Move-Pruning Techniques for Monte-Carlo Go

Bruno Bouzy

UFR de mathematiques et d'informatique,
Université René Descartes, Paris, France
`bouzy@math-info.univ-paris5.fr`

Abstract. Progressive Pruning (PP) is employed in the Monte-Carlo Go-playing program INDIGO. For each candidate move, PP launches random games starting with this move. The goal of PP is: (1) to gather statistics on moves, and (2) to prune moves statistically inferior to the best one [7]. This papers yields two new pruning techniques: Miai Pruning (MP) and Set Pruning (SP). In MP the second move of the random games is selected at random among the set of candidate moves. SP consists in gathering statistics about two sets of moves, GOOD and BAD, and it prunes the latter when statistically inferior to the former. Both enhancements clearly speed up the process of selecting a move on 9×9 boards, and MP improves slightly the playing level. Scaling up MP to 19×19 boards results in a 30% speed-up enhancement and in a four-point improvement on average.

1 Introduction

Computer Go remains a difficult task for computer science [14,12] mainly for two reasons. First, the branching factor of the game tree and the game length prohibit global tree search. Second, evaluating non-terminal Go positions is hard [13]. Meanwhile, computer Go has been used as an appropriate testbed for AI methods [6] during the last decade. I started twelve years ago, with the development of the Go-playing program, INDIGO [5]. Since 2002, INDIGO includes a Monte-Carlo approach that enriches the knowledge-based approach developed previously. Our Monte-Carlo approach is inspired by usual experiments [7] reproducing the original approach of Monte-Carlo Go [8]. Subsequently, these experiments introduced different enhancements to the basic Monte-Carlo algorithm. Currently, Progressive Pruning (PP) is the umbrella enhancement to be used in INDIGO. In 2003, we combined our Monte-Carlo Go approach with a knowledge-based approach [3] and with a global tree-search approach [4]. The result was successful because INDIGO won the bronze medal at the 2004 Olympiad on 19×19 Go [10]. Yet this successful combination is not the topic of the current paper. The innovative question that motivates the work presented here is: how can we improve Progressive Pruning (a) in the game of Go, and (b) in general? To this purpose we assess two pruning techniques intended to improve PP: Miai Pruning (MP) and Set Pruning (SP). We will introduce these two new pruning techniques, and provide an experimental assessment.

© Springer-Verlag Berlin Heidelberg 2006

Section 2 discusses related work that deals with Monte-Carlo games, and it recalls the underlying idea of PP. Section 3 defines the two pruning techniques, MP and SP. Then, Section 4 yields the results of the experiments assessing these two techniques in isolation, and in combination. Some remarks are discussed in Sect. 5. Section 6 provides a conclusion and some prospects.

2 Related Work and Motivations

Below we discuss three topics, viz. Monte Carlo in computer games (2.1), progressive pruning (2.2), and motivations (2.3).

2.1 Monte Carlo in Computer Games

Monte-Carlo methods were designed in order to simulate physical models. Because they used random number generation such as games in the casino, the name Monte Carlo was adopted. Then Monte-Carlo methods were embraced by computer games, and so, to some extent, a loop has been closed. In games such as Poker and Scrabble, hidden information is sampled with the help of random distributions that are plausible according to past actions performed in the game. In such games, random generation can also be used to perform random simulations of games, which is done by POKI at Poker [2] and by MAVEN at Scrabble [15]. In games containing randomness in their rules, such as Backgammon, random simulations are used quite naturally [17]. In complete information games not containing any chance, such as Go, Chess, and Othello, the idea of simulating games at random is less natural. Nevertheless, this is not the first time that Monte-Carlo methods have been tried in complete information games.

In 1990, Abramson [1] gave a seminal description of evaluating a position of a two-person complete information game with statistics. He proposed the *expected-outcome model*, in which the evaluation of a game-tree node is the expected value of the game's outcome given random play from that node on. The author showed that the expected outcome is a powerful heuristic. He concluded that the expected-outcome model of two-player games is "precise, accurate, easily estimable, efficiently calculable, and domain-independent". In 1990, he tried the expected-outcome model on the game of 6×6 Othello.

Brügmann [8] was the first to develop a Go program based on random games. The architecture of the program, GOBBLE, was remarkably simple. In order to choose a move in a given position, GOBBLE played a large number of random games from this position to the end, and scored them. Then, he evaluated a move by computing the average of the scores of the random games in which it had been played.

We believe that Abramson's approach (or Brügmann's) are quite appropriate for the game of Go because they enable the program to reach terminal positions that are easy to evaluate and particularly representative of the current position. By computing a mean on terminal positions reached at random, the program obtains a first-rate evaluation of the current position. We admit that computing a Monte-Carlo evaluation costs much more time than computing a conceptual evaluation using domain-dependent knowledge, but we believe that the cost is worthwhile. This is why we follow the Monte-Carlo approach in Indigo. The next subsection recalls how PP is used in Indigo.

2.2 Progressive Pruning

The aim of PP is to be able to choose the best move. The current description is based on Bouzy and Helmstetter [7]. As contained in the basic idea of Abramson, each move has a mean value m, a standard deviation σ, a left expected outcome m_l and a right expected outcome m_r. For a move, $m_l = m - \sigma r_d$ and $m_r = m + \sigma r_d$. r_d is a ratio fixed by practical experiments. Currently, $1.5 \leq r_d \leq 2.0$ is for us a good tradeoff between playing level and time. A move M_1 is said to be statistically inferior to another move M_2 if $M_1.m_r < M_2.m_l$. Two moves M_1 and M_2 are statistically equal when $M_1.\sigma{<}\sigma_e$ and $M_2.\sigma{<}\sigma_e$ and no move is statistically inferior to the other. σ_e is called the standard deviation for equality, and its value is determined by experiments.

In PP, after a minimal number N_{min} of random games (currently 50 per move), a move is pruned as soon as it is statistically inferior to another move. N_{rg} is the current number of random games performed; the standard deviation of the mean value computed after N_{rg} random games is $\sigma/\sqrt{N_{\mathrm{rg}}}$. Therefore, moves are pruned as N_{rg} increases, and the number of candidate moves decreases while the process is running. The process stops if one of three conditions is fulfilled: (1) when there is only one move left, (2) when the moves left are statistically equal, or (3) when a maximal threshold of iterations N_{total} is reached. In all cases, the move with the highest expected outcome is chosen. This progressive pruning algorithm is similar to the one described in [2].

Due to the increasing precision of mean evaluations while the process is running, the mean value of the current best move is decreasing. Consequently, a move can be statistically inferior to the best one at a given time and not later. Thus, the pruning process can be either hard (a pruned move cannot be a candidate later on) or soft (a move pruned at a given time can be a candidate later on). Of course, soft PP is more precise than hard PP. Nevertheless, in the experiments shown here, we use hard PP.

2.3 Motivations

Using PP or any move-pruning scheme is debatable. For example, Sheppard [16] uses a clever scheme to drive the choice of which simulation to perform on which move, and this scheme does not prune any move. In contrast, we assume that PP is used, and attempt to improve it. We do not debate on the use or non-use of move pruning.

The background of this work is the architecture of Indigo. It is made up of a pre-selection module and a Monte-Carlo module. N_{select} is the number of moves; it is the output of the pre-selection module, and the input of the Monte-Carlo module. As long as PP is running, the number of possible moves is decreasing, namely from N_{select} down to 1 at which point the process stops. Two remarks can be made. First, PP spends most of its time when two possible moves are left. Frequently, the evaluations of the two moves left are almost equal, and a great deal of iterations are necessary to separate them. Thus, the first goal is to reduce the time spent by PP when two moves are left. Second, INDIGO's playing level highly depends on N_{select}. INDIGO's playing level roughly increases with N_{select}. (Admittedly, this is not completely correct. Actually, INDIGO's playing level reaches an optimal value with $N_{\text{select}} = 8$, 16, or 32, depending on the size of the board 9×9, 13×13, or 19×19, and then, it decreases.) However, the optimal value of N_{select} can be quite high and therefore the second goal is to reduce the time spent by PP to eliminate moves at the beginning of the process. To sum up, we need (1) a pruning technique that lowers the time spent when two or a few moves are left at the end of the process, and (2) another technique to eliminate quickly most of the moves at the beginning of the process; both techniques should operate under the same statistical confidence when pruning. With this aim, we designed two pruning techniques.

3 Two Pruning Techniques

This section describes the two pruning techniques. Miai Pruning (MP) is the technique that speed-up the end of PP process when a few moves are left (in particular two), and Set Pruning (SP) is the technique that speeds up the beginning of the PP process when many moves are involved.

3.1 Miai Pruning

Without any loss of generality, we assume that two moves are left, A and B, and that Black is to move. PP aims at finding the move with the best mean. Therefore, PP launches many games (1) starting with Black A and with the following moves randomly chosen, and (2) starting with Black B, and with the following moves also randomly chosen. Unfortunately, in the half of the games starting with move A, move B is also played by Black. In addition, in the half of the games starting with move B, move A is also played by Black. Thus, in the half of the random games played out to separate the moves A and B, A and B are played by the same player. When the order of the moves of a sequence is not important to reach a position (which is not rare in Go) the half of the random games does not help much to discriminate A and B. So, the idea of MP is (1) to launch games starting with Black A and White B, and (2) to launch games starting with Black B and White A to separate A and B. Now we should make MP working when the number of possible moves is arbitrary small (say three or more). MP works as follows. For each possible move A, B being another possible move different from A chosen at random, MP launches games starting with Black A in the first move, and White B in the second move. The term "miai" emerged because it is used by human Go players for the same concept: "miai" means equivalent. When two moves are miai, then it happens that if a player plays one of them, the other player plays the other one. Finally, we remark that MP is designed to separate moves which are not miai by imitating the actual way of human playing. When played after move A, B can be an illegal move. In such case, MP is simply not used.

3.2 Set Pruning

Let us assume that N_{possible} moves are left. SP applies two stages. First, to be cautious and avoid rather bad pruning due to bad chance, N_{min} is devised by PP to forbid any pruning before N_{min} random games per move are arrived at. Second, at the beginning of the PP process, the σ of a move is high for each move. Thus, no move has a good chance to be statistically inferior to the best one with a given statistical confidence. The idea underlying SP is to associate a mean value and a σ not only to the possible moves but also to all the possible *sets* of moves of size $N_{\text{possible}}/2$. N_{rgpm} is the number of random games performed per candidate move, then the mean value of the random games performed given that the first move belongs to a given set of size $N_{\text{possible}}/2$ is known with a σ that is in $1/\sqrt{N_{\text{rgpm}} \times N_{\text{possible}}/2}$. Consequently, the width of the confidence interval around the mean value associated to sets of moves is $\sqrt{N_{\text{possible}}/2}$ times smaller than the width of the confidence interval around the mean value associated to moves. Thus, it is possible to prune a set of moves at once, with a statistical confidence which equals the statistical confidence at which PP prunes moves one by one. This way, the number of possible moves is divided by two, each time a set of moves is pruned. The idea is quite attractive because in practice we do not need to consider all of the possible sets of size $N_{\text{possible}}/2$ (a big effect) but only two sets. Because the moves are ranked from the best move down to the worst move, they can be grouped into two sets, the set of the $N_{\text{possible}}/2$ best moves, called GOOD, and the set of the $N_{\text{possible}}/2$ worst moves, called BAD. The mean associated to GOOD is the highest mean associated to any other set of size $N_{\text{possible}}/2$, and the mean associated to BAD is the lowest mean associated to any other set of size $N_{\text{possible}}/2$. Therefore, the first set to be pruned is the pruning of BAD. In practice, SP works as follows. In addition to the mean and σ computed by PP for each move, SP builds the two sets GOOD and BAD, and computes their means and their σ. When BAD is found to be statistically inferior to GOOD, it is pruned with a statistical confidence identical to the statistical confidence at which moves are pruned by PP.

4 Experiments

Starting from PP we determine empirically the size of N_{select} (4.1). Thereafter, this section evaluates the relative merits of MP (4.2) and SP (4.3), of their direct combination (M+SP) (4.4), their strong combination (M+gbSP) (4.5), and of a special combination of the two (M+gbP) (4.6), all of this regarding time and playing level. We end up this section with an all-against-all tournament (4.7) gathering the best programs of the experiments.

Since we explore move-pruning abilities of a Monte-Carlo Go program, we wish first to observe the move-pruning effect isolated from deep-tree search effects. Thus, we have performed experiments with depth-one search only (4.8). Furthermore, because (1) we need a large amount of game results to obtain a sufficient statistical significance, and (2) 19×19 games are too long, we have used 9×9 boards. When a pruning technique has been demonstrated as performing well on 9×9 boards at depth-one, it is assessed in a second stage either at depth-n on 9×9 boards (4.9) or at depth-one on 19×19 boards (4.10).

For each technique, we set up experiments to assess its effect on the time level and on the playing level. An experiment consists of a match of 200 games between the program to be assessed and the experiment reference program, each program playing 100 games with Black. The result of an experiment is generally a set of relative scores assuming that the assessed program is the max player. Given that the standard deviation of 9×9 games played by our programs is roughly 15 points, 200 games enable our experiments to lower σ down to 1 point and to obtain a 95% confidence interval of which the radius equals 2σ, i.e., 2 points. We have used 2.8 GHz computers. Furthermore, all programs in these experiments do not use any conservative or aggressive style depending on who is ahead in a game, they only try to maximize their own score. The score of a game is more significant than the winning percentage. N_{select} is a power of 2 between 2 and 64. r_d is set to 2.0.

Prune is the name of the program to assess. In its basic version, Prune uses PP only. The notation to name a program to be assessed is simple: for example, Prune($MP = true$) is the program that uses additionally the MP technique, and so on for Prune($SP = true$) or Prune($N_{\text{select}} = N$).

4.1 N_{select} Versus $N_{\text{select}}/2$

We start attempting to obtain the best playing level and the minimal response time, knowing that the effect of increasing the value of N_{select} is worth remembering. Table 1 shows the effects of simply doubling N_{select}. Each number corresponds to a confrontation between Prune(N_{select}) and Prune($N_{\text{select}}/2$).

Table 1 shows an increase of the playing level in N_{select}. However, the returns diminish as N_{select} increases. Although being not statistically significant, Table 1 shows that Prune($N_{\text{select}} = 32$) is slightly superior to Prune($N_{\text{select}} = 16$), and even that Prune($N_{\text{select}} = 64$) is almost equal to Prune($N_{\text{select}} = 32$). Regarding the relative time between the two programs, for low values of N_{select},

Table 1. Result of doubling N_{select}

	N_{select}				
	4 vs. 2	8 vs. 4	16 vs. 8	32 vs. 16	64 vs. 32
Mean score	+8.0	+6.0	+4.3	+0.4	−0.2
Winning percentage	72	65	63	54	52
Mean relative time	2.0	1.75	1.58	1.30	1.12

PRUNE(N_{select}) is about twice slower than PRUNE($N_{\text{select}}/2$), but for high values of N_{select}, PRUNE(N_{select}) is almost as fast than PRUNE($N_{\text{select}}/2$).

The results of this introductory experiment show that increasing N_{select} from 2 up to 16 is worthwhile considering in terms of playing level on a 9×9 board. The explanation is straightforward for PRUNE. We assume that the pre-selection module being based on hand-crafted domain-dependent knowledge still contains errors, and that the Monte-Carlo module is quite adequate at selecting the right move. With this assumption, the larger N_{select}, the larger the probability of selecting a good move input of Monte Carlo. However, one thing cannot be omitted: the pre-selection module gives a penalty to tactically bad moves but it does not eliminate them. Thus, when N_{select} is sufficiently large, the tactically bad moves are also input of Monte Carlo. Monte Carlo is bad at recognizing tactically bad moves, which explains that PRUNE($N_{\text{select}} = 64$) is worse than PRUNE($N_{\text{select}} = 32$). Meanwhile, the time for obtaining the overall playing-level jump is multiplied by a factor 6.

4.2 Miai Pruning Versus Progressive Pruning

This subsection first compares MP with PP. Then it shows the effects of doubling N_{select} while using MP.

Table 2. Result of MP vs. PP

	N_{select}					
	2	4	8	16	32	64
Mean score	+2.9	+1.3	+2.1	−1.3	−0.6	−0.7
Winning percentage	53	50	51	49	50	47
Mean relative speed	1.50	1.45	1.37	1.33	1.31	1.27

Table 2 shows that MP is worth considering for low values of N_{select}. Regarding the motivations of this paper (see 2.3), the result of PRUNE($MP = true$, $N_{\text{select}} = 2$) is crucial to comment upon. First, PRUNE($MP = true$, $N_{\text{select}} = 2$) has a non-negative result against PRUNE($MP = false$, $N_{\text{select}} = 2$): +3 points and 53% wins. The mean score is statistically significant because 3 points is superior to the radius of the confidence interval which equals 2 points. Second, the speed is enhanced significantly, multiplied by 1.5. Thus, the first column of Table 2 experimentally proves the relevance of MP, and it adheres the goals set in Section 3. The non-negative mean score and the speed enhancement of two next columns ($N_{\text{select}} = 4, 8$) of Table 2 confirm the effectiveness of MP. The right part of the table then shows slightly negative mean scores. MP appears to be less adapted to situations in which many moves are candidate than to situations with a few candidate moves.

As already shown in Table 1, Table 3 shows that with MP the playing level also increases in N_{select}. Between $N_{\text{select}} = 2$ and $N_{\text{select}} = 4$, the return improves faster with MP than without MP. However, for high values of N_{select} the return diminishes faster with MP than without MP. Moreover, PRUNE($N_{\text{select}} = 64$)

Table 3. Result of doubling N_{select} while using MP

	N_{select}				
	4 vs. 2	8 vs. 4	16 vs. 8	32 vs. 16	64 vs. 32
Mean score	+15.4	+3.7	+1.0	−1.2	−4.3
Winning percentage	67	58	52	49	43
Mean relative speed	0.47	0.52	0.61	0.60	0.62

looks like inferior to PRUNE($N_{\text{select}} = 32$) with some statistical significance; this is remarkable. It confirms the experimental fact that MP is less adapted to situations in which many moves are candidate than to situations with a few candidate moves. Our current explanation is the following. Without MP, the second move of a random game is selected pseudo-randomly with domain-dependent knowledge: one-liberty string or 3×3 pattern urgencies [3]. With MP, the second move is selected at random with uniform probability among the set of candidate moves. When N_{select} is small, the candidate moves are all approximately good, thus the second move selected by MP has a good chance to be better than the move generated pseudo-randomly with domain-dependent knowledge: one-liberty string or 3×3 pattern urgencies. When N_{select} is high, the candidate moves are approximately average, thus the second move selected by MP has a good chance to be worse than the move selected pseudo-randomly with domain-dependent knowledge. In conclusion, selecting the second move of the random game is a question of superiority between MP and the pseudo-random generator based on domain dependent knowledge. To be effective, MP must be better than the current pseudo-random move generator. If random games based on uniform probability were used, then MP would have no difficulty to be superior. In the background of the pseudo-random generator using domain-dependent knowledge, employing MP when N_{select} is high is consequently a bad idea. Subsection 4.8 will show a remedy to this problem.

4.3 Set Pruning Versus Progressive Pruning

This subsection compares SP with PP. Table 4 shows the results.

For low values of N_{select}, PRUNE($SP = true$) plays at the same level as PRUNE($SP = false$) and the increase in speed is not high. For high values of N_{select}, the relative speed of the two programs is significantly superior to 1, which was expected, because the SP technique is designed for high

Table 4. Result of SP vs. PP

	N_{select}					
	2	4	8	16	32	64
Mean score	+0.1	+0.5	+0.2	−1.3	−2.4	−3.5
Winning percentage	50	49	52	48	44	43
Mean relative speed	1.00	1.05	1.08	1.12	1.14	1.17

values. However, while the relative speed increases, the playing level decreases significantly, PRUNE($SP = true$, $N_{\text{select}} = 64$) being significantly inferior to PRUNE($SP = false$, $N_{\text{select}} = 64$). This result confirms the fact that using SP is debatable.

4.4 M+SP Versus Progressive Pruning

This subsection directly combines MP and SP, and compares this combination with PP. The direct combination means that SP is used in addition to MP, and that no other enhancement is used, as will be shown in the next subsection.

Table 5. Result of M+SP vs. PP

	N_{select}					
	2	4	8	16	32	64
Mean score	−0.5	+0.4	−3.9	−8.5	−11.4	−12.5
Winning percentage	49	51	42	33	28	29
Mean relative speed	1.5	1.5	1.7	2.2	2.6	2.8

Table 5 shows the results of MP+SP versus PP. The results are bad. Losing by eleven or twelve points on average on 9×9 boards is huge in Go standards. This result is disappointing. While SP did not give good results, it was risky to combine them so directly. The next experiment aims at combining MP in a sophisticated way.

4.5 M+gbSP Versus Progressive Pruning

Since the direct combination M+SP did not work well, we tried a more sophisticated combination of MP and SP, called M+gbSP. In addition to MP and SP, the two sets, GOOD and BAD updated by SP, were used by M+gbSP to launch the random games: for each possible move A, MP+gbS launches games starting with Black A and White B, B being picked up at random among BAD if A is in GOOD, and picked up in GOOD otherwise. Thus, when launching the random games, the idea underlying M+gbSP is to apply the miai principle on the two sets GOOD and BAD instead of applying them on moves.

Table 6 shows the results of M+gbSP versus PP. The results are still bad. Losing by ten or fifteen points on average on 9×9 boards is huge in Go standards. Our current explanation is that the strong combination reinforces the pruning

Table 6. Result of M+gbSP vs. PP

	N_{select}					
	2	4	8	16	32	64
Mean score	+0.3	+1.7	−8.3	−10.2	−11.3	−15.4
Winning percentage	52	52	35	33	32	26
Mean relative speed	1.5	1.6	1.9	2.8	3.5	3.8

strategy. Here, games are launched in order to enhance move pruning instead of neutrally finding the mean value of moves. When the first part of a run badly ranks a move - a “good” move M is put into BAD - the next part of the run tries to reinforce the current finding: when it launches games starting by M, the second move of the game (played by the opponent) is picked up from GOOD, thus the mean value of M is penalized. Symmetrically, when a “bad” move M is put into GOOD in the beginning of the run, the next part of the run launches games starting by M, then the second move of the game (played by the opponent) is picked up from BAD, thus the mean value of M is optimistic, and M remains in GOOD.

4.6 M+gbP Versus Progressive Pruning

Since M+gbSP does not work, probably due to its complexity, this subsection tries a simplification. It combines MP with the sole use of the two sets, GOOD and BAD, but not with SP. We call this combination M+gbP. GOOD and BAD are used in the same way as they are used in the strong combination M+SP: for each possible move A, M+gbP launches games starting with Black A and White B, B being picked up at random among BAD if A is in GOOD, and picked up in GOOD otherwise.

Table 7 shows the results obtained by M+gbP versus PP. The relative speed is higher than it was in Table 2. For $N_{\text{select}} = 2, 4, 8, 16$, the playing level of M+gbP seems identical to the playing level of MP. However, for $N_{\text{select}} = 32$ or 64, the results are still bad. The arguments highlighted by the previous subsection could still explain them.

4.7 All-Against-All Tournament

In the previous subsections, we have made relative assessments of the pruning techniques against PP with constant N_{select}, and relative assessments of doubling N_{select} with a fixed pruning technique (either MP or PP). In this subsection, we look for the best programs, in term of time and playing level. Thus, based on the previous experiments' results, we have built two tables approximating the values of the programs against PRUNE(PP, $N_{\text{select}} = 2$). Table 8 yields the average time used by the programs PRUNE to play one game, and Table 9 the relative playing level of PRUNE estimated with the previous results.

Table 7. Results of M+gbP vs. PP

	N_{select}					
	2	4	8	16	32	64
Mean score	+0.7	+2.1	+0.0	−1.1	−3.5	−8.1
Winning percentage	51	55	50	44	43	35
Mean relative speed	1.5	1.5	1.5	1.5	1.6	1.7

Table 8. Average time (in minutes) spent by PRUNE(N_{select}, P) to play out one game

N_{select}	PP	MP	SP	M+SP	M+gbSP	M+gbP
2	1.0	0.7	1.0	0.7	0.7	0.7
4	2.0	1.4	2.0	1.3	1.2	1.3
8	3.5	2.7	3.5	2.1	1.8	2.5
16	5.5	4.3	5.5	2.5	1.9	3.7
32	7.0	5.2	6.2	2.7	2.0	4.5
64	7.5	6.0	6.5	2.7	2.0	4.5

Table 9. Relative playing level of PRUNE(N_{select}, P) estimated by the previous subsections

N_{select}	PP	MP	SP	M+SP	M+gbSP	M+gbP
2	0	+3	0	0	0	0
4	+8	+11	+8	+8	+8	+10
8	+14	+15	+13	+10	+8	+14
16	+18	+18	+17	+10	+8	+17
32	+18	+16	+16	+9	+7	+15
64	+18	+14	+15	+8	+4	+10

Table 9 clearly shows that PRUNE(M+SP OR M+GBSP) and PRUNE(N_{select} = 2 OR 4), are not worth considering. Table 8 shows that PRUNE(N_{select} = 64) is slower than PRUNE($N_{\text{select}} = 8, 16, 32$). Meanwhile, it is slightly weaker, thus eliminated. Thus, we kept nine programs PRUNE(PP, MP, SP, N_{select} = $8, 16, 32$) for an all-against-all tournament.

Table 10 gives the final rankings with the average score per game. The σ of each average result is about 1.2. The radius of the 95% confidence interval is 2.4. Consequently, clear conclusions can hardly be drawn from this tournament. Concerning the playing level, all the players are on a par. The best value of N_{select} seems to be 32, unfortunately lowering the relevance of the pruning techniques. SP seems to be a better enhancement than MP regarding both playing level and time. However this tournament is not fair for MP because of the high values of N_{select}. This leads to the perspective to apply MP only when the number of candidate moves is inferior to a threshold. Conclusions on the playing level are hard to draw; the time considerations may break the tie. Table 8 shows that MP8 is the fastest program among the players of the all-against-all tournament, enhancing the interest of MP.

Table 10. Final ranking of the all-against-all tournament

	Rank								
	1	2	3	4	5	6	7	8	9
Prune	P32	S32	S16	P8	S8	M8	M16	P16	M32
Mean score	+3.6	+1.7	+1.2	0.0	−0.3	−1.2	−1.5	−1.7	−2.0

4.8 Weak Miai Pruning (WMP)

Since the all-against-all tournament has shed the light on a weakness of MP when $N_{\text{select}} = 16$ or 32, this subsection considers a weak version of MP which consists in using the MP rule only when the number of candidate moves is strictly inferior to a threshold T.

Conversely to straightforward MP which is effective for low values of N_{select} only, Table 11 shows that WMP (with $T = 5$) is worth considering for any values of N_{select}. First, PRUNE($WMP = \textit{true}$, $N_{\text{select}} = 2, 4$) keeps the positive result shown by Table 2. Second, PRUNE($WMP = \textit{true}$, $N_{\text{select}} = 8, 16$) has quite a positive result against PRUNE($WMP = false$, $N_{\text{select}} = 8, 16$): +3 points and 60% wins. As mentioned in Subsection 4.2, the mean score is statistically significant. Third, the positive mean score of two next columns ($N_{\text{select}} = 32, 64$) of Table 11 confirms the effectiveness of WMP, and removes the negative mean scores of Table 2. Finally, the speed is enhanced significantly, multiplied by 1.5 for $N_{\text{select}} = 2$, and not lowered for high values of N_{select}. To sum up, WMP is experimentally demonstrated to be superior to PP on 9×9 boards for any value of N_{select}, both in time and in playing level. This experiment is a success.

4.9 Integrating WMP with Global Tree Search on 9 × 9 Boards

The result obtained by WMP within the basic MC framework on 9×9 boards, namely depth-one search, suggests using WMP in the framework combining MC and Tree Search [4]. Currently, INDIGO uses a depth-3 global tree search on 9×9 boards. Consequently, we set up an experiment assessing PRUNE($WMP = \textit{true}$, $N_{\text{select}} = 8$, $\textit{Depth} = 3$) against PRUNE($WMP = false$, $N_{\text{select}} = 8$, $\textit{Depth} = 3$) on 9×9 boards. It turns out that, although playing 5% faster, PRUNE ($WMP = \textit{true}$, $N_{\text{select}} = 8$, $\textit{Depth} = 3$) is 1.7 point inferior to PRUNE ($WMP = false$, $N_{\text{select}} = 8$, $\textit{Depth} = 3$) and wins 45% of games only. Thus, integrating WMP with global tree search on 9×9 boards is not a success.

We have the following explanation. First, the MP principle can be discussed in front of depth-2 search. Actually, since MP launches games beginning by two given moves, the mean values computed correspond to depth-2 nodes. However, the background in which WMP is used cannot be forgotten. WMP is used only when previous random games have pruned moves, and moreover, after move A, MP develops move B only, and not all the children of move A. Thus depth-2 search dominates MP. MP is a trick used because of time constraints, when

Table 11. Result of Weak MP (WMP) vs. PP for $T = 5$

	N_{select}					
	2	4	8	16	32	64
Mean score	+1.5	+1.0	+3.1	+4.5	+2.7	+3.0
Winning percentage	51	50	60	61	53	55
Mean relative speed	1.50	1.48	1.25	1.12	1.05	1.02

depth-2 search cannot be used. Second, [4] and MP both expand the child nodes of a parent node when the number of children decreases and reaches a threshold ($W-$ in [4] and T in MP). Therefore, the two techniques do not live well all together. Finally, [4] being a kind of iterative deepening algorithm, we have also tried to use WMP at the maximal depth only, and not at intermediate depths, but this attempt was not satisfactory.

4.10 Scaling WMP Up to 19 × 19 Boards

As explained in the introduction of Section 4, to speed up the validation process of our ideas, we have first performed our experiments on 9×9 boards. After such experiments, SP does not fulfil our initial expectation, but MP, and in particular WMP, is still worth considering. Therefore, in a second stage, WMP deserves a 19×19 assessment. To make the programs playing in adequate time on 19×19 boards, Monte-Carlo parameters are set differently. For example, r_d is set to 1.5 and not to 2.0. Moreover, when scaling up to 19×19 boards from 9×9 boards, the maximal number of random games is reduced in a 40% proportion. Obviously on 19×19 boards, the time constraints bring about a depth-one search. The value of the parameters being different, it was not certain that WMP behaves on 19×19 boards in the same way as it does on 9×9 boards. After 400 games, $\textsc{Prune}(WMP = true,\ N_{\text{select}} = 8)$ turns out to be +4 point superior to $\textsc{Prune}(WMP = false,\ N_{\text{select}} = 8)$ winning 51.6% of the games. The 95% confidence interval is [-3.4, +11.8] and the 68% confidence interval is [+0.4, +8.0]. Hopefully, this result shows that WMP scales well on 19×19 with a depth-one search. More interesting is the fact that $\textsc{Prune}(WMP = true)$ used 36 minutes on average to complete one 19×19 game. Meanwhile, $\textsc{Prune}(WMP = false)$ used 46 minutes on average. Thus $\textsc{Prune}(WMP = true)$ is 1.27 faster than $\textsc{Prune}(WMP = false)$. This positive result on 19×19 boards is explained by the fact that a depth-one search is mandatory. In this context, WMP appears to be a trick when depth-two search remains forbidden.

5 Discussion

To explain why MP works, we introduce the notion of the *incentive of a move* as the difference between the MC evaluation of the position reached by this move and the MC evaluation of the current position. Let a be the incentive of Black playing move A, and b the incentive of Black playing move B. PP launches games to assess a and b. It stops when the difference $a - b$ is statistically different from zero. Let us assume that the incentive of White playing move A (resp. B[1]) is $-a$ (resp. $-b$), which is plausible in most cases. When A and B are not dependent, the games launched by MP starting with Black A (resp. B) and White B (resp. A) contribute to assess $a - b$ (resp. $b - a$). MP stops when the difference between the two means, i.e., $2 \times (a - b)$, is statistically different from zero. Thus, when A

[1] The notation "(resp. B)" is shorthand for "(or B respectively)".

and B are not miai, the average number of games launched by MP to separate A and B is smaller than the average number of games launched by PP.

SP was designed to speed up the beginning of the PP process. The experiments show that SP is a failure. One could say that SP has no more theoretical foundation than PP itself. One could expect that pruning the set of moves would exhibit the same move quality versus CPU time tradeoffs obtained by pruning the individual moves. The experimental results are not inconsistent with this hypothesis.

Scared by the long lasting experiments on 19×19 boards, we have chosen to spend the CPU time to perform experiments on 9×9 games first. Considering that Indigo uses depth-3 on 9×9 boards, and that MP does not work well with depth-n search, all this work does not result in profitable behavior of Indigo on the 9×9 board. Hopefully, scaling up to 19×19 boards after assessing the quality of MP was a good surprise. MP works well with depth-one search and Indigo uses depth-one on 19×19 boards. Considering Indigo's development, the speed-up and move quality improvements are finally effective on 19×19 boards, and not on 9×9 boards. In Subsection 4.8, WMP experiments were presented. The difference between WMP and MP lies in the use of a threshold enabling the program to use the MP heuristic only when the number of candidate moves is lower than the threshold.

A more important reason of the success of WMP over MP as been revealed since the time of the experiments. In fact, as specified in 3.1, MP was designed to discriminate moves that are tied *after several iterations* of PP to speed up the end of the process. Thus, it is quite important not to start MP at the beginning of the PP process. If the MP technique is applied from the beginning of PP, then, on tactical positions, the program may select a very bad move. This kind of blunder occured recently in a game against CrazyStone [9], the new Monte-Carlo Go program of Rémi Coulom. Indigo overlooked the good move on a tactical position near the end of the game. This error did not change the outcome of the game, but it costed the loss of a large group of stones, which could have been avoided. As against this, we may state that its merit was to point out the problem. On this position which was quite near the end of the game, two moves only (and not eight as usual) were selected by the knowledge-based move selector, let us say A, the good one, and B, a bad one. Consequently, the PP process started with two moves, i.e., less than the threshold. Thus, the MP rule was applied from the start of PP. On the mentioned position, playing random games starting by B and A lead to positions in which Indigo, in its view, expected to obtain everything (because move A did not work for CrazyStone, but it still was obliged to be played as second move by the MP rule). Besides, playing random games starting by A and B led to stable positions in which half of the points was for Indigo, and the other half for CrazyStone. Thus, Indigo assessed (wrongly) that B was superior to A. To debug this blunder, we observed what would have happened without MP. Then, we saw that playing random games starting by B lead to positions in which CrazyStone obtained everything (because move A that did not work for CrazyStone, was not forced as second move by the MP rule). Furthermore, playing random games starting by A led to stable positions in which half of the points was for Indigo. Thus, without MP, PP quickly and correctly found out that A was superior to B. Consequently, to fix up the bug, we added another threshold forbidding to use MP before a minimal number of random games. This error underlines the fact that MP has the big downside of considering that a good move for the opponent is also a good move for itself, which is a famous Go proverb, but which is wrong in many tactical situations.

6 Conclusion and Perspectives

We presented two pruning heuristics: Miai Pruning and Set Pruning. They were intended to improve the existing Progressive Pruning technique. Miai Pruning actually simulates the miai principle used by human players which consists in playing the second move when the first is played. Set Pruning manages two sets of moves, GOOD and BAD, and tries to prune BAD when possible. These two pruning techniques have been assessed on 9×9 Go boards first. MP is domain-dependent and experimentally effective both in time and playing level, when N_{select} is low and when combined with a depth-one search. Weak MP has been shown effective both in time and playing level on 9×9 boards with depth-one search. However, MP and WMP are not effective within a depth-n search. Moreover, the recent history of Indigo showed that MP must be used only after a minimal number of random games, when PP alone has shown that it cannot break the tie between the remaining moves. SP seems experimentally effective in time as well, but it does not offer a satisfactory compromise between time and move quality. Combinations of MP and SP has also been tested but they all failed. Finally, we scaled WMP up to 19×19 boards, and we obtained a significant speed-up (about 1.3). Besides, we gained 4 points in terms of playing level.

Our experiments assessed the effect of MP within the global tree search algorithm proposed in [4]. A further step consists in translating MP into the game-tree search framework forgetting the statistical framework presented here. Besides, SP is general, and to this extent, it should be tried on other games with a high branching factor, such as Amazons [11]. Finally, assessing the ideas of [16] within the Monte-Carlo Go landscape, and performing local statistical search are still on our to-do list.

References

1. B. Abramson. Expected-outcome: a General Model of Static Evaluation. *IEEE Transactions on PAMI*, 12:182–193, 1990.
2. D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The Challenge of Poker. *Artificial Intelligence*, 134:201–240, 2002.
3. B. Bouzy. Associating Domain-Dependent Knowledge and Monte-Carlo Approaches within a Go Program. *Information Sciences*, 175:247–257, 2005.

4. B. Bouzy. Associating Shallow and Selective Global Tree Search with Monte-Carlo for 9×9 Go. In *4th Computers and Games conference (CG 2004)* (eds. H.J. van den Herik, Y. Björnsson, and N.S. Netanyahu), LNCS 3846, pages 67–80, Springer-Verlag, Berlin, 2006.
5. B. Bouzy. Indigo Home Page. www.math-info.univ-paris5.fr/ ~bouzy/INDIGO.html, 2005.
6. B. Bouzy and T. Cazenave. Computer Go: an AI-oriented Survey. *Artificial Intelligence*, 132:39–103, 2001.
7. B. Bouzy and B. Helmstetter. Monte-Carlo Go Developments. In *10th Advances in Computer Games (ACG10), Many Games, Many Challenges* (eds. H.J. van den Herik, H. Iida, and E.A. Heinz), pages 159–174, Kluwer Academic Publishers, Boston, 2004.
8. B. Brügmann. Monte-Carlo Go. www.joy.ne.jp/welcome/igs/Go/computer/-mcgo.tex.Z, 1993.
9. R. Coulom. CrazyStone Home Page. remi.coulom.free.fr/CrazyStone/, 2005.
10. D. Fotland. Go Intellect Wins 19x19 Go Tournament. *ICGA Journal*, 27(3):169–170, 2004.
11. J. Lieberum. An Evaluation Function in the Game of Amazons. In *10th Advances in Computer Games (ACG10), Many Games, Many Challenges* (eds. H.J. van den Herik, H. Iida, and E.A. Heinz), pages 299–308, Kluwer Academic Publishers, Boston, 2004.
12. M. Müller. Computer Go. *Artificial Intelligence*, 134:145–179, 2002.
13. M. Müller. Position Evaluation in Computer Go. *ICGA Journal*, 25(4):219–228, 2002.
14. J. Schaeffer and H.J. van den Herik. Games, Computers, and Artificial Intelligence. *Artificial Intelligence*, 134:1–7, 2002.
15. B. Sheppard. World-Championship-Caliber Scrabble. *Artificial Intelligence*, 134:241–275, 2002.
16. B. Sheppard. Effective Control of Selective Simulation. *ICGA Journal*, 27(2):67–80, 2004.
17. G. Tesauro and G. Galperin. On-line Policy Improvement using Monte-Carlo Search. In *Advances in Neural Information Processing Systems*, pages 1068–1074. MIT Press, 1996.