Grid coevolution for adaptive simulations; application to the building of opening books in the game of Go

Pierre Audouard, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, J. Perez, Olivier Teytaud

Grid coevolution for adaptive simulations; application to the building of opening books in the game of Go

P. Audouard*, G. Chaslot**, J.-B. Hoock***, J. Perez***, A. Rimmel***, O. Teytaud***

*Go Expert, ** Maastricht University, ***TAO (Inria), LRI, UMR 8623(CNRS - Univ. Paris-Sud), bat 490 Univ. Paris-Sud 91405 Orsay, France

Abstract. This paper presents a successful application of parallel (grid) coevolution applied to the building of an opening book (OB) in 9x9 Go. Known sayings around the game of Go are refound by the algorithm, and the resulting program was also able to credibly comment openings in professional games of 9x9 Go. Interestingly, beyond the application to the game of Go, our algorithm can be seen as a "meta"-level for the UCT-algorithm: "UCT applied to UCT" (instead of "UCT applied to a random player" as usual), in order to build an OB. It is generic and could be applied as well for analyzing a given situation of a Markov Decision Process.

1 Introduction

The game of go is an ancient Asian game with the nice property that it is much harder for computers than chess and therefore still unsolved - whereas nowadays computers play handicap games in chess against top-level humans (the handicap is here for helping the human!), the top human Go players still win very easily handicap games against humans (with the handicap for helping the computer!). However, the handicap has strongly reduced. In 1998, a 6D human player could win with 29 handicap stones [9] against a top program, Many Faces of Go, whereas in 2008, MoGo (running on the supercomputer Huygens) could win with 9 handicap stones against a top-level human player, Kim Myungwan (8^{th} Dan pro and winner of the US Open 2008); the human also said that MoGo could probably win also with only 8 handicap stones and was roughly 2nd or 3rd Dan. The most important tool for this strong improvement is Monte-Carlo Tree Search [12, 4, 8, 2, 7].

Complexity of OB in Go. Handcrafting OB in Go is quite hard because the usual knowledge is not encoded as a mapping *situation* → *answer* but much more as a set of local rules (termed Joseki), to be applied only after consideration of the global situation of the board. Also, a well known proverb in Go says "Learn joseki, loose two stones" (LJLTS): when a player learns these rules, he becomes weaker in a first time. The simple explanation to this fact is that local context in Go must always be related to the global situation of the board; so,

blindly applying newly learned opening sequences leads to inevitable mistakes concerning the global context of the game known as "errors in the direction of play". Indeed, choosing between several opening sequences depending on the global context is one of the hardest skill to acquire in the game of go, because the amount of experience required to handle the emptiness of the beginning of the game is huge.

For games in which standard game algorithms (using variants of minimax) work, building automatically an OB is more or less difficult, but usually it works [1], possibly with complicated tricks for removing bad moves [5]. In the case of 9x9 Go, efficient computer players are very recent and there's no positive result on 9x9 Go OB. Also, as these recent (and strong) players are mainly best-first search, i.e. focus on a small number of moves and not on all moves, it is not clear that they can be used for building OB - OB imply some diversity. We will here show that this approach works; this result, besides its positive effectiveness, is also interesting for understanding how best-first search algorithms can be efficient for games in spite of the fact that they only study a little set of moves (without efficient expert pruning).

Best first research in games. A main element in games is the branching factor. More than the incredible number of legal situations [14], the trouble in Go is that pruning rules are quite inefficient and therefore the "practical" branching factor is not so far from the "theoretical" branching factor. This is in particular true for building OB; requesting suggestions from professional players is quite difficult when the number of possible situations is huge as in the case of Go. How do we handle this problem ? A naive approach would require 1 request for the first move, 81 for the second move, and so on. How to reduce this combinatorial explosion ? As well as MCTS is efficient in front of the combinatorial explosion of Go, we show here that a meta-level of MCTS (MCTS on top of MCTS, see below) is efficient in front of the combinatorial explosion of OB. As the detailed presentation of MCTS is beyond the scope of this paper, we here refer the reader to [4, 8, 2, 7] for more information around MCTS, and here we insist only on the fact that MCTS is a coevolution [1].

MCTS leads to a "best-first" approach[2] in the sense that when we build a tree of future situations, a move is further analyzed as long as it is the move with best success rate in the simulations. Is this reasonable for OB ? The experimental answer given by this paper is positive, and we justify it as follows: (i) assume that for a fixed height d (i.e. there are d moves at most before the end), the estimate of the winning probability of winning converges to 0 or 1, depending on whether this situation is winning or not; (ii) then for a situation with height $d + 1$, the

[1] This looks obvious in our case as we have "real" games as initial points for our coevolution instead of random games, and "real" moves instead of naive random moves, but MCTS also is definitely a coevolution between variations as well.

[2] MCTS has often been emphasized as a compromise between exploration and exploitation, but many practitioners have seen that in the case of deterministic games, the exploration constant, when optimized, is zero or indistinguishable from zero, leading to a best-first approach.

move with best success rate will not stay the move with best success rate if it is not a winning move, and this will ensure diversification. Of course, this is quite "sequential": no alternate move is explored if the first explored move is still at a high success rate in spite of the fact that it is not the best move. But MCTS has shown that with a strong enough computational power, this approach is in fact very reliable and better than classical approaches; we here show that it is also reliable in the difficult framework of OB in spite of the intuitive diversity requirement.

All our experiments are based on MoGo, the first program which (i) won a game against a pro player in 9x9 Go (Comp.-Go Olympiads, Amsterdam 2007), (ii) won a game against a pro player without blitz in 9x9 Go (IAGO challenge, Paris, 2008), (iii) won a game against a pro player in 19x19 Go with handicap 9 (US Go Congress, Portland, 2008). The OB built as explained below was used in the Hakone 2008 computer-Go Cup, won by MoGo. All Go figures use the PSGO package. Comments which require a high-level in 9x9 Go are given by Pierre Audouard (P.A.). P.A. is teaching Go since 1992; he was 17^{th} in the World Amateur Go Championship (Tokyo 2008). He has a special interest in 9x9 Go and won the 1st international disabled Go tournament of 9x9 Go (Seoul, 2008). Section 2 presents the methods investigated in this paper and shows experiments in an artificial setup. Section 3 shows real-size experiments. The conclusion will emphasize some generality of the result.

2 Methods and preliminary results on fast time settings

The goal is the incremental building of a set of games (an OB), supposed to be used by a (human or computer) player as follows: *If the current situation is in the set of games, choose the most frequent move in won games.* We choose the most simulated move in won games, instead of the move with highest success rate - there are far less erroneous moves with the most simulated move in won games. This is consistent with the MCTS literature (see e.g. [15]). We use self-play as a basic tool for building OB, as in [10] (for computer-Shogi). Precisely, while looking for a methodology for building an OB with coevolution, we understood that MCTS is precisely a tool for building OB, usually applied to a naive random player and with small time settings, and that what we need is essentially MCTS with however (i) very different time settings (ii) a good computer player; in this paper, the good player is itself the original MCTS. Our non-evolutionary methods for building OB are as follows. **Expert OB:** A handcrafted set of games played by top level players and optimized manually (even pro games can be optimized offline by experts). **4H-OB:** A version of MoGo spending 4H studying each move plays against a MoGo spending 1s per move. After 4 moves, the MoGo playing 1s per move resigns. These two MoGos use no OB and are not adaptive (no learning). The 4H-OB was rejected as weaker than the expert OB.

Coevolution has already been used for building partial strategies in Chinese chess [11] and also in the game of Go [6], but with less impressive results - our

coevolution will therefore be exactly MCTS. MCTS is usually not presented as a coevolution, and we designed our coevolutionary algorithm (CA) for OB without conceiving it initially as a MCTS; but MCTS consists in evolving a white strategy and a black strategy in a given situation, by mutating the policy in the currently studied variation. Our CA is "MCTS applied to MoGo-black and MoGo-white", as well as MoGo is "MCTS applied to a naive random black player and a naive random white player" (not "very" naive; see [15] for details). Our co-evolutionary techniques for building OB are as follows, starting from a part of the handcrafted OB (the complete version was not available at that time). **Mutate bad moves (MBM, Algo. 1):** *MoGo plays against itself (each player has 6h per side on a quad-core, roughly equivalent to 24h per side as the speed-up of MCTS for 4 cores is very good). The two MoGo use the OB and choose the move with highest success rate if at least one move with success rate > 50 % is available. Otherwise, the standard MoGo algorithm is used.* **Mutate very bad moves (MVBM):** *Mogo plays against itself (each player has 6h per side). The two mogo use the OB and choose the move with highest success rate if at least one move with success rate > 10 % is available. Otherwise, the standard MoGo algorithm is used.* MVBM was introduced because, with MBM, there were too many cases in which black did not follow any OB at all because the success rate of black was lower than 50 %. Both algorithms are a coevolution, in which an individual is a game (the games won by black (resp. white) are the black (resp. white) population; each won (resp. lost) game is a good (resp. bad) partial strategy), and as in the Parisian approach [3], the solution (the OB) is made of all the population. This algorithm is used on a grid (www.grid5000.fr, a great grid provided freely for scientific experimentation). This introduces some constraints: λ is not known in advance and not constant (λ depends on the number of available CPUs, and jobs might be preempted). In order to have preliminary results, we first run MBM on a simplified setting (see section 2.1). We see first that the depth-first approach works fine, what is a great success as building OB in Go is quite difficult; these results, quite good for white, are based on MBM. We will see however that this approach is not satisfactory for black and therefore derived the MVBM approach.

2.1 Why we should mutate very bad moves only

We here experiment the MBM approach, on a simplified case of 10 seconds per move, 8-cores machine; in this easier framework we could generate 3496 games. The results follow:

Conditions	Before learning	After learning	Against handcrafted OB
Success rate (white)	51.5 % ± 1.8 %	64.3 % ± 1.9 %	64.1 % ± 1.8 %
Success rate (black)	48.5 % ± 1.8 %	48.0 % ± 1.9 %	46.1 % ± 1.8 %

The first column sums to 100 % as it is the success rate of white (resp. black) in the case of no learning. The second column shows the results for white (resp. black) after learning against a non-learning version: the sum is higher than one as

Algorithm 1 The "mutate bad move" (MBM) algorithm. λ is the number of machines available. The random choice is performed by MoGo with long time settings (except in section 2): therefore, it is quite expensive. Readers familiar with UCT/MCTS might point out that there is no exploration term in the version of MCTS presented here; however, this is the case in many successful MCTS implementations, and other implementations often have a very small constant which is equivalent, in fact, to 0.

```
Population = small handcrafted OB.
while True do
  for l = 1..λ, generate one individual (a game) in parallel with two steps as follows do
    s =initial state; g = (s).
    while s is not a final state do
      bestScore = 0.5
      bestMove = Not A Move
      for m in the set of possible moves in s do
        score =frequency of won games in Population with move m in s
        if score > bestScore then
          bestScore = score
          bestMove = m
        end if
      end for
      if bestMove = Not A Move then
        Mutation: bestMove = RandomChoice(s).
      end if
      s = nextState(s, bestMove) (transition operator)
      g = concat(g, s)
    end while
    Population = Population ∪ {g}
  end for
end while
```

MBM did a good job on average. The third column shows that the learning has provided better results than the handcrafted OB. However, we see no progress for black. This can be explained as follows: as long as black has not found a move with success rate $> 50\%$, he always mutates. Therefore, white improves his results by choosing moves with higher success rates, but not black. This is why in next sections, we will use the "mutate very bad moves" approach - asymptotically it is probably equivalent, but non-asymptotically it's better to use an approach in which 40% of success rate is better than 30 % of success rate instead of repetedly mutating in order to find at least 50 %.

2.2 Learn Joseki, loose two stones (LJLTS)

These results above look quite promising, but we then tested what happens if we use this learnt OB in a stronger version of MoGo (i.e. with bigger time settings), using 60s per move instead of 10s (still on 8-cores machines).

Validation of the OB for a stronger version of MoGo. Success rate of the handcrafted OB against no OB for 60 s per move: 50 % as black, 63 % as white. Clearly the handcrafted OB still works for this stronger MoGo. Success rate of the learnt OB (learnt as above, with 10s per move games) against handcrafted OB: 30.6 % as black, 40.7 % as white. The learnt OB, learnt for 10s per move, is definitely not sufficiently strong for 60s per move. These numbers show that a weak player can't estimate efficiently an opening sequence, whenever

this weak player performs hundreds of games - the opening sequence might be good for this weak-player, but this opening sequence will become a handicap when the player will become stronger.

Robustness of the handcrafted OB. We then switched to 300s per move, still on 8-cores machines, to see if the handcrafted OB was still efficient. Success rate of the handcrafted OB against no OB: 49.6 % as black, 57.0 % as white. We then considered an optimized handcrafted OB, still for 300s per move, and reached 49.7% as black, 62.9 % as white. This shows the LJLTS: if you learn Joseki (handcrafted opening moves) which are too difficult for you (if MoGo has only 10s per move), they are not efficient (you'd better use your own learnt opening moves). But if you become stronger (MoGo with 60s/move or 300s/move), you can trust the Joseki (the handcrafted rules).

3 Real-size experiments on MVBM for 9x9 Go, Komi 7.5

In this section we present the results of the real-size experiments on MVBM. Each game is played with 6h per side on a quad-core. 3000 games are played, i.e. the final population has 3000 individuals. MVBM is "Parisian": all the population is the solution. Also, there's no removal: the population is incrementally increasing, and no individual is never eliminated (it might just become negligible). The handcrafted OB used in this section is an improved version of the one used in the previous section. We first show that the learning version becomes stronger and stronger, in front of the non-learning version. Results are presented in figure 1. The good point is that on average on the learning we have a success rate as

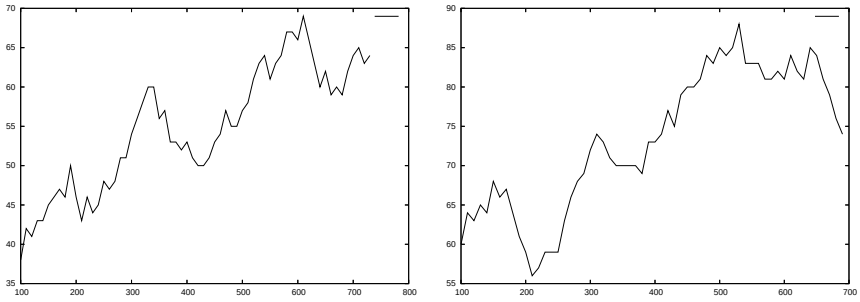

Fig. 1. Left: success rate as black against the non-learning version. Right: success rate as white against the non-learning version. For both figures, the abscissa is the time index (number of test games played); the plot is made on a moving average on 100 games.

follows against the version without OB, more than if we use the handcrafted OB (Figure 2, left). We can also check the results in self-play: is one of the player winning more and more often, converging to a resolution towards 100 % of won, i.e. a complete solving of the game ? We plot the results in figure 2 (right). High-level Go-players usually consider that with Komi 7.5, white has the advantage.

Color	Coevolutionary opening book	Handcrafted opening book
White	74.5% ± 2.3 %	62.9 % ± 3.8 %
Black	64.6 % ± 2.4 %	49.7 % ± 3.8 %

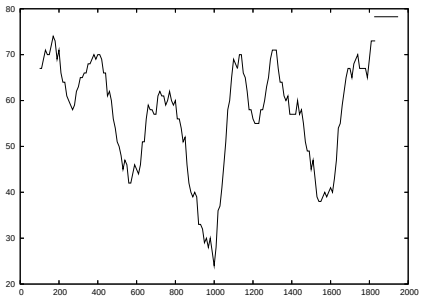

Fig. 2. Success rate of the self-learning version against itself, as white. This is consistent with usual knowledge about 9x9 Go with Komi 7.5 for strong players: the white player has the advantage.

This is supported by the final result (Figure 2). **Comment from P.A.:** *It is known that with Komi 7.5 white should have the advantage on average, if the level is sufficiently high - this is therefore consistent.*

Below, we: present some general statistics on the optimization run and the resulting OB (section 3.1); check some known points in Go (section 3.2); present links between variations (i.e. sequences) studied by MoGo and professional games (section 3.3).

3.1 Statistics on the final population of our grid-coevolution

We checked that all answers to all moves cited as classical in 9x9 Go in `http://senseis.xmp.net/?9x9Openings` have their answer in the OB. Therefore, if a (black or white) player plays his first move in this library of moves, then an opponent playing with the OB has an automatic answer. Also, on average, the number of moves played by MoGo inside his OB against GnuGo 3.6 and Fuego 0.2.2 is 3.28 and 4.64 moves respectively.

3.2 Comments by P.A. on the OB built by MVBM

We consider here some well known elements, proposed for test by P.A.: Mane Go, i.e. a strategy known very efficient for white after an initial move of black different from E5; Kosumi, i.e. a class of moves for which some elements are known. **Mane Go**, for the white player, consists in replying symmetrically to the black moves, for white, when black does not start with E5. With this simple strategy for white, black has to break the symmetry (by playing in the center E5) and is in a more difficult situation. P.A. tested Mane Go as follows. First move black F6: MoGo as white answers D4; then we test black D6 and MoGo as white replies symmetrically, i.e. F4. First move black F5: MoGo as white answers D5; then we test black E7 and MoGo as white replies symmetrically, i.e. E3. **Kosumi** consists in playing two stones (of the same color), in diagonal. This might be a good idea in a game, but usually not in the beginning. Here are the statistics in the final population. Many cases of Kosumi appeared just

a few times and then disappeared; the only one which appeared significantly is a Kosumi between the first and third move for black: E5 G5 F4. F4 and G5 are in Kosumi. However, this Kosumi is somewhat rare and is not chosen by MoGo when it uses the usage rule of the move used in most won games - it just appears in the exploration. However, this shows that our MVBM approach can spend time on weak moves. **Comment from P.A.:** *The absence of Kosumi for white before the seventh move is a good feature. Kosumi might happen later in the game. On the other hand, the choice E5 / G5 / F4 often studied by MoGo (though not selected at the end) for black is not a good one.* We can also check the currently analyzed openings: the last games are based on E5 / G5 / G4 / C5 or E5 / G5 / G4 / C6. This is a quite classical beginning. Also, P.A. could check that MVBM never plays as black E5/C5/E3/G5/E7 (a classical sequence of MoGo when openings are removed), which was quite bad.

3.3 Comments by MoGo on professional games, and their evaluation by a Go-expert

Due to length constraints, this section, which can only be evaluated by 9x9-Go experts, is reported to a technical report and we only summarize here the analysis of some professional games provided by P.A. MVBM does not like the variation E5/G5/G4/C4 found in several professional games; in the population, after E5/G5/G4/C4, Black wins with probability 67.9% (whereas C5 instead of C4 is fine). According to the population, E5/G5/G4/C4/F5 is strong for black. This is consistent with pro games, as the 5 professional games with this sequence were all won by black. Also, the only game played in which black answered F4 was lost by black; this move was not considered at all by MVBM. Only one game was played with H5 (E5/G5/G4/C4/H5); according to MoGo, this sequence is very good for black; however, black lost that game. Interestingly, MVBM has simulated also the next moves until the 20^{th} move; MVBM considers that black made a mistake later in the game and should have won with a different move (winning with probability 100% according to MVBM). **Comment by P.A.:** *MVBM is right when he says that C4 is not the best choice after E5/G5/G4 and that C5 should be chosen. Also, MVBM is right when he says that black made a mistake later in the game lost after the good initial sequence E5/G5/G4/C4/H5 - but the mistake is later than the move pointed out by MVBM. On the other hand, the move proposed by MVBM is quite strong and really leads to a good position for black.* Also, MVBM does not like the sequence E5/G5/F3; according to MVBM, White can then win with probability 75% by playing C6, or possibly C4 (70%), G3 (70%); D3 is possible also (50%) and all other moves are considered as weak by MVBM. A professional game features this sequence E5/G5/F3 with a win for black, and this is in accordance with MVBM as white then played C5, one of the bad moves according to MVBM. **Comment by P.A.:** *In spite of the fact that MVBM's prediction is in accordance with the result of the game, C5 a good move after E5/G5/F3. Other moves proposed by MoGo are also interesting and it's difficult to choose between them. C4/C6 are more complicated than C5, and G3 is a contact move, it might be dangerous - however, in 9x9, such an*

early contact move can be considered. Finally, MVBM analyzed E5/G5/G4/C5. A professional player played F4 in this situation, and MVBM explored this solution with the same preferred answer after E5/G5/G4/C5/F4 as the one played in this professional game; however, MoGo considers that F4 should be replaced by E3 with success rate 59 % instead of F4. **Comment by P.A.:** E3 is not a good idea. This is not a good reason for the result of the considered professional game.

4 Conclusion

We used a CA for generating OB for the game of Go. There's no big efficient OB available. First, the positive results follow. **First, effectiveness.** The resulting OB is satisfactory and consistent with results of professional games. It leads to quite good results in the sense that the program is much stronger with the OB than without. It is also satisfactory, in most cases, from the point of view of an expert human player: some known rules have been found independently by the algorithm (Kosumi, Mane Go, comments on professional games). **Second, grid-compliance.** Thanks to preemptable jobs (the algorithm is completely fault tolerant), the use of the grid was not too much a trouble for other users. It was also possible to have simultaneously a small number of non-preemptable jobs and plenty of preemptable jobs, so that the grid is used adaptively. A much bigger run is under progress on grid5000, and thanks to preemptable jobs has little impact on the grid. **Third, parallelization.** The speed-up is seemingly good, as (i) there's very little redundancies between individuals in the population (ii) the individuals are different just a very few moves after the initial sequence which is not randomly mutated. This is consistent with [13] which has shown that evolution strategies have a very good speed-up when the problem has a huge dimensionality: the domain here is huge [14]. **Fourth, user-friendly aspect.** It is easy to monitor the optimization, as at any fixed time step we can see which sequence is currently analyzed by the algorithm. **Fifth, generality.** We applied our algorithm to the empty Goban (initial situation), but it could be applied easily to a given situation (Tsumego analysis). Also, it can be used for analyzing a particular situation for any Markov Decision Process. After all, MVBM is an adaptive Monte-Carlo algorithm: it is based on an initial strategy, and incrementally improves this strategy, based on simulations - it is like a Monte-Carlo algorithm used for plotting a Value-At-Risk, but used adaptively. Applications in finance or power plant management is straightforward. Some less positive points are as follows: First, one of the solution proposed by the algorithm is weak according to P.A., namely E5/G5/G4/C5/E3. By the way, the solution was later replaced (in the current run), but there was no indicator, at that time, of the fact that E3 was a bad solution. We have an algorithm which is therefore consistent, but we can't guess a priori that a solution is still unstable. Also, E5/G5/F4 was not in the final result, but a long exploration has been made on this move, whereas it is, according to P.A., a poor solution that could be removed by expert knowledge around "Kosumi". Second, no free lunch:

the OB generated by a poor random player (MoGo with small time settings) is useless for a better MoGo (with longer time settings). This is consistent with [5]: the presence of erroneous moves is a major trouble when building OB.

Acknowledgements. We are very grateful to B. Bouzy, T. Cazenave, A. Cohen, R. Coulom, C. Fiter, S. Gelly, B. Helmstetter, R. Munos, Z. Yu, the computer-go mailing list, the KGS community, the Cgos server, the IAGO challenge, the Recitsproque company, the French Federation of Go, and Grid5000.

References

1. M. Buro. Toward opening book learning. *ICCA Journal*, 22:98–102, 1999.
2. G. Chaslot, M. Winands, J. Uiterwijk, H. van den Herik, and B. Bouzy. Progressive strategies for monte-carlo tree search. In P. Wang et al., editors, *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pages 655–661. World Scientific Publishing Co. Pte. Ltd., 2007.
3. P. Collet, E. Lutton, F. Raynal, and M. Schoenauer. Polar ifs + parisian gp = efficient inverse ifs problem solving. *Genetic Programming and Evolvable Machines*, 1(4):339–361, 2000.
4. R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, 2006.
5. C. Donninger and U. Lorenz. Innovative opening-book handling. In H. J. van den Herik, S. chin Hsu, T. sheng Hsu, and H. H. L. M. Donkers, editors, *ACG*, volume 4250 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2006.
6. P. Drake and Y.-P. Chen. Coevolving partial strategies for the game of go. In *International Conference on Genetic and Evolutionary Methods*. CSREA Press, 2008.
7. S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM Press.
8. L. Kocsis and C. Szepesvari. Bandit-based monte-carlo planning. In *ECML'06*, pages 282–293, 2006.
9. M. Müller. Computer go. *Artificial Intelligence*, 134(1-2):145–179, 2002.
10. J. Nagashima, T. Hashimoto, and H. Iida. Self-playing-based opening book tuning. *New Mathematics and Natural Computation (NMNC)*, 2(02):183–194, 2006.
11. C. Ong, H. Quek, K. Tan, and A. Tay. Discovering chinese chess strategies through coevolutionary approaches. In *IEEE Symposium on Computational Intelligence and Games*, pages 360–367, 2007.
12. L. Péret and F. Garcia. On-line search for solving markov decision processes via heuristic sampling. In R. L. de Mántaras and L. Saitta, editors, *ECAI*, pages 530–534. IOS Press, 2004.
13. O. Teytaud and H. Fournier. Lower bounds for evolution strategies using vc-dimensio n. In G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, editors, *PPSN*, volume 5199 of *Lecture Notes in Computer Science*, pages 102–111. Springer, 2008.
14. J. Tromp and G. Farnebäck. Combinatorics of go. In *Proceedings of 5th International Conference on Computer and Games*, Torino, Italy, May 2006.
15. Y. Wang and S. Gelly. Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii*, pages 175–182, 2007.