

# Move Pruning Techniques for Monte-Carlo Go

Bruno Bouzy<sup>1</sup>

Université René Descartes, Paris,  
bouzy@math-info.univ-paris5.fr,  
<http://www.math-info.univ-paris5.fr/~bouzy/>

**Abstract.** Progressive Pruning (PP) is used in the Monte-Carlo go playing program INDIGO. For each candidate move, PP launches random games starting with this move. PP gathers statistics on moves, and it prunes moves statistically inferior to the best one [5]. This paper 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 on  $9 \times 9$  boards, and MP improves slightly the playing level. Scaling up MP to  $19 \times 19$  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 [13, 11] mainly for two reasons. First, the branching factor of the game tree and the game length forbid global tree search. Second, evaluating non terminal go positions is hard [12]. Meanwhile, computer go has been used as an appropriate testbed for AI methods [6] during the last decade. In our work started twelve years ago, we have developed a go playing program, INDIGO [7]. It turns out that since 2002, INDIGO includes a Monte Carlo approach that enriches the knowledge-based approach developed previously. Our Monte Carlo approach was started on experiments [5] reproducing the original approach of Monte Carlo go [8]. These experiments assessed different enhancements to the basic Monte Carlo algorithm, and Progressive Pruning was the enhancement which is used now in INDIGO. Then, in 2003, we have associated our Monte Carlo go approach with our knowledge-based approach [4] and with a global tree search approach [3]. These two associations were successful because INDIGO won the bronze medal at the 2004 Olympiad on  $19 \times 19$  go [9] but they are not the topic of the current paper. The initial question that motivates the work presented here is how to improve Progressive Pruning (PP) in the case of the game of Go, and in general. This year we have assessed two pruning techniques intended to improve PP: Miai Pruning (MP) and Set Pruning (SP). The aim of this paper is to introduce these two new pruning techniques, and to provide an experimental assessment.

Section 2 relates the works dealing with Monte Carlo games, and it reminds the idea underlying PP. Section 3 defines the two pruning techniques, MP and SP,

to be evaluated. Then, section 4 yields the results of the experiments assessing these two techniques in isolation, and their combinations. Before conclusion and perspectives, some remarks are discussed by section 5.

## 2 Related work and motivations

### 2.1 Monte Carlo in computer games

Monte-Carlo methods were designed in order to simulate physical models. Because they used random number generation like games in the casino, the name Monte Carlo came up. In turn, Monte-Carlo methods have been used in computer games, and to some extent a loop has been closed. In games containing hidden information, such as Poker and Scrabble, hidden information is sampled with random distributions plausible with 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 [14]. In games containing randomness in their rules, such as Backgammon, random simulations are very naturally used [16]. In complete information games not containing 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.

Evaluating a position of a two-person complete information game with statistics was described by [1]. 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 6x6 Othello.

[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 think that Abramson's approach, or Brüggmann's, are appropriate for the game of Go because they enable the program to reach terminal positions easy to evaluate and especially representative of the current position in the sense they are derived from the current position. By computing a mean on terminal positions reached at random, the program gets a very good evaluation of the current position. Computing a Monte Carlo evaluation costs much more time than computing a conceptual evaluation using domain-dependent knowledge, but we believe this cost is worthwhile. This is the reason why we follow this approach in INDIGO, and the next subsection reminds how PP is used in INDIGO.

## 2.2 Progressive Pruning (PP)

The aim of PP is to choose the best move. The current description is based on [5]. As contained in the basic idea of Abramson, each move has a mean value  $m$ , a standard deviation  $\sigma$ , 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 up by practical experiments. Currently,  $1.5 \leq r_d \leq 2.0$  is 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.  $\sigma_e$  is called 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.  $N_{rg}$  being the current number of random games performed, the standard deviation of the mean value computed after  $N_{rg}$  random games is in  $1/\sqrt{N_{rg}}$ . Therefore, moves are pruned as  $N_{rg}$  increases, and the number of candidate moves decreases while the process is running. The process stops either when there is only one move left, or when the moves left are statistically equal, or 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, [15] uses a clever scheme to drive the choice of which simulation to perform on which move, and this scheme does not prune any move. However, the current work assumes that PP is used, and it tries to improve it. The current work does not debate on the use or non-use of move pruning.

The background of this work is the architecture of INDIGO. INDIGO's architecture is made up with a pre-selection module and a Monte Carlo module.  $N_{select}$  is the number of moves output of the pre-selection module, and input of the Monte Carlo module. As long as PP is running, the number of possible moves is decreasing from  $N_{select}$  down to 1 and the process stops. Two remarks can be made. First, PP spends most of its time when two possible moves are left. Most of the time, the evaluations of the two moves left are almost equal, and a lot of iterations are necessary to separate them. Thus, the first requirement 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  $N_{select}$ . (This is not completely true. Actually, INDIGO’s playing level reaches an optimal value with  $N_{select} = 8, 16$  or  $32$ , depending on the size of the board, and then, it decreases.) However, the optimal value of  $N_{select}$  can be quite high and the second requirement is to reduce the time spent by PP to eliminate moves at the beginning of the process. To sum up, we need a pruning technique that lowers the time spent when two or few moves are left at the end of the process, and another technique to quickly eliminate most of the moves at the beginning of the process, all of this in keeping the same statistical confidence when pruning. In this aim, we have designed two pruning techniques, each of them answering one requirement.

### 3 Two pruning techniques

This section describes the two pruning techniques answering our two requirements. Miai Pruning (MP) is the technique that speed-up the end of PP process when few moves are left, and Set Pruning (SP) is the technique that speed-up the beginning of PP process when many moves are input of the MC module.

#### 3.1 Miai Pruning (MP)

To highlight the presentation of MP, and without any loss of generality, let’s 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. To do this, PP launch many games starting with Black A, the following moves being randomly chosen, and many games starting with Black B, the following moves also being randomly chosen. Unfortunately, in the half of the games starting with move A, B is also played by Black. In addition, in the half of the games starting with move B, A is also played by Black. Thus, in the half of the random games played out to separate 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 seldom in Go, the half of the random games does not help much to discriminate A and B. Therefore, the idea of MP is to launch games starting with Black A and White B, and launch games starting with Black B and White A to separate A and B. To make MP working when the number of possible moves is arbitrary, 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 first, and White B in second. The term “miai” came up because it is used by human go players for the same concept. “miai” means equivalent. When two moves are miai, if a player moves on one of them, the other player moves on the other one. Thus, 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 (SP)

Let's assume that  $N_{possible}$  moves are left. First, to be cautious and avoid very bad pruning due to bad chance,  $N_{min}$  is devised by PP to forbid any pruning before  $N_{min}$  random games per move. Second, at the beginning of the PP process, the  $\sigma$  of 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  $\sigma$  not only to the possible moves but also to all the possible *sets* of moves of size  $N_{possible}/2$ .  $N_{rgpm}$  being the number of random games performed per candidate move, the mean value of the random games performed given that the first move belongs to a given set of size  $N_{possible}/2$  is known with a  $\sigma$  that is in  $1/\sqrt{N_{rgpm} \times N_{possible}/2}$ . Consequently, the width of the confidence interval around the mean value associated to sets of moves is  $\sqrt{N_{possible}/2}$  times smaller than the width of the confidence interval around the mean value associated to moves. Thus, it is possible to prune 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. This idea is very attractive because in practice we do not need to consider all of the possible sets of size  $N_{possible}/2$ , which would be terrible, 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  $N_{possible}/2$  best moves, called GOOD, and the set of the  $N_{possible}/2$  worst moves, called BAD. The mean associated to GOOD is the highest mean associated to any other set of size  $N_{possible}/2$ , and the mean associated to BAD is the lowest mean associated to any other set of size  $N_{possible}/2$ . Therefore the first set pruning to occur is the pruning of BAD. In practice, SP works as follows. In addition to the mean and  $\sigma$  computed by PP for each move, SP builds the two sets GOOD and BAD, and computes their means and their  $\sigma$ . 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, this section evaluates the relative merits of MP and SP, of their direct association (M+SP), their strong association (M+gbSP), and of a special combination of the two (M+gbP), all of this regarding time and playing level. We end up this section with an all-against-all tournament gathering the best programs of the experiments.

Since we explore move pruning abilities of a Monte-Carlo go program, we want first to observe the move pruning effect isolated from deep tree search effects. Thus, we have performed experiments with depth-one search only. Furthermore, because we need a lot of game results to obtain a sufficient statistical significance, and because  $19 \times 19$  games are too long, we have used  $9 \times 9$  boards. When a pruning technique has been demonstrated as performing well on  $9 \times 9$  boards at depth-one, it is assessed in a second stage either at depth-n on  $9 \times 9$

boards (see subsection 4.9) or at depth-one on  $19 \times 19$  boards (see subsection 4.10).

For each technique, we set up experiments to assess its effect on the time and playing level. One experiment consists in 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  $9 \times 9$  games played by our programs is roughly 15 points, 200 games enable our experiments to lower  $\sigma$  down to 1 point and to obtain a 95% confidence interval of which the radius equals  $2\sigma$ , i.e., 2 points. We have used 2.8 GHz computers. Furthermore, all programs in this work 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 assessed program is simple: for example,  $\text{PRUNE}(MP = true)$  is the program that uses additionally the MP technique, and so on for  $\text{PRUNE}(SP = true)$  or  $\text{PRUNE}(N_{select} = N)$ .

#### 4.1 $N_{select}$ versus $N_{select}/2$

Before assessing the effects of the two techniques, and because we want to obtain the best playing level and the minimal response time, knowing the effect of increasing the value of  $N_{select}$  is worth remembering. Table 1 shows the effects of simply doubling  $N_{select}$ .

**Table 1.** Result of doubling  $N_{select}$ . Each number corresponds to a confrontation between  $\text{PRUNE}(N_{select})$  and  $\text{PRUNE}(N_{select}/2)$

$N_{select}$	4	8	16	32	64
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

From  $N_{select} = 2$  up to  $N_{select} = 32$ , table 1 highlights the 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  $\text{PRUNE}(N_{select} = 32)$  is slightly superior to  $\text{PRUNE}(N_{select} = 16)$ , and even that  $\text{PRUNE}(N_{select} = 64)$  is almost equal to  $\text{PRUNE}(N_{select} = 32)$ . Regarding the relative time between the two programs, for low values of  $N_{select}$ ,  $\text{PRUNE}(N_{select})$  is about twice slower than  $\text{PRUNE}(N_{select}/2)$ , but for high values of  $N_{select}$ ,  $\text{PRUNE}(N_{select})$  is almost as fast as  $\text{PRUNE}(N_{select}/2)$ .

The results of this introductory experiment show that increasing  $N_{select}$  from 2 up to 16 is worthwhile considering in term of playing level on  $9 \times 9$  for PRUNE. The explanation is simple. 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 good at selecting the right move. With this assumption, the greater  $N_{select}$ , the greater 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 high enough, the tactically bad moves are unfortunately input of Monte-Carlo. Monte-Carlo is bad at recognizing tactically bad moves, which explains that  $\text{PRUNE}(N_{select} = 64)$  is worse than  $\text{PRUNE}(N_{select} = 32)$ . Meanwhile, the time for obtaining the overall playing level jump is multiplied by a factor 6.

## 4.2 Miai Pruning versus Progressive Pruning

First, this subsection compares MP with PP. Second, it shows the effects of simply doubling  $N_{select}$  while using MP.

**Table 2.** Result of MP against 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 the current paper, the result of  $\text{PRUNE}(MP = true, N_{select} = 2)$  is crucial to comment upon. First,  $\text{PRUNE}(MP = true, N_{select} = 2)$  has a non-negative result against  $\text{PRUNE}(MP = false, N_{select} = 2)$ : +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 answers the requirements presented in section 3. The non-negative mean score and the speed enhancement of two next columns ( $N_{select} = 4, 8$ ) 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 by table 1, table 3 shows that with MP the playing level also increases in  $N_{select}$ . Between  $N_{select} = 2$  and  $N_{select} = 4$ , the return improves more quickly with MP than without MP. However, for high values of  $N_{select}$  the return diminishes more quickly with MP than without MP. Moreover,  $\text{PRUNE}(N_{select} = 64)$  looks like inferior to  $\text{PRUNE}(N_{select} = 32)$  with some

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

$N_{select}$	4	8	16	32	64
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

statistical significance, which is astonishing. This 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 3x3 pattern urgencies [4]. 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 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 3x3 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 pseudo-random generator using domain-dependent knowledge, using 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.

**Table 4.** Result of SP against 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

For low values of  $N_{select}$ ,  $\text{PRUNE}(SP = true)$  plays at the same level as  $\text{PRUNE}(SP = false)$  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 values.

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

#### 4.4 M+SP versus Progressive Pruning

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

**Table 5.** Result of M+SP against PP

$N_{\text{select}}$	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 against PP. The results are bad. Losing by eleven or twelve points on average on  $9 \times 9$  boards is huge in go standards. This result is disappointing. While SP did not give good results, it was risky to associate them so directly. The next experiment aims at associating MP in a sophisticated way.

#### 4.5 M+gbSP versus Progressive Pruning

Since the direct association M+SP did not work well, we have tried a more sophisticated association of MP and SP, called M+gbSP. In addition to MP and SP, the two sets, GOOD and BAD updated by SP, are 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.** Result of M+gbSP against PP

$N_{\text{select}}$	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

Table 6 shows the results of M+gbSP against PP. The results are still bad. Losing by ten or fifteen points on average on  $9 \times 9$  boards is huge in go standards. Our current explanation is that the strong association reinforce the pruning 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 in GOOD, thus the mean value of M is penalised. 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 in 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 associates MP with the sole use of the two sets, GOOD and BAD, but not with SP. We call this association M+gbP. GOOD and BAD are used in the same way as they are used in the strong association 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.** Results of M+gbP against 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 7 shows the results obtained by M+gbP against PP. The relative speed is higher than it was in table 2. For  $N_{select} = 2, 4, 8, 16$ , the playing level of M+gbP seems identical to the playing level of MP. However, for  $N_{select} = 32$  or 64, the results are still bad. The arguments highlighted by 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,  $N_{select} = 2$ ). 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 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	0.7	1	0.7	0.7	0.7
4	2	1.4	2	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	5.2	6.2	2.7	2.0	4.5
64	7.5	6	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( $N_{select} = 8, 16, 32$ ). 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.** 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

Table 10 gives the final rankings with the average score per game. The  $\sigma$  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. 32 seems the best value of  $N_{select}$ , 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 being 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.

#### 4.8 Weak Miai Pruning (WMP)

Since the all-against-all tournament has shed the light on a weakness of MP when  $N_{select} = 16, 32$ , this subsection now 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  $T$ .

**Table 11.** Result of Weak MP (WMP) against 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

Conversely to simple 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,  $\text{PRUNE}(WMP = true, N_{select} = 2, 4)$  keeps the positive result shown by table 2. Second,  $\text{PRUNE}(WMP = true, N_{select} = 8, 16)$  has a very positive result against  $\text{PRUNE}(WMP = false, N_{select} = 8, 16)$ : +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 ( $N_{select} = 32, 64$ ) of table 11 confirms the effectiveness of WMP, and remove the negative mean scores of table 2. Finally, the speed is enhanced significantly, multiplied by 1.5 for  $N_{select} = 2$ , and not lowered for high values of  $N_{select}$ . To sum up, WMP is experimentally demonstrated to be superior to PP on  $9 \times 9$  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 $9 \times 9$ boards

The result obtained by WMP within the basic MC framework on  $9 \times 9$  boards, namely depth-one search, suggests using WMP in the framework associating MC and Tree Search [3]. Currently, INDIGO uses a depth-3 global tree search on  $9 \times 9$  boards. Consequently, we set up an experiment assessing  $\text{PRUNE}(WMP =$

*true*,  $N_{select} = 8$ ,  $Depth = 3$ ) against  $\text{PRUNE}(WMP = \textit{false}, N_{select} = 8, Depth = 3)$  on  $9 \times 9$  boards. It turns out that, although playing 5% faster,  $\text{PRUNE}(WMP = \textit{true}, N_{select} = 8, Depth = 3)$  is 1.7 point inferior to  $\text{PRUNE}(WMP = \textit{false}, N_{select} = 8, Depth = 3)$  and wins 45% of games only. Thus, integrating WMP with global tree search on  $9 \times 9$  boards is a failure.

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 to use because of time constraints, when depth-2 search cannot be used. Second, [3] and MP both expand the child nodes of a parent node when the number of children decreases and reaches a threshold ( $W$  – in [3] and  $T$  in MP). Therefore, the two techniques do not live well all together. Finally, [3] 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 $19 \times 19$ 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  $9 \times 9$  boards. After several experiments, SP does not answer our initial expectation, but MP, and more particularly WMP, is still worth considering. Therefore, in a second stage, WMP deserves a  $19 \times 19$  assessment. To make the programs playing in adequate time on  $19 \times 19$  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  $19 \times 19$  boards from  $9 \times 9$  boards, the maximal number of random games is reduced in a 40% proportion. Obviously on  $19 \times 19$  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  $19 \times 19$  boards in the same way as it does on  $9 \times 9$  boards. After 400 games,  $\text{PRUNE}(WMP = \textit{true}, N_{select} = 8)$  turns out to be +4 point superior to  $\text{PRUNE}(WMP = \textit{false}, N_{select} = 8)$  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  $19 \times 19$  with depth-one search. More interesting is the fact that  $\text{PRUNE}(WMP = \textit{true})$  used 36 minutes on average to complete one  $19 \times 19$  game. Meanwhile,  $\text{PRUNE}(WMP = \textit{false})$  used 46 minutes on average. Thus  $\text{PRUNE}(WMP = \textit{true})$  is 1.27 speedier than  $\text{PRUNE}(WMP = \textit{false})$ . This positive result on  $19 \times 19$  boards is explained by the fact that 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 call incentive of a move the difference between the MC evaluation of the position reached by this move and the current position MC evaluation. 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) 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 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 set of moves would exhibit the same move quality versus CPU time tradeoffs obtained by pruning individual moves. The experiments' results are not inconsistent with this hypothesis.

Scared by long experiments on  $19 \times 19$  boards, we have chosen to spend the CPU time to perform  $9 \times 9$  games first. Considering that INDIGO uses depth-3 on  $9 \times 9$  boards, and that MP does not work well with depth- $n$  search, all this work does not result in visible behaviour of INDIGO on  $9 \times 9$ . Hopefully, scaling up to  $19 \times 19$  boards after assessing the goodness of MP was a good surprise. MP works well with depth-one search and INDIGO uses depth-one on  $19 \times 19$  boards. Considering INDIGO's development, the speed-up and move quality improvements are finally effective on  $19 \times 19$  boards, and not on  $9 \times 9$  boards.

## 6 Conclusion and perspectives

We have presented two pruning heuristics: Miai Pruning and Set Pruning, 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  $9 \times 9$  go first. MP is domain-dependent, and it is experimentally effective both in time and playing level when  $N_{select}$  is low and when associated with depth-one search. Weak MP has been shown effective both in time and playing level on  $9 \times 9$  boards with depth-one search. However, MP and WMP are not effective within a depth- $n$  search. SP seems experimentally effective in time as well, but it does not offer a satisfactory compromise between time and move quality. Associations between MP and SP has also been tested but unfortunately they all failed. Finally, we have scaled WMP up to  $19 \times 19$  boards, and we have obtained a very significant speed-up (about 1.3). Besides, we have gained 4 points in terms of playing level.

Our experiments assessed the effect of MP within the global tree search algorithm proposed in [3]. 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 [10]. Finally, assessing the ideas of [15] within the Monte Carlo go landscape, and performing local statistical search are still in our to-do list.

## References

1. Abramson, B.: Expected-outcome : a general model of static evaluation. *IEEE Transactions on PAMI*, **12** (1990) 182–193
2. Billings, D., Davidson, A., Schaeffer, J., Szafron, D.: The challenge of poker. *Artificial Intelligence*, **134** (2002) 201–240
3. Bouzy B.: Associating shallow and selective global tree search with Monte-Carlo for 9x9 go, 4th Computers and Games conference, J. van den Herik, Y. Björnsson, N. Netanyahu (eds), Springer, (2004) to appear.
4. Bouzy B.: Associating knowledge and Monte-Carlo approaches within a Go program. 7th Joint Conference on Information Sciences, (2003) 505–508
5. Bouzy, B., Helmstetter, B.: Monte Carlo Go Developments. *Advances in Computer Games, Many Games, Many Challenges*. J. van den Herik, H. Iida, E. Heinz (eds) (2003) 159–174
6. Bouzy, B., Cazenave, T.: Computer Go: an AI oriented survey. *Artificial Intelligence*, (2001) **132** 39–103
7. Bouzy, B.: Indigo home page. [www.math-info.univ-paris5.fr/~bouzy/INDIGO.html](http://www.math-info.univ-paris5.fr/~bouzy/INDIGO.html) (2005)
8. Brüggmann, B.: Monte Carlo Go. [www.joy.ne.jp/welcome/igs/Go/computer/-mcgo.tex.Z](http://www.joy.ne.jp/welcome/igs/Go/computer/-mcgo.tex.Z) (1993)
9. Fotland, D.: Go Intellect wins 19x19 Go tournament. *ICGA Journal* (2004) **27** 3 169–170
10. Lieberum, J.: An Evaluation Function in the Game of Amazons. 10th *Advances in Computer Games, Many Games, Many Challenges*. H. Jaap van den Herik, Hiroyuki Iida, Ernst A. Heinz, Kluwer Academic Publishers, (2003) 299–308
11. Müller, M.: Computer Go. *Artificial Intelligence*, (2002) **134** 145–179
12. Müller, M.: Position Evaluation in Computer Go. *ICGA Journal*, (2002) **25** 4 219–228
13. Schaeffer, J., van den Herik, J.: Games, Computers, and Artificial Intelligence. *Artificial Intelligence*, **134** (2002) 1–7
14. Sheppard, B.: World-championship-caliber Scrabble. *Artificial Intelligence*, (2002) **134** 241–275
15. Sheppard, B.: Effective Control of Selective Simulation. *International Computer Game Association Journal*, (2004) **27** 2 67–80
16. Tesauro, G., Galperin, G.: On-line policy improvement using Monte-Carlo search. *Advances in Neural Information Processing Systems*, MIT Press (1996) 1068–1074