a vector of feature weights. The objective of learning algorithms is to find a good value for θ.

2.2 Supervised Learning with MM

MM learns feature weights by supervised learning over a database of sample moves. It computes maximum-a-posteriori values for θ, given a prior distribution and sample moves. Typically, the training set is made of moves extracted from game records of strong players. It may also be made of self-play games if no expert game records are available.

2.3 Policy-Gradient Simulation Balancing (SB)

SB does not learn from examples of good moves, but from a set of evaluated positions. This training set may be made of random positions evaluated by a strong program, or a human expert. Feature weights are trained so that the average of playout outcomes matches the target evaluation given in the training set.

The details of SB are given in Algorithm 1. In this algorithm, $\psi(s,a)$ is defined by:

$$\psi(s,a) = \nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \sum_b \pi_\theta(s,b)\phi(s,b) \ .$$

$V^*(s_1)$ is the target value of position s_1. α is the learning rate of steepest descent. z is the outcome of one playout, from the point of view of the player who made action a_1 (+1 for a win, -1 for a loss, for instance). s_i and a_i are successive states and actions in a playout of T moves. M and N are integer parameters of the algorithm. V and g are multiplied in the update of θ, so they must be evaluated in two separate loops, in order to obtain two independent estimates.

Algorithm 1 Policy-Gradient Simulation Balancing (SB)

$\theta \leftarrow 0$
for all $s_1 \in$ training set **do**
  $V \leftarrow 0$
  for $i = 1$ to M **do**
    simulate(s_1, a_1, ..., s_T, a_T; z) using π_θ
    $V \leftarrow V + \frac{z}{M}$
  end for
  $g \leftarrow 0$
  for $j = 1$ to N **do**
    simulate(s_1, a_1, ..., s_T, a_T; z) using π_θ
    $g \leftarrow g + \frac{z}{NT} \sum_{t=1}^{T} \psi(s_t, a_t)$
  end for
  $\theta \leftarrow \theta + \alpha(V^*(s_1) - V)g$
end for

3 Experiments

Experiments were run with the Go-playing program Erica. The SB algorithm was applied repeatedly with different parameter values, in order to measure their effects. Playing strength was estimated with matches against Fuego. The result of applying SB is compared to MM, both in terms of playing strength and feature weights.

3.1 Erica

Erica is developed by the first author as a PhD research. The development of Erica is supervised by the second author and project-supported by the third author. In 2009, Erica won the 3rd and 2nd position in 9×9 and 19×19 events respectively in the TAAI Computer Go Tournament in Taiwan and scored the 6th position in the 3rd UEC Cup in Japan. In Erica, there are several standard MCTS implementations and enhancements, such as UCT [16], RAVE [10], and progressive bias [17]. MM [13] is used to compute the patterns in both progressive bias and the playout. Not only the light-weight features, but also the heavy-weight features are included in progressive bias, such as larger patterns and ladder.

3.2 Playout Features

The playouts of Erica are based on 3×3 stone patterns, augmented by the atari status of the four directly-connected points. These patterns are centered on the move to be played. By taking rotations, symmetries, and move legality into consideration, there is a total of 2,051 such patterns.

In addition to stone patterns, Erica uses 7 features related to the previous move:

1. Contiguous to the previous move. Active if the candidate move is among the 8 neighboring points of the previous move. Also active for all features 2–7.
2. Save the string in new atari by capturing. The candidate move that is able to save the string in new atari by capturing has this feature.
3. Same as Feature 2, which is also self-atari. If the candidate move has Feature 2 but is also a self-atari, then instead it has Feature 3 (Fig. 1).
4. Save the string in new atari by extension. The candidate move that is able to save the string in new atari by extension has this feature.
5. Same as Feature 4, which is also self-atari.
6. Solve a new ko by capturing. If there is a new ko, then the candidate move that is able to solve the ko by capturing any one of the neighboring strings has this feature.
7. 2-point semeai. If the previous move reduces the liberties of a string to only two, then the candidate move that is able to kill its neighboring string by giving atari has this feature. Fig. 1 gives an example. This feature deals with the most basic type of semeai.

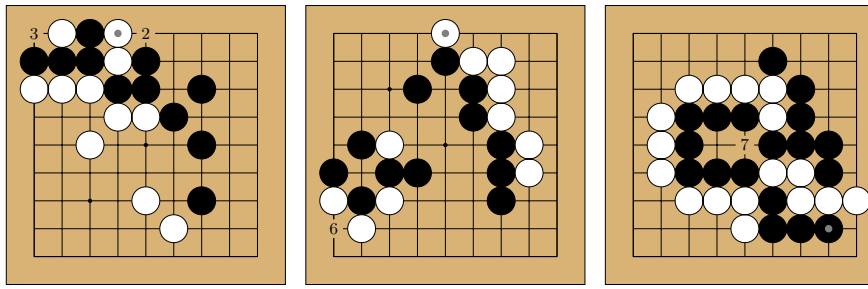


Fig. 1. Examples of Features 2, 3, 6, and 7. Previous move is marked with a dot.

3.3 Experiment Setting

The performance of MM and SB was measured by the winning rate of Erica against Fuego 0.4 with 3,000 playouts per move for both programs. For reference, performance of the uniform random playout policy and the MM policy are shown in Table 1.

Table 1. Result against Fuego 0.4, 1000 games, 9×9, 3k playouts/move

Playout Policy	Winning Rate
Uniform Random	$6.8\% \pm 0.8$
MM	$68.9\% \pm 1.4$
9x9 MM	$40.9\% \pm 1.6$

For fairness, both the training of MM and SB were performed with the same features described above. The training of MM was performed on 1,400,000 positions, chosen from 150,000 19×19 game records by strong players. This games were KGS games collected from the web site of Kombilo [18], combined with professional games collected from the web2go web site [19].

The production of the training data and the training process of SB were accomplished through Erica without any external program. The training positions were randomly selected from the games self-played by Erica with 3,000 playouts per move. Then Erica was directly used to evaluate these positions.

These 9x9 positions were also used to measure the performance of MM in the situation equivalent to that of SB. Same 5k positions, that were served as the training set of SB, were trained on MM to compute the patterns. The strength of these patterns was measured and shown in Table 1 as 9x9 MM.

3.4 Influence of Algorithm Meta-parameters

SB has a few meta-parameters that need tuning. For the gradient-descent part, it is necessary to choose M, N, and α. Two other parameters define how the training

set was built: number of positions, and number of playouts for each position evaluation. Table 2 summarizes experiment results with these parameters.

Since the algorithm is random, it would have been better to replicate each experiment more than once, in order to measure the effect of randomness. Because of limited computer resources, we preferred trying many parameter values rather than replicating experiments with the same parameters.

In the original algorithm, the simulations of outcome 0 are ignored when N simulations are performed to accumulate the gradient. The algorithm can be safely modified to use outcome -1/1 and replace z with (z-b), where b is the average reward, to make the 0/1 and -1/1 cases equivalent [20]. The results of the 1st and 4th columns in Table 2 show that the learning speed of outcome -1/1 is much faster than 0/1, so that the winning rate of outcome -1/1 of iteration 20 (69.2%) is even higher than that of outcome 0/1 of iteration 100 (63.9%).

A critical issue of the training set is the quality of its evaluation. Better evaluation produces better learning results is conspicuously demonstrated by that 100k evaluation (4th column in Table 2) performed much better in average than 10k evaluation(3rd column).

The SB algorithm was designed to reduce the mean squared error (MSE) of the whole training set by stochastic gradient-descent. As a result, the MSE should gradually decrease if the training is performed on the same training set ever and again. Running the SB algorithm through the whole training set once is defined as an *Iteration*. Although the MSE reduces gradually (Fig. 2), the playing strength will increase in the beginning and finally stop to increase after certain iterations, even start to decline.

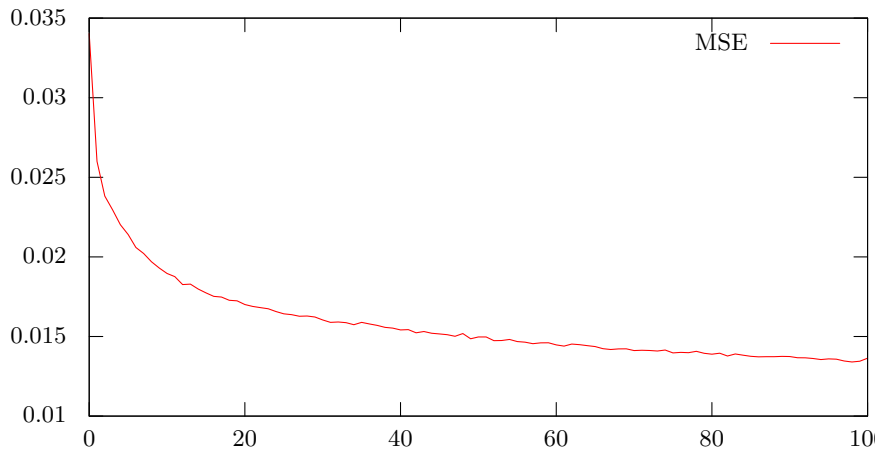


Fig. 2. Mean squared error as a function of iteration number. $M = N = 500$, $\alpha = 10$, training set has 5k positions evaluated with 100k playouts. Error was measured with 1000 playouts for every position of the training set.

Table 2. Experiment results. Winning rate was measured with 1000 games against FUEGO 0.4, with 3,000 playouts per move. 95% confidence is ±1.6 when the winning rate is close to 50%, and ±1.3 when it is close to 80%.

Positions	5k	5k	5k	5k	5k	10k
Playouts	100k	100k	10k	100k	100k	100k
M	500	100	500	500	100	500
N	500	100	500	500	100	500
α	10	10	10	10	1	10
Outcome	0/1	-1/1	-1/1	-1/1	-1/1	-1/1
20	51.5%	69.2%	65.7%	69.3%	51.8%	71.2%
40	57.6%	75.5%	68.5%	75.4%	57.2%	76.0%
60	58.1%	70.1%	70.8%	77.9%	57.2%	74.0%
80	61.3%	78.2%	72.2%	76.8%	63.7%	76.9%
100	63.9%	76.2%	74.0%	73.5%	65.4%	76.0%
200	60.8%	77.4%	71.6%	76.3%	70.1%	74.1%
300	61.9%	73.9%		75.0%	73.2%	
500					75.4%	
700					74.8%	
900					74.3%	
1100					76.2%	
Iteration	Winning Rate					

3.5 Comparison between MM and SB Feature Weights

For this comparisons, SB values that scored 77.9% against FUEGO 0.4 were used (60 iterations, fourth column of Table 2). Table 3 shows the γ-values of local features ($\gamma_i = e^{\theta_i}$ is a factor proportional to the probability that feature i is played). Table 4 shows some interesting 3×3 patterns (top 10, bottom 10, top 10 without atari, and most different 10 patterns).

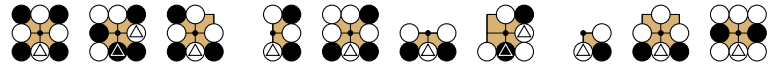
Table 3. Comparison of local features, between MM and SB

Feature	Description	MM γ	SB γ
1	Contiguous	11.12	7.43
2	Save new atari by capturing	32.37	151.04
3	2 + self-atari	0.24	0.53
4	Save new atari by extending	6.71	23.11
5	4 + self-atari	0.05	0.02
6	Capture after ko	0.65	6.37
7	2-point semeai	32.07	141.80

Local features (Table 3) show that SB plays tactical moves such as captures and extensions in a way that is much more deterministic than MM. A possible interpretation is that strong players may sometimes find subtle alternatives to those tactical moves, such as playing a move in sente elsewhere. But those

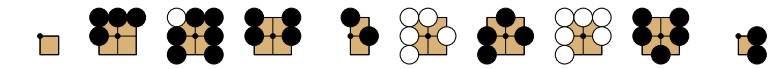
Table 4. 3×3 patterns. A triangle indicates a stone in atari. Black to move.

SB rank	1	2	3	4	5	6	7	8	9	10
MM rank	816	1029	8	1058	1055	403	441	431	960	555
SB γ	47.63	30.85	29.33	29.26	25.53	25.51	25.24	15.72	15.03	14.64
MM γ	1.55	0.95	16.98	0.88	0.89	3.34	3.10	3.15	1.10	2.50

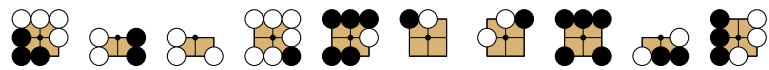


SB rank	1371	951	1870	1519	1941	148	546	3	1486	1180
MM rank	1	2	3	4	5	6	7	8	9	10
SB γ	0.92	1.01	0.43	0.85	0.24	2.35	1.13	29.33	0.86	0.98
MM γ	112.30	52.78	45.68	39.43	30.41	25.52	24.16	16.98	14.66	14.34

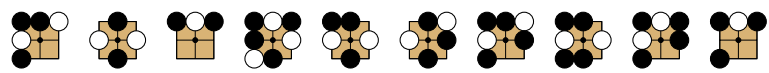
SB rank	2008	2007	2006	2005	2004	2003	2002	2001	2000	1999
MM rank	1982	1573	1734	2008	1762	1953	1907	1999	1971	1751
SB γ	0.02	0.02	0.03	0.03	0.04	0.04	0.04	0.04	0.05	0.06
MM γ	0.00	0.21	0.08	0.00	0.07	0.01	0.01	0.00	0.00	0.07



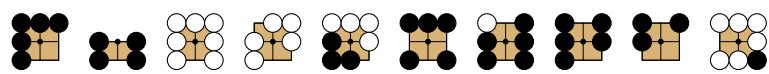
SB rank	2005	1896	1929	251	1910	1818	1874	1969	1915	2001
MM rank	2008	2007	2006	2005	2004	2003	2002	2001	2000	1999
SB γ	0.03	0.36	0.28	1.60	0.34	0.53	0.42	0.16	0.33	0.04
MM γ	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00



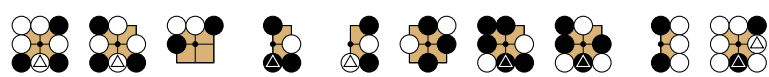
SB rank	11	13	14	15	16	19	25	27	28	32
MM rank	1847	1770	1775	1808	1509	420	900	1857	425	1482
SB γ	14.43	14.15	12.36	12.33	11.71	9.82	8.23	8.11	7.93	7.29
MM γ	0.03	0.07	0.06	0.04	0.28	3.25	1.27	0.03	3.21	0.29



SB rank	1317	702	815	497	1448	1759	397	1080	1466	537
MM rank	15	16	18	21	23	25	26	28	30	31
SB γ	0.94	1.06	1.03	1.16	0.88	0.62	1.27	0.99	0.87	1.14
MM γ	13.04	12.84	12.53	11.39	11.00	10.90	10.79	10.62	10.51	10.44



SB rank	34	90	40	119	11	27	61	145	72	15
MM rank	1975	1976	1904	1978	1847	1857	1889	1965	1868	1808
SB γ	6.85	3.38	5.90	2.72	14.43	8.11	4.73	2.36	4.15	12.33
MM γ	0.00	0.00	0.01	0.00	0.03	0.03	0.02	0.00	0.02	0.04



SB rank	1941	1870	1856	1898	1985	1759	1928	1872	1881	1737
MM rank	5	3	33	109	249	25	200	183	201	67
SB γ	0.24	0.43	0.45	0.35	0.10	0.62	0.28	0.42	0.41	0.65
MM γ	30.41	45.68	10.38	7.28	4.64	10.90	5.23	5.49	5.21	8.45

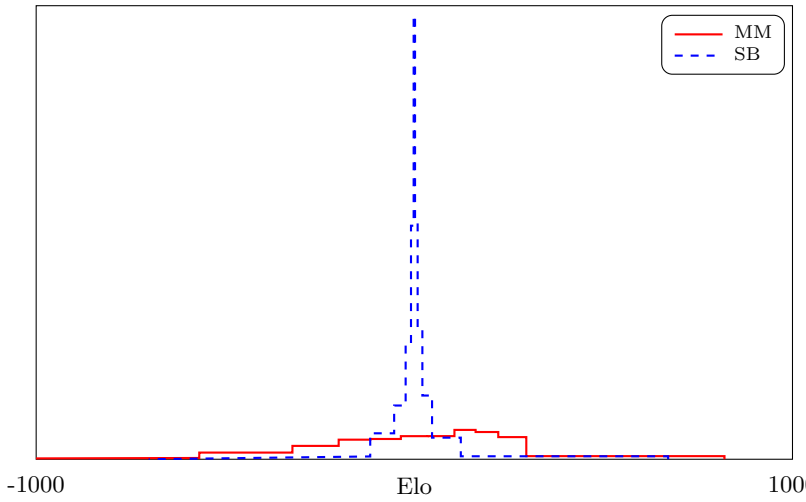


Fig. 3. 3×3 pattern density by Elo rating ($400\theta / \log(10)$).

considerations are far beyond what playouts can understand, so more deterministic captures and extensions may produce better Monte-Carlo evaluations.

Pattern weights obtained by SB are very different from those obtained by MM. Figure 3 shows that SB has a very high density of neutral patterns. Observing individual patterns on Table 4 shows that patterns are sometimes ranked in a very different order. Top patterns (first two lines) are all captures and extensions. Many of the top MM patterns are ko-fight patterns. Again, this is because those happen often in games by strong humans. Resolving ko fight is beyond the scope of this playout policy, so it is not likely that ko-fight patterns help the quality of playouts. Remarkably, all the best SB patterns, as well as all the worst SB patterns (line 3) are border patterns. That may be because the border is where most crucial life-and-death problems occur.

The bottom part of Table 4 shows the strangest differences between MM and SB. Lines 5 and 6 are top patterns without atari, and lines 7 and 8 are patterns with the highest difference in pattern rank. It is very difficult to find convincing interpretations for most of them. Maybe the first pattern of line 7 (with SB rank 34) allows to evaluate a dead 2×2 eye. After this move, White will probably reply by a nakade, thus evaluating this eye correctly. Patterns with SB ranks 40, 119, and 15 offer White a deserved eye. These are speculative interpretations, but they show the general idea: playing such ugly shapes may help playouts to evaluate life-and-death correctly.

3.6 Against GNU Go on 9×9 Board

The same patterns of SB in Section 3.5 were also used to play against GNU Go, which has been the most popular comparative object in computer Go for the past

years. For having more evident statistical observations, ERICA was set to play with 300 playouts per move to keep the winning rate as close to 50% as possible. The results presented in Table 5 indicate that SB still performs better, although its leading over MM is not as significant as in the previous experiments. The reason for this result is maybe that progressive bias still has dominant influence to guide the UCT search within 300 playouts. Also, it is a usual observation that improvement against GNU GO is often much less than improvement against other Monte-Carlo programs.

Table 5. Result against GNU GO 3.8 Level 10, 1000 games, 9×9, 300 playouts/move

Playout Policy	Winning Rate
Uniform Random	$22.1\% \pm 1.3$
MM	$59.3\% \pm 1.6$
SB	$62.6\% \pm 1.5$

3.7 Playing Strength on 19×19 Board

The comparison between MM and SB was also carried out on 19×19 board by playing against GNU GO 3.8 Level 0 with 1,000 playouts per move. Although the foregoing experiments confirms that SB surpasses MM on 9×9 board under almost every setting of M, N, and α, MM is still more effective on 19×19 board. In Table 5, the original SB scored only 33.2% with patterns which winning rate was 77.9% on 9×9 board. Even the γ-values of all local features of SB are replaced by that of MM (MM and SB Hybrid), the playing strength still does not improve at all (33.4%). Nonetheless, the winning rate of SB raises to 41.2% if the γ-value of Feature 1 is manually multiplied by 4.46 ($= (19 \times 19)/(9 \times 9)$), which was empirically obtained from the experimental results. This clearly points out that patterns computed by SB on 9×9 board are far from optimal on 19×19 board.

Table 6. Result against GNU GO 3.8 Level 0, 500 games, 19×19, 1000 playouts/move

Playout Policy	Winning Rate
Uniform Random	$8.2\% \pm 1.2$
SB	$33.2\% \pm 2.1$
MM and SB Hybrid	$33.4\% \pm 2.1$
SB(4.46)	$41.2\% \pm 2.2$
MM	$42.0\% \pm 2.2$

4 Conclusion

Experiments presented in this paper demonstrate the good performance of SB on the 9×9 board. This is an important result for practitioners of Monte-Carlo tree search, because previous results with this algorithm were limited to more artificial conditions.

Results also demonstrate that SB gives high weights to some patterns in very bad shape. This remains to be tested, but it indicates that SB pattern weights may not be appropriate for progressive bias. Also, learning opening patterns on the 19×19 board seems to be out of reach of SB, so MM is likely to remain the learning algorithm of choice for progressive bias.

The results of experiments also indicate that SB has the potential to perform even better. Many improvements seem possible.

First, steepest descent is an extremely inefficient algorithm for stochastic function optimization. More clever algorithms may provide convergence that is order of magnitude faster [21], without having to choose meta-parameters.

Second, it would be possible to improve the training set. Using many more positions would probably reduce risks of overfitting, and may produce better pattern weights. It may also be a good idea to try to improve the quality of evaluations by cross-checking values with a variety of different programs, or by incorporating positions evaluated by a human expert.

Acknowledgments

We thank David Silver for his comments and encouragements. We are also grateful to Lin Chung-Hsiung for kindly providing access to the game database of the web2go web site. Hardware was provided by project NSC98-2221-E-003-013 from National Science Council, R.O.C. This work was supported in part by the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886. This work was supported in part by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the "CPER 2007–2013". This publication only reflects the authors' views.

References

1. Abramson, B.: Expected-outcome: A general model of static evaluation. IEEE Transactions on Pattern Analysis and Machine Intelligence 12(2) (February 1990) 182–193
2. Brügmann, B.: Monte Carlo Go (1993) Unpublished technical report.
3. Bouzy, B., Helmstetter, B.: Monte Carlo Go developments. In van den Herik, H.J., Iida, H., Heinz, E.A., eds.: Proceedings of the 10th Advances in Computer Games Conference, Graz (2003)
4. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In van den Herik, H.J., Ciancarini, P., Donkers, H.J., eds.: Proceedings of the 5th International Conference on Computer and Games. Volume 4630 of Lecture Notes in Computer Science., Turin, Italy, Springer (June 2006) 72–83

5. Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with patterns in Monte-Carlo Go. Technical Report RR-6062, INRIA (2006)
6. Bouzy, B.: Associating domain-dependent knowledge and Monte-Carlo approaches within a Go program. Information Sciences, Heuristic Search and Computer Game Playing IV 175(4) (November 2005) 247–257
7. Chen, K.H., Zhang, P.: Monte-Carlo Go with knowledge-guided simulations. ICGA Journal 31(2) (June 2008) 67–76
8. Chaslot, G., Fiter, C., Hoock, J.B., Rimmel, A., Teytaud, O.: Adding expert knowledge and exploration in Monte-Carlo tree search. In: Proceedings of the Twelfth International Advances in Computer Games Conference, Pamplona, Spain (May 2009)
9. Bouzy, B., Chaslot, G.: Monte-Carlo Go reinforcement learning experiments. In Kendall, G., Louis, S., eds.: 2006 IEEE Symposium on Computational Intelligence and Games, Reno, USA (May 2006) 187–194
10. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Proceedings of the 24th International Conference on Machine Learning, Corvallis Oregon USA (2007) 273–280
11. Chaslot, G.M.J.B., Winands, M.H.M., Szita, I., van den Herik, H.J.: Cross-entropy for Monte-Carlo tree search. ICGA Journal 31(3) (September 2008) 145–156
12. Silver, D., Tesauro, G.: Monte-Carlo simulation balancing. In Bottou, L., Littman, M., eds.: Proceedings of the 26th International Conference on Machine Learning, Montreal, Canada, Omnipress (June 2009) 945–952
13. Coulom, R.: Computing Elo ratings of move patterns in the game of Go. ICGA Journal 30(4) (December 2007) 198–208
14. Enzenberger, M., Muller, M.: Fuego—an open-source framework for board games and Go engine based on Monte-Carlo tree search. Technical Report TR 09-08, University of Alberta, Edmonton, Alberta, Canada (2009)
15. Anderson, D.A.: Monte Carlo search in games. Technical report, Worcester Polytechnic Institute (2009)
16. Kocsis, L., Szepesvári, C.: Bandit-based Monte-Carlo planning. In Fürnkranz, J., Scheffer, T., Spiliopoulou, M., eds.: Proceedings of the 15th European Conference on Machine Learning, Berlin, Germany (2006)
17. Chaslot, G., Winands, M., Bouzy, B., Uiterwijk, J.W.H.M., van den Herik, H.J.: Progressive strategies for monte-carlo tree search. In Wang, P., ed.: Proceedings of the 10th Joint Conference on Information Sciences, Salt Lake City, USA (2007) 655–661
18. Goertz, U., Shubert, W.: Game records in SGF format. http://www.u-go.net/gamerecords/ (2007)
19. Chung-Hsiung, L.: web2go web site. http://www.web2go.idv.tw/gopro/ (2009)
20. Silver, D.: Message to the computer-go mailing list. http://www.mail-archive.com/computer-go@computer-go.org/msg11260.html (2009)
21. Schraudolph, N.N.: Local gain adaptation in stochastic gradient descent. In: Proceedings of the 9th International Conference on Artificial Neural Networks, London, IEEE (1999)