Current Frontiers in Computer Go

Arpad Rimmel, Olivier Teytaud, Chang-Shing Lee, Shi-Jim Yen, Mei-Hui Wang, Shang-Rong Tsai

Current Frontiers in Computer Go

Arpad Rimmel, Olivier Teytaud, Chang-Shing Lee, Shi-Jim Yen, Mei-Hui Wang, and Shang-Rong Tsai

Abstract—This paper presents the recent technical advances in Monte-Carlo Tree Search for the Game of Go, shows the many similarities and the rare differences between the current best programs, and reports the results of the computer-Go event organized at FUZZ-IEEE 2009, in which four main Go programs played against top level humans. We see that in 9x9, computers are very close to the best human level, and can be improved easily for the opening book; whereas in 19x19, handicap 7 is not enough for the computers to win against top level professional players, due to some clearly understood (but not solved) weaknesses of the current algorithms. Applications far from the game of Go are also cited. Importantly, the first ever win of a computer against a 9th Dan professional player in 9x9 Go occurred in this event.

Index Terms—Monte-Carlo Tree Search, Upper Confidence Trees, Game of Go

I. INTRODUCTION

THE game of Go is one of the main challenges in artificial intelligence. In particular, it is much harder than chess, in spite of the fact that it is fully observable and has very intuitive rules.

Currently, the best algorithms are based on Monte-Carlo Tree Search [1], [2], [3]; they reach the professional level in 9x9 Go (the smallest, simplest form) and strong amateur level in 19x19 Go.

During FUZZ-IEEE 2009, in Jeju Island, games were played between four of the current best programs against a top level professional player and a high-level amateur. We will use the results of the different games in order to summarize the state of the Monte-Carlo Tree Search algorithm, the main differences between the programs and the current limitations of the algorithm.

History of computer Go.

The ranks in the game of Go are ordered by decreasing Kyu, increasing Dan, and then increasing professional Dans: 20Kyu is the lowest level, 19K, 18K, ...,and 1K; 1Dan, 2D, 3D,..., and 7D; the first professional Dan 1P is then considered as nearly equivalent to 7D, followed by 2P, 3P, 4P,..., and 9P. The title "top pro" is given to professional players who recently won at least one major tournament.

9x9 Go: In 2007, MoGo won the first ever game against a pro, Guo Juan 5P, in 9x9, in a blitz game (10 minutes per side). This was done a second time, with long time settings, in 2008, also by MoGo and against Catalin Taranu 5P. The only

A. Rimmel and O. Teytaud are with the TAO team, Inria Saclay IDF, LRI, UMR 8623(CNRS - Universite Paris-Sud), bat 490 Universite Paris-Sud, 91405 Orsay Cedex, France. e-mail:rimmel@lri.fr. Chang-Shing Lee and Mei-Hui Wang are with the Dept. of Computer Science and Information Engineering, National University of Tainan, Taiwan. Shi-Jim Yen is with the Computer Science and Information Engineering department from the National Dong Hwa University, Taiwan. Shang-Rong Tsai is with the Chang Jung Christian University, Taiwan.

wins as black against a pro were realized by MoGo against Catalin Taranu (5P) in Rennes (France, 2009) and the win against C.-H. Chou (Taipei, 2009).

19x19 Go: In 1998, Martin Müller could win against Many Faces Of Go, one of the top programs at that time, in spite of 29 handicap stones, an incredibly big handicap, so big that it does not make sense for human players. In 2008, MoGo won the first ever game in 19x19 against a pro, Kim Myungwan, 8P, in Portland; however, this was with the largest usually accepted handicap, *i.e.* 9 stones. CrazyStone then won against a pro with 8 and 7 handicap stones in Tokyo (Aoba Kaori 4P, in 2008); finally, MoGo won with handicap 7 against a top level human player, Chou-Hsun Chou (9P and winner of the famous LG Cup in 2007), and against a 1P player with handicap 6 in Tainan (Taiwan, 2009).

During FUZZ-IEEE 2009 there was the first win of a computer program (the Canadian program Fuego) against a 9P player in 9x9 as white. On the other hand, none of the program could win against Chou-Hsun Chou in 19x19, in spite of the handicap 7, showing that winning with handicap 7 against a top level player is still almost impossible for computers, in spite of the win by MoGo a few months earlier with handicap 7. Also, during FUZZ-IEEE 2009, no program could win as black in 9x9 Go with komi 7.5 against the top pro.

The two human players.

Chou-Hsun Chou is a top level professional player born in Taiwan. He became professional in 1993 and reached 7P in 1997 and 9P in 1998. He won the LG Cup in 2007, beating Hu Yaoyu 2 to 1.

Shen-Su Chang is a 6D amateur from Taiwan.

Technical terms from the game of Go.

In this section we define several Go terms. A *group* is a connected set of stones (for 4-connectivity). A *liberty* is an empty location, next to a group; a group is *captured* when it has no more liberties; it is then removed from the board. A group is termed *dead* when it is definitely going to be captured. An *atari* is a situation in which a player plays a move in the liberties of a group, so that only one liberty remains. A *semeai* is a fight between two groups, each of them being alive only if it kills the other (unless seki cases). A *seki* is a situation in which two groups have common liberties and none of the players can play in these liberties without being in self-atari. The *komi* is the number of points given to white, as a compensation for playing second. The *handicap* in a game is a number of stones; with handicap N, the black player plays N stones before white plays its first move. Even games are games with handicap 0 and komi around 7.5 (the precise komi depends on federations and rules). A *moyo* is an area of the

board where one player has a lot of influence and that could become territory.

The rest of this paper is organized as follows: Section II describes the main concepts in Monte-Carlo Go. Section III introduces the results and comments for the FUZZ-IEEE 2009 computer Go invited session. Section IV concludes.

II. MONTE CARLO TREE SEARCH ALGORITHM AND IMPLEMENTATIONS

Section II-A describes the main concepts in Monte-Carlo Go. Section II-B describes techniques for dealing with the large action space. Section II-C explains how to extract additional useful information from simulations. Section II-D presents some expert modules useful for biasing the Monte-Carlo part. Section II-E will summarize some known differences between the programs.

A. Main concepts in Monte-Carlo Go

The main concepts in Monte-Carlo Tree Search were defined in [1], [2], [3]; one of the most well known variants is Upper Confidence Bounds applied to Trees [3]. The main idea is to construct a tree of possible futures. This tree will be biased in order to explore more deeply moves that have good results so far. This is done by the repetition of 4 steps as long as there is some time left: *descent*, *evaluation*, *update*, *growth*.

In the *descent* part, we use the statistics of the tree to chose new nodes until we reach a node outside of the tree. This is done by considering that the selection of a child is a bandit problem[4]. In a bandit problem, you have a fixed number of arms, each arm is associated to an unknown probability distribution. At each turn you select an arm and receive a reward which is drawn according to the distribution of the arm. Your goal is to maximize your rewards. The formula used to solve this problem is called a bandit formula and is usually based on a compromise between exploration and exploitation; a classical example is given below. This formula is used during all the descent step.

In the *evaluation* part of the algorithm, also called playout, the goal is to have a value for the nodes selected during the descent part. In order to do that, a legal move is chosen randomly (but not uniformly) until the game is finished; see section II-D.

In the *update* part, the statistics of the tree are updated according to the result of the game.

In the *growth* part, the node just outside of the tree selected at the end of the descent part is added to the tree.

All algorithms based on this principle will be termed Monte-Carlo Tree Search in the rest of this paper.

An efficient way of solving the bandit problem is to chose the move with the highest upper confidence bound. This is done with the UCB formula. It consists in choosing the child c of the current situation q which maximizes:

$$s_q(c) = \frac{W(c)}{n(c)} + C\sqrt{\frac{\log(N(q))}{n(c)}}, \qquad (1)$$

where
- $s_q(c)$ is the score of child c of node q;
- $n(c)$ is the number of simulations of move c;
- $N(q)$ is the number of simulations of state q;
- $W(n)$ is the number of won simulations of node n;
- the constant C controls the compromise between exploitation of good moves and exploration of new moves.

When an other term that plays the role of exploration, like the RAVE values originating in [5], is added to the formula, the constant C becomes usually very small or even zero:

$$s_q(c) = \alpha\frac{W(c)}{n(c)} + (1-\alpha)\frac{W_{rave}(c)}{n_{rave}(c)}. \qquad (2)$$

The "RAVE" values will be defined later (Eq. 4). In the rest of this paper, we will identify the node c and the move played to obtain c from q; this is an approximation only, as MoGo has a transposition table as well as many strong programs; this will just clarify the equations.

When the bandit part is based on Eq. 1 or a variant of it, the MCTS is termed UCT (Upper Confidence Trees[3]). In the case of Go, more sophisticated formula are usually preferred; nonetheless, UCT provides a very sound and principled way of designing a general purpose MCTS. This is in particular important as MCTS is particularly well known for its efficiency in general game playing, *i.e.* when the game is not known in advance and the program must read the rules (in a given formalism) before playing[6].

There are also several other modules which enhance the performance, detailed in sections below.

B. Bandits for large action spaces: introducing a bias in the tree search

The most classical idea for choosing a move in the tree part is to maximize the score given in Equation 1. However, Equation 1 gives score $+\infty$ to moves which have no simulation. This implies that if there are N legal moves at situation q, then the first N simulations at node q will all choose one different initial move. This is of course a poor policy. Therefore, other solutions have been proposed: first play urgency, progressive widening and progressive unpruning. The last two are based on ranking heuristics, which are detailed later.

First Play Urgency: [7] proposes the "first play urgency" (FPU); this is a constant score, given to moves with no simulations. The FPU can be improved, e.g. by replacing the constant by a function of Go expertise. However, FPU was replaced by other rules in all strong implementations (note however that for other applications with less expertise available, FPU might be a good rule of thumb).

Progressive widening: [8] proposed progressive widening, consisting in optimizing Eq. 1 only among moves with index lower than $\Theta(n^K)$; precisely,

$$\text{decision}(q,n) = \arg\max_{index(q,c)\le n^K} s_q(c) \qquad (3)$$

for the nth simulated move at situation q. This requires the use of a function $index(q,c)$, which gives to each legal move c at situation s a rank. Usually, a prior is computed for each c at situation q, and then $index(q,c)$ is the rank of move

c according to this prior; therefore, what is really needed for progressive widening is a score for each move, as for progressive unpruning.

It has been shown in [9] that even if $index(.)$ is a random ranking of moves, this algorithm can provide an improvement; in applications, K ranges between $\frac{1}{4}$ and $\frac{1}{2}$ depending on the efficiency of the heuristic [8], [10]. Interestingly, with progressive widening, UCT can be applied to problems with infinite action space. However, in many problems and in particular in Go and Havannah, progressive unpruning (defined below) performs better and has been chosen in recent implementations [5], [11].

Progressive unpruning: Instead of an abrupt change as progressive widening, which adds new moves to the pool of moves considered in the $\arg\max$ of Eq. 3, [2] proposes to add a term in Eq. 1, *e.g.* as follows:

$$s_q(c) = \frac{W(c)}{n(c)} + C\sqrt{\frac{\log(N(q))}{n(c)}} + \frac{H(q,c)}{n(c)}.$$

$H(q,c)$ is a heuristic function for valuating move c in state q. The formula above can be adapted in order to take into account RAVE values as in Eq. 2.

A priori evaluation of moves: There are two main forms of a priori evaluations of moves, cumulated in best implementations:

- **Patterns**. In the case of Go, [12], [8], [2] propose the use of patterns extracted from a database D of professional games for building the function $index()$ of progressive widening (Eq. 3) or the function $H()$ of progressive unpruning (Eq. 4). Complex and essentially empirical formula have been derived for this; they work roughly as follows for estimating the value of a move:
  - find the biggest pattern, centered on this move, which appears in D;
  - the empirical probability p_1 for this pattern to be played in D (the confidence of this pattern, in the usual database terminology);
  - the frequency p_2 of this pattern in D (the support of the pattern, in the usual database terminology, i.e. the number of times the move was played divided by the size of D);
  - the heuristic value is then a linear compromise between p_1 and p_2 (p_1 being much stronger).

  The reader is referred to [12], [8], [2] for various formulas combining p_1 and p_2 into a $H(q,c)$. There's no widely accepted formula; for most important patterns (like e.g. the empty triangle, the wall, the keima and many others as described in [13]), it is worth tuning manually the coefficients by tedious experiments[14] - the usual general formulas don't reach the state of the art performance.

- **Tactical and strategical rules.** Important tactical or strategical rules are used for biasing the tree search, e.g. atari, extensions, line of influence (positive value for the moves located on the third line), line of death (negative value for the sides of the board); see [13] for more. Some papers also propose common fate graphs[15]; however, these common fate graphs have not been extensively used in successful MCTS implementations, except if one considers that the use of the notion of groups is a particular simple form of common fate graphs.

C. Side-informations extracted from simulations

MCTS is based on a huge number of simulations. The only information which is kept, from these simulations, is the number of won/lost games at each situation of the tree. It is somewhat natural to try to extract more informations from the simulations. The current main works around that are the owner information, the rapid action value estimates, and the criticality.

Owner information: "Owner information"[1] is the heuristic consisting in computing, for each location l of a board q, with which probability it belongs (at the end of simulations containing q) to the player whose turn it is to move. If the probability is close to $\frac{1}{3}$, the move is considered to be important; in CrazyStone, the probability of the move is increased in the Tree ($H(.)$ in Eq. 3). For example, in Fig. 1 extracted from [16], we see the probability for a move to be black/white at the end; this is the owner information, and the heuristic consists in playing more often, for white (resp. black), in locations which will be white with probability $\simeq 33\%$ (resp. 67%).

Rapid Action Value Estimates: Rapid Action Value Estimates (RAVE [5], see also [17], [18]) are a heuristic value for moves. The RAVE value for move m in situation q is as follows:

$$\text{rave}(q,m) = \frac{W(q,m)}{n(q,m)} \qquad (4)$$

if black (resp. white) is to play at q, with

- $W(q,m)$=number of won simulations where black (resp. white) plays first at m after situation q;
- $n(q,m)$=number of simulations where black (resp. white) plays first at m after situation q.

The important point, which makes the difference with the classical UCT values, is that black (resp. white) plays first (before white) at m *after* situation q, but not necessarily at situation q. RAVE values are updated at each simulation, and can only be used when a table of RAVE values is stocked in each node (this moderately extends the space complexity, as this is just storing one more value alongside the usual statistics). They provide a big improvement (see discussion in section II-E).

Criticality: Criticality has been specified in [16]. The idea is a generalization of the owner information. Whereas the owner information suggests playing in unsettled territory (see Fig. 1), the criticality suggests playing in locations highly correlated with the victory (the semeai in the upper left part of the Figure). Formally, the criticality of a location m in a situation q is defined as follows:

$$\text{criticality}_q(m) = \frac{v(m)}{N} - \frac{w(m)W + b(m)B}{N^2},$$

where:

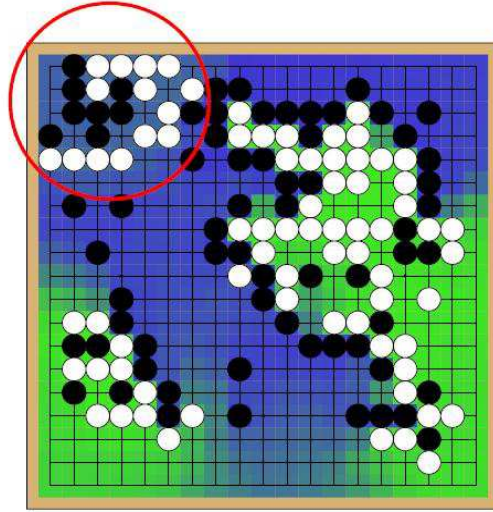- $v(m)$ is the number of simulations including situation q won by the owner of m;

Fig. 1. Plot of the "Owner" value: blue areas (dark in black and white) are expected to belong to black. We see that the owner value suggests playing around the frontier, in order to extend the domain owned by the player. The drawback is that in e.g. semeais the Monte-Carlo simulator is wrong (e.g. in the upper-left part, the colors show that the territory belongs to black, while, in fact, the black group is dead and the white lives). The figure and the semeai example on the upper-left corner are kindly provided by Rémi Coulom[16].

- N is the number of simulations including situation q;
- W (resp. B) is the number of simulations at q won by white (resp. black);
- $w(m)$ (resp. $b(m)$) is the number of simulations at q with m owned by white (resp. black).

We note that the formula is symmetric with regard to black and white. The first term increases for locations highly correlated with victory and the second term is a normalization; the formula is intuitively a covariance.

Criticality was tested without success in Zen (according to the author's post in the computer-Go mailing list) and provided a very little improvement in MoGo. This might be due to the redundancy with other heuristics (e.g. rapid action value estimates or Go expertise); nonetheless, criticality and variants of it are the only current tool for detecting semeais, a very important weakness of MCTS/UCT (see section II-D).

D. Expertise in the playouts

The design of the playouts is a very sensitive part of the algorithm. A small modification usually has a huge impact on the performance, in one way or the other. That's why it is very interesting to improve it. It is also the only way to correct some inherent problem of the UCT algorithm as for example in the case of nakade (see below) . However, except in some specific cases, the reasons explaining the success of a modification are still unknown. The current theory is that the modification should improve the level of the Monte-Carlo simulations while keeping the diversity and removing the undue bias. As this is very hard to predict, all the following modifications have been validated by numerous experiments.

Sequence-like Monte-Carlo (originating in MoGo): The main innovation of the early versions of MoGo was the design of the playouts [19], [7]. They pointed out that improving the strength of the playouts directly could lead to a decrease of performance for the overall algorithm. That is why whereas

previous works on the playouts focused on increasing the quality of the Monte-Carlo player as a standalone player, this work designed a Monte-Carlo from a very empirical point of view (accepting a modification of the playouts if the MCTS based on these playouts plays better, and not if the playout generator plays better). All strong algorithms now use "sequence-like" simulations, in which a move is highly correlated to the previous move. More precisely, a move is played in the immediate neighborhood (in 8 connectivity) of the last move if it matches a database of handcrafted patterns, which are reasonable for human experts. If there are several such moves, one of them is randomly chosen and played; if not, then a randomly chosen move in the board is played (Alg. 1).

A crucial property of the playouts is that it should be *balanced* (i.e. equilibrated between black and white); this is much more important than having a strong playout generator. Ultimately, if the players play exactly equally well in all situations, then the playouts are a perfect evaluation function. The weaknesses of Monte-Carlo Tree Search (detailed later) are in situations in which the simulations are not equilibrated; for example, in semeais, Monte-Carlo may give around 50 % of probability of winning the semeai to each player, even if the semeai is a clear win for one of the players. This idea of balancing the simulations was developed in [19], [7]; there's a recent effort in automatizing this [20], [21], with not yet good results on big boards.

A counterpart to "sequence-like" simulations is the use of the "fill board" modifications, a kind of "Tenuki"-rule, which switches to another (empty) part of the goban and therefore prevents the loss of diversity in the simulations. This modification is described in detail in [13]. This is somehow controversial, as this rule (i) brings very big improvements in MoGo (ii) is not yet tested in many implementations (iii) is only efficient for long enough time settings (and can be

detrimental for short time settings).

Algorithm 1 Algorithm for choosing a move in MC simulations. The patterns used for "sequential" moves are described in [19]. The implementation is a bit more complicated than that, with some levels more, as well as in Fuego; a significantly implementation is the one used in CrazyStone (and probably Zen as well), which updates a complete table of probability for all moves.

---

    if the last move is an atari, **then**
        Save the stones which are in atari if possible (this is checked by liberty count).
    else
        if there is an empty location among the 8 locations around the last move which matches a pattern **then**
            Sequential move: play randomly uniformly in one of these locations.
        else
            if there is a legal move **then**
                Legal move: Play randomly a legal move
            else
                Return pass.
            end if
        end if
    end if

Nakade: A nakade is a situation in which a surrounded group has a single large internal, enclosed space in which the player won't be able to establish two eyes if the opponent plays correctly. Most of current go programs don't estimate properly this kind of situation. It is not evaluated by the tree because no player wants to play there (the Monte-Carlo evaluation is the same unless many moves are played in the nakade) and it is not correctly handled by the playouts without the addition of a specific rule. This situation is a good example of case where the addition of expert knowledge in the playouts can contribute to solve the problem. In MoGo, the rule consists in playing at the center of three empty locations surrounded by opponent stones. This rule is called in Algorithm 1 before other rules. It is a simple and efficient modification but it does not work in all cases of nakade. Examples of nakade solved and not solved by this method are given in Fig. 2. To the best of our knowledge, the detailed implementation of Nakade rules in other programs is not known in details; in Fuego, there is a simple rule of moving single stone selfataries to the adjacent point.

Semeai: Semeai are situations where two opponent groups can't live without one killing the other or being in seki with each other. It happens often in Go game and the result of the semeai (which group is alive at the end) has a huge impact on the score. That is why it is really important for a Go program to handle such situations correctly. However, it often requires a very long sequence of complicated moves to determine the result, even the order of the moves can matter. In this case, the tree is often not deep enough to solve the semeai. There is for the moment no good solution to handle perfectly those situations but some modifications of the Monte-Carlo simulations can help. For example, we introduce in MoGo the approach move. This is described on the left of Fig. 3, black should play in B before playing in A for killing white; this is an approach move. In MoGo, we improve the behavior of Monte Carlo simulations by replacing self atari moves by a connection to an other group when this is possible. More details are given in [13] . However, as shown on the right of Fig. 3, there are still simple semeai not correctly handled by MoGo.

Two-liberties rules: A lot of rules in the playouts are based on the number of liberties of a group. The basic rules, like avoiding atari and killing group, are based on groups with one liberty. By creating rules for groups with two liberties, we can cover a larger number of situations and improve the quality of the simulations. For example, the two-liberties killing rule is "if when removing one of the liberties, the group has no way to escape (no move can improve the number of liberties), then play it" and the corresponding two-liberties escape rule "if one group has two liberties and the opponent can play a two liberties killing move, then play a move that prevents it". Those rules are only examples. They are illustrated on Fig. 4; see also [22]. Similar rules are implemented in MoGo, ManyFaces, and Fuego.

Other rules: Other classical rules consist in avoiding big self-atari (but this can be complicated for nakade situations); a detailed analysis of several rules (captures, extensions, distance to the borders, ladder atari and ko atari) and their relative weights can be found in [8]. Each program has his own expert rules and they appear to be very implementation-dependent. A rule that works for one program doesn't necessarily work for another. Furthermore, when a program is modified, the rules might not work any more or at least not with the same parameters. Therefore, using expert knowledge in the playouts is very time-consuming in term of experiments. However, it is worth doing it as we can see for example with the program Zen: it is currently ranked 2D on KGS and, according to its creator, possesses a lot of hard coded Go knowledge in its playouts.

E. Differences between programs

We here briefly survey the differences between the four computer-Go programs involved in the games against humans. There are not a lot of public informations on Zen; Zen is according to his author's post on the computer-Go mailing list based on papers describing CrazyStone [8], with a lot of expert knowledge added.

Differences in the playouts: All implementations use sequence-like Monte-Carlo based on local patterns. The Nakade modification described above is used in MoGo and provides a big improvement in particular in 9x9. Fill board is used in MoGo but not in other implementations.

Differences in the bias for the bandit part: There is three main modifications that can be applied to the bandit part of the algorithm: (i) Rapid Action Value Estimates [5], (ii) a database of patterns (as in [2], [8]), (iii) expert knowledge (patterns, tactical and strategical rules detailed in [13]). The CrazyStone algorithm in [8] handles (ii) and (iii) in a unified framework.

The use of those modifications in the different programs is presented in Table I.

(a)   (b)   (d)
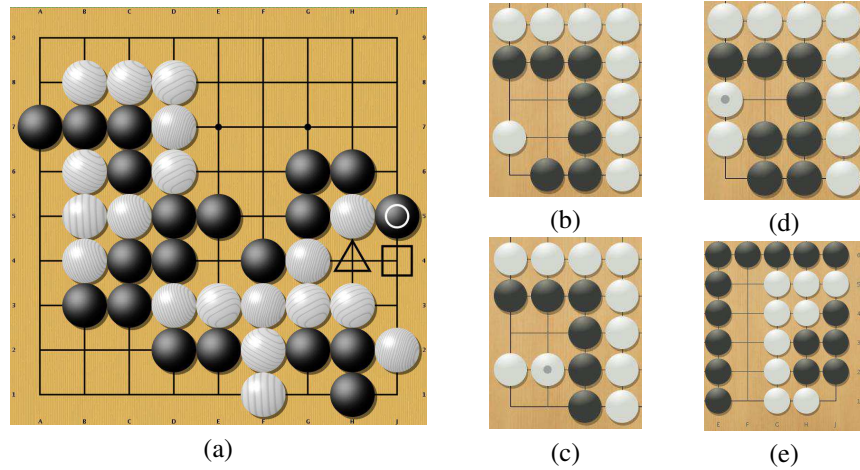   (c)   (e)

Fig. 2.   In Figure (a) (a real game played and lost by MoGo), MoGo (white) without specific modification for the nakade chooses H4 (triangle); black plays J4 (square) and the group F1 is dead (MoGo loses). The right move is J4 (square); this move is chosen by MoGo after the modification presented in section II-D. Examples (b), (c) and (d) are other similar examples in which MoGo (as black, without the nakade module) evaluates the situation poorly and doesn't realize that his group is dead. The modification solves the problem for (a), (b), (c) and (d). (e) is an example of more complicated nakade, which is not solved by MoGo (the white group won't be able to make two eyes after capturing the black stones and therefore will die).
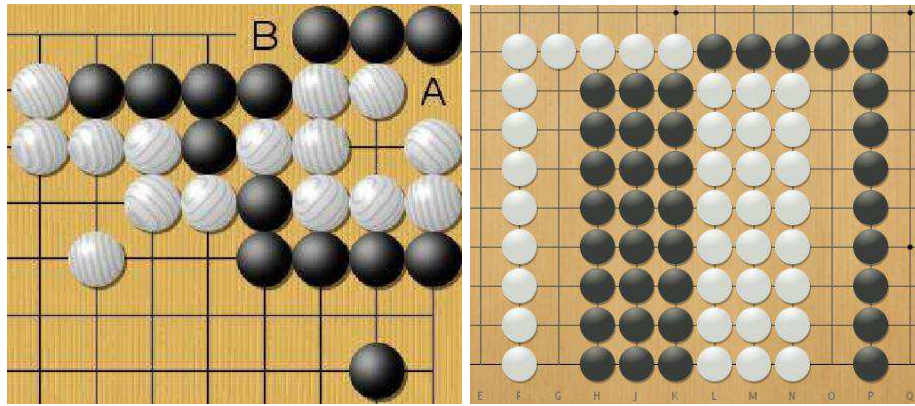

Fig. 3.   Left: Example of situation which is poorly estimated without approach moves. Black should play B before playing A for killing the white group and live. Right: situation which is not handled by the "approach moves" modification.
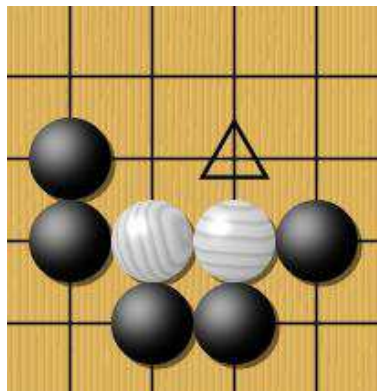

Fig. 4.   This figure illustrates the two-liberties killing rule: if it is black turn, the rule activates and black plays on the triangle. It also illustrates the two-liberties escape rule: if it is white turn, the rule activates and white also play on the triangle to prevent black from playing it

Remarks:

- in MoGo the weight of (i) in 19x19 had to be reduced when databases of patterns (providing offline heuristic values for moves) have been added; this suggests that RAVE values are a very good heuristic (also for other games [11]), but their weight should be reduced when other heuristics are available.
- (ii) is removed in 9x9 for optimal performance.
- (ii) is seemingly more developed in ManyFaces, MoGo and Zen than in Fuego; (iii) is more developed in Many-

	MoGo	ManyFaces	Fuego	Zen
Bandit part				
RAVE	X	X	X	X
Learnt patterns	X			X
Expert Knowledge	X	XX	X	XX
Playouts				
Fill board	X			
Sequence-like patterns	X	X	X	X
Learnt patterns				X
Expert Knowledge	X	XX	X	XX
Full board probabilistic model (e.g. low liberty rule)				X

TABLE I
DIFFERENT MODIFICATIONS OF THE BANDIT FORMULA USED IN EACH PROGRAM (TOP) AND OF THE PLAYOUTS (BOTTOM). THE XX MEANS THAT THE AUTHORS EMPHASIZE A BIG WORK ON THIS PART. LEARNT PATTERNS REFER TO BIG DATABASES OF PATTERNS AUTOMATICALLY LEARNT FROM GAMES AND NOT TO HANDCRAFTED PATTERNS. IN ZEN, AS IN CRAZYSTONE, A FULL-BOARD PROBABILISTIC MODEL UPDATES THE PROBABILITY OF ALL LOCATIONS IN THE BOARD AT EACH MOVE.

Faces and Zen than in MoGo and in Fuego.

- (iii) is always efficient, whenever RAVE values or databases of patterns are present, and this suggests that databases are a great tool as they need little development and expertise, but databases are not enough to catch the tactical knowledge of experts.

Other differences: In 9x9 MoGo uses a huge automatically built opening book. As shown in [23], this provides a big improvement; also it saves up a lot of time as many moves are immediately played by the opening book thanks to permutations/rotations/symmetries; however some bad moves are sometimes introduced in this automatically generated opening book and corrections by experts analyzing games are very efficient. Zen and Fuego use handcrafted 9x9 opening books, but Fuego contains also some weak moves in the opening book as shown later.

All implementations use a multi-core parallelization (each core performs simulations independently of the others, but all cores write their results in the same tree). Some of them use lock-free hashtables for improved performance [24], [25]. MoGo, ManyFaces and Fuego all use also message-passing parallelization, *i.e.* can benefit from the computational power of clusters. This is known as much more efficient in 19x19 than in 9x9. See [26], [27], [28], [29] for more informations on the parallelization. Later than the IEEE-Fuzz 2009 event, Zen has been equipped with the same message-passing parallelization.

III. RESULTS AND COMMENTS

This section presents the games between humans and computers (Many Faces of Go, MoGo, Fuego, Zen), in FUZZ-IEEE 2009. The overall results are presented in Table II and discussed in the rest of this paper. The hardware used in the competition is presented in Table III.

All comments around the game of Go are given by experts: Chun-Hsun Chou 9P, Shen-Su Chang 6D, Shi-Jim Yen 6D,and Shang-Rong Tsai 6D. The ability of MCTS for fights is illustrated in section III-A. The 9x9 opening books are discussed in section III-B. The weaknesses in corners are discussed in

section III-C. The aggressivity of the programs is discussed in section III-D. The weakness in semeais and in seki, probably the current most important weakness, is discussed in section III-E.

A. Ability for fights

MCTS/UCT algorithms are known for being very strong in killing. This is illustrated in the game won by Zen as white against Shen-Su Chang 6D (Fig. 5, left).

B. 9x9 opening books

We distinguish below handcrafted opening books and self-built opening books.

1) Handcrafted opening books: Fuego's opening book is handcrafted; nonetheless, Fuego plays a bad move very early, namely the "kosumi" (move 3, Fig. 6, left). This move was supposed to be good with a komi of 6.5 but is not aggressive enough with a komi of 7.5. Kosumis (diagonal move), according to [23], are very often bad moves in the beginning of a 9x9 game. On the other hand, Fuego won as white with good opening moves (only 3 moves in the opening book), see Fig. 6 (right).

Opening moves by Zen were all good in 9x9 according to experts; Zen won one game as black and one game as white against Shen-Su Chang 6D (Fig. 5). There were very few moves in the opening book.

2) Self-built opening books: MoGo has a huge opening book built on a cluster [23]. However, the two openings (black and white) contained mistakes which were exploited by Chun-Hsun Chou 9P, who won both as black and as white against MoGo (Figure 7).

C. Weaknesses in corners

It is often said that MCTS algorithms have a bad strategy, as they try to develop a big moyo instead of focusing in corners; this has been related to cosmic go. However, it is also often said that computers have a strong sense of "aji", which is a

No	Rank	Date	Setup	White	Black	Result
			Scheduled games			
1	9P	08/21/2009	19x19 H7	**Mr. Chou**	Many Faces of Go	W+Res.
2	6D	08/21/2009	19x19 H4	**Mr. Chang**	MoGo	W+Res.
3	9P	08/21/2009	9x9	MoGo	**Mr. Chou**	B+Res.
4	6D	08/21/2009	9x9	Many Faces of Go	**Mr. Chang**	B+6.5
5	9P	08/21/2009	9x9	**Mr. Chou**	MoGo	W+Res.
6	6D	08/21/2009	9x9	**Mr. Chang**	Many Faces of Go	W+Res.
7	9P	08/22/2009	9x9	**Fuego**	Mr. Chou	W+2.5
8	6D	08/22/2009	9x9	**Zen**	Mr. Chang	W+Res.
9	9P	08/22/2009	9x9	**Mr. Chou**	Fuego	W+Res.
10	6D	08/22/2009	9x9	Mr. Chang	**Zen**	B+Res.
11	9P	08/22/2009	19x19 H7	**Mr. Chou**	Zen	W+Res.
12	6D	08/22/2009	19x19 H4	Mr. Chang	**Fuego**	B+Res.

TABLE II

OVERVIEW OF RESULTS; GAMES PLAYED DURING FUZZ-IEEE AT JEJU ISLAND, KOREA.

Program	Machine
MoGo	Supercomputer "Huygens" with 20 nodes of 32 cores (640 cores). Linux.
Fuego	Ten 8-cores nodes. Each node has two quad core Xeon E5462 @ 2.80GHz processors and 32GB of mainstore. 20Gbps Infinite band network between the nodes.
Many Faces of Go	4 nodes: 32 cores, with a total of 64 GB of RAM Each node has 2 x quad core Intel Xeon (x5460) 3.16 GHz 16 GB of RAM; Microsoft Windows HPC.
Zen	Mac Pro with 8 core processors (Quad-Core Intel Xeon 2.26GHz x2).

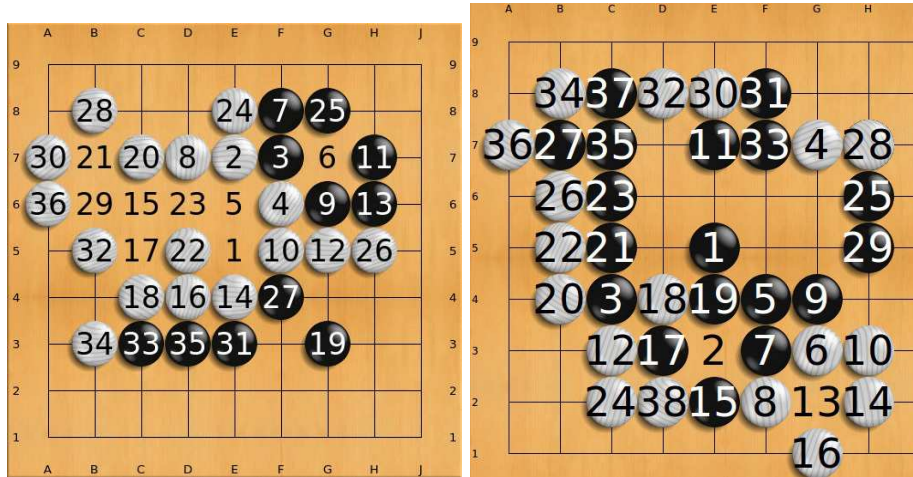TABLE III

HARDWARE USED BY THE COMPUTERS.

Fig. 5. Left: game won by Zen as white against Shen-Su Chang 6D; black made a mistake (move 29 at B6 instead of B4), immediately punished by White killing E5. Right: game won by Zen as black against Shen-Su Chang 6D (black plays E3 and wins). In both cases, Zen had good opening moves. As black, Zen had a big moyo.

deep concept - the influence that one might expect from his dead stones. In 9x9, having a big moyo can be efficient, as in e.g. Fig. 5 (right) where Zen, with a big moyo only, wins the game as black. On the other hand, in 19x19, protecting the moyo is very difficult, and it is therefore often preferable to take care of corners.

For example, ManyFaces lost against Chun-Hsun Chou 9P in spite of handicap 7 with 4 corners taken by the pro, and then the moyo also invaded (N15, N11 at least can have access to the moyo, Fig. 8, left). Zen and MoGo lost against Chun-Hsun Chou 9P with the same settings. Shen-Su Chang won his

games with H4, except the one against Fuego (Fig. 8, right) in which he made a mistake and could not invade the moyo.

D. Programs are too aggressive

It is often said that MCTS programs are quite efficient for killing, but that they are too confident in their ability to kill. This is confirmed in *e.g.* Fig. 9.

E. Weaknesses in semeais and sekis

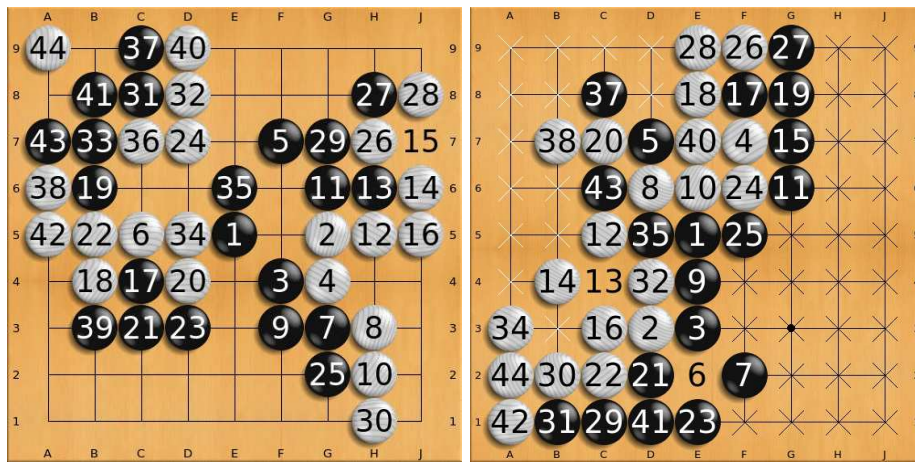MCTS programs are known for being weak in semeais; this is also true for sekis.

Fig. 6. Left: game won as white by Chun-Hsun Chou 9P against Fuego. Move 3 (handcrafted move from the opening book) is a kosumi and is considered to be bad in early 9x9 game. Right: game won as white by Fuego against Chun-Hsun Chou 9P; according to experts the opening by Fuego was good. 33 was in A2, 36 in A1, 39 in A2.
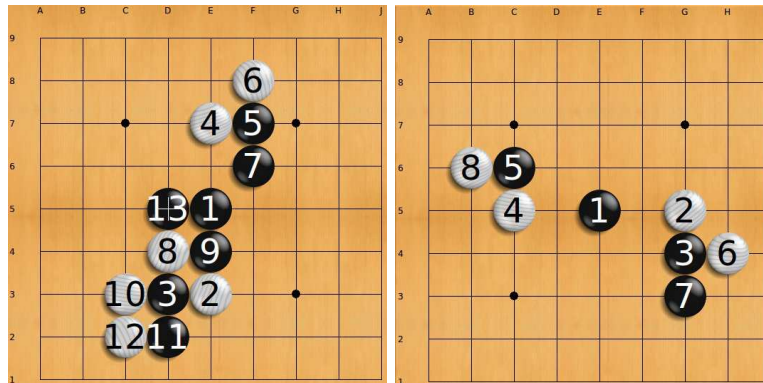

Fig. 7. Situation at the end of MoGo's opening book as white (left) and black (right). According to Chun-Hsun Chou 9P, the situation at the end of the opening book (the two situations presented here) was bad. Left: we could not conclude to which move should be corrected - no really bad move, but at that point in the game, the pro considers that the situation is lost - maybe the opening by black is just too well known and, due to the high 7.5 komi, human can find the correct answer for white. Right: move 7 is bad.

Fig. 8. Left: ManyFaces was black, handicap 7, against Chun-Hsun Chou 9P and lost with the 4 corners taken by the pro; the pro also invaded the moyo. Right: Fuego was black, H4, against Shen-Su Chang 6D. White was in very good situation on the picture, but played a bad move, L19, instead of L15 which would invade the moyo and win. Fuego could keep the moyo and therefore won.

Figure 6, where Fuego made a mistake in the opening, is also an example of semeai, as B8 could only live by killing A5; however there are much more liberties for white which easily kills B8 by nakade.

Figure 10 (left) shows an example in which a seki was used by the human for winning as black against ManyFaces in 9x9.
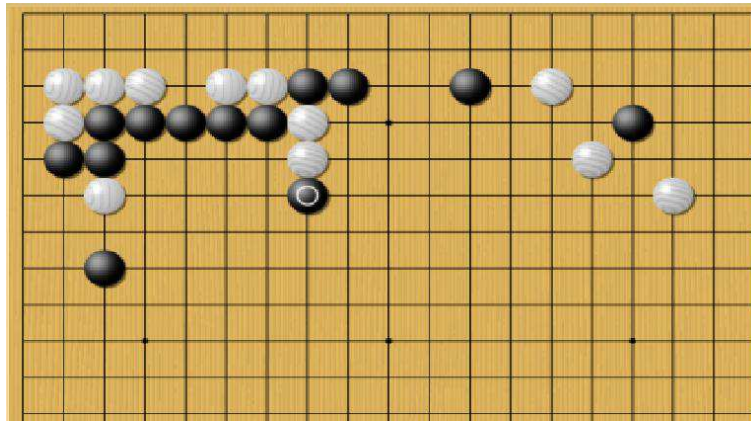
Fig. 9.   MoGo is playing as black against Shen-Su Chang with H4. MoGo plays the circled black stone, trying to kill the two white stones; this was impossible, and as MoGo was keeping trying to kill white, it lost the upper center part of the goban and lost.

Fig. 10 (right) shows an example in which the human won by semeai against ManyFaces, also in 9x9.

Figure 11 (left) shows that Zen lost a semeai in the upper-right corner, and Fig. 11 (right) shows that MoGo lost a semeai in the upper right corner, and only understood it when the situation was completely clarified by the pro.

IV. CONCLUSIONS

During FUZZ-IEEE in Jeju Island, Fuego won the first ever victory of a computer against a top pro in 9x9 with komi 7.5 as white. Komi should be smaller according to the experts, if we want the setting to be fair; maybe 6.5 makes the game more equilibrated; this would have a big impact on the opening book. The 9x9 opening books could easily be made stronger with the help of high-level players; current handcrafted opening books are too short, and automatically built opening books contain errors. Humans suggest 13x13 as a future challenge, and also consider that ensuring a win with handicap 7, from the current strength of programs, should be possible if they make less mistakes in the corners early on. One possible way of dealing with this is to include a big joseki database; yet, if nobody has succeeded yet in doing so, one can think that this is non trivial.

Technically speaking, semeais and sekis are still poorly analyzed by MCTS, in spite of many research on criticality [16] and introduction of tactical solvers [30]. Also, MCTS programs are too much interested in the moyo and neglect the corners. There's no sharing of information between one branch of the tree and another, and no use of machine learning for automatically adapting the playouts.

It is interesting to point out the tools that were used also in other successful applications of MCTS/UCT. UCT is the most classical formula used in one-player applications ([31], [10] for non-linear optimization and active learning respectively), but there are other bandit rules also ([32] for optimization on grammars, using max-bandits). There are plenty of applications to other games; for Havannah (a game which is specially difficult for computers and for which the RAVE heuristic is highly efficient [11]), general game playing [6]; multiplayer games [33] and in particular multiplayer Go [34] and Settlers of Catan [35]. It has been shown that for sudden-death games there are fruitful possible modifications [36], and for partially observable games like phantom-Go heuristic adaptations have been proposed [37], [38] - a principled application to the partially observable case has been proposed in [10] but it is deeply limited to one-player applications.

REFERENCES

[1] Coulom, R.: 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)

[2] Chaslot, G., Winands, M., Uiterwijk, J., van den Herik, H., Bouzy, B.: Progressive strategies for monte-carlo tree search. In Wang, P., et al., eds.: Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007), World Scientific Publishing Co. Pte. Ltd. (2007) 655–661

[3] Kocsis, L., Szepesvari, C.: Bandit-based monte-carlo planning. In: ECML'06. (2006) 282–293

[4] Lai, T., Robbins, H.: Asymptotically efficient adaptive allocation rules. Advances in Applied Mathematics **6** (1985) 4–22

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

[6] Sharma, S., Kobti, Z., Goodwin, S.: Knowledge generation for improving simulations in uct for general game playing. In: AI '08: Proceedings of the 21st Australasian Joint Conference on Artificial Intelligence, Berlin, Heidelberg, Springer-Verlag (2008) 49–55

[7] Wang, Y., Gelly, S.: Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In: IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii. (2007) 175–182
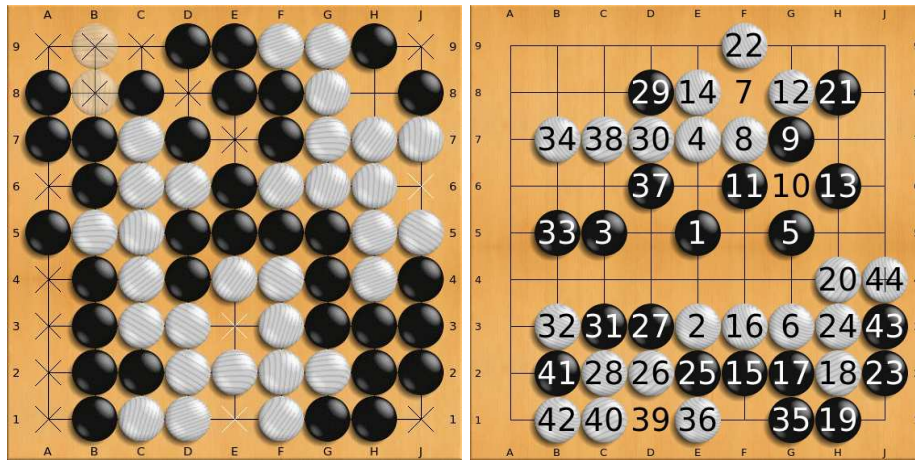
Fig. 10. Left: ManyFaces plays as white and has two groups alive; nonetheless, black wins thanks to the seki in the upper right corner (the two black stones are alive). Right: ManyFaces plays as black and loses by semeai in the lower part. In both cases, ManyFaces was playing against Shen-Su Chang 6D.
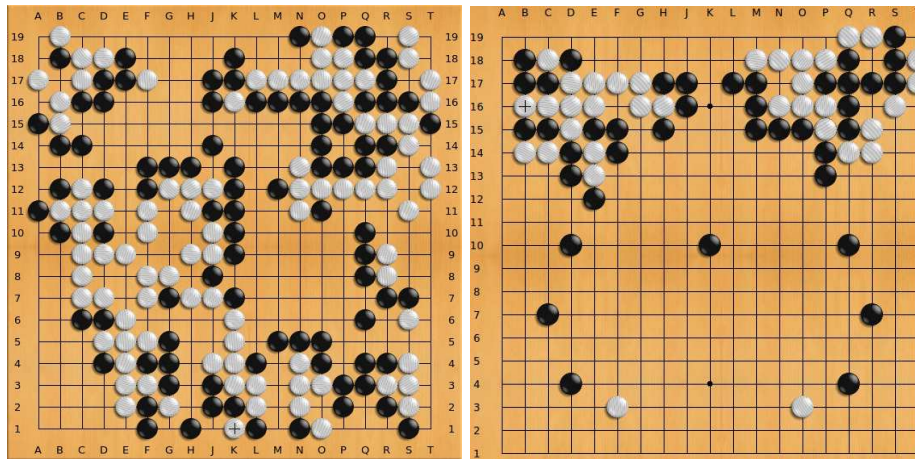

Fig. 11. Left: Zen was black, handicap 7, against Chun-Hsun Chou 9P and lost with 3 corners taken by the pro (white stones on the bottom right are dead); the pro also invaded the moyo. The situation was good for black at move 65 but after that Zen made some mistakes by not defending the corners which caused the loss. Right: MoGo was black, H7, against Chun-Hsun Chou 9P; as in other 19x19 games, the pro takes most of the corners, invades the moyo and wins. In this situation, MoGo played F13 (which is of little interest as the white group E13-E14 is in dead ladder) and the pro played K4 which invades the moyo. MoGo could have prevented the invasion by playing K4 itself instead of F13.

[8] Coulom, R.: Computing elo ratings of move patterns in the game of go. In: Computer Games Workshop, Amsterdam, The Netherlands. (2007)

[9] Wang, Y., Audibert, J.Y., Munos, R.: Algorithms for infinitely many-armed bandits. In: Advances in Neural Information Processing Systems. Volume 21. (2008)

[10] Rolet, P., Sebag, M., Teytaud, O.: Optimal active learning through billiards and upper confidence trees in continous domains. In: Proceedings of the ECML conference. (2009)

[11] Teytaud, F., Teytaud, O.: Creating an Upper-Confidence-Tree program for Havannah. In: ACG 12, Pamplona Espagne (2009)

[12] Bouzy, B., Chaslot, G.: Bayesian generation and integration of k-nearest-neighbor patterns for 19x19 go. In: G. Kendall and Simon Lucas, editors, IEEE 2005 Symposium on Computational Intelligence in Games, Colchester, UK. (2005) 176–181

[13] Chaslot, G., Fiter, C., Hoock, J.B., Rimmel, A., Teytaud, O.: Adding expert knowledge and exploration in Monte-Carlo Tree Search. In: Advances in Computer Games, Pamplona Espagne, Springer (2009)

[14] Lee, C.S., Wang, M.H., Chaslot, G., Hoock, J.B., Rimmel, A., Teytaud, O., Tsai, S.R., Hsu, S.C., Hong, T.P.: The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. IEEE Transactions on Computational Intelligence and AI in games (2009) 73–89

[15] Ralaivola, L., Wu, L., Baldi, P.: Svm and pattern-enriched common fate graphs for the game of go. In: ESANN. (2005) 485–490

[16] Coulom, R.: Criticality: a monte-carlo heuristic for go programs (2009) Invited talk at the University of Electro-Communications, Tokyo, Japan.

[17] Bruegmann, B.: Monte carlo go (1993)

[18] Bouzy, B., Helmstetter, B.: Monte-Carlo go developments (2003)

[19] Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with patterns in monte-carlo go. Rapport de recherche INRIA RR-6062, Inria (2006)

[20] Silver, D., Tesauro, G.: Monte-carlo simulation balancing. In: Proceedings of the International Conference on Machine Learning. (2009) 119

[21] Huang, S.C., Coulom, R., Lin, S.S.: Monte-carlo simulation balancing in practice. In: Proceedings of the International Conference on Computers and Games. (2010)

[22] Cazenave, T.: Playing the right atari. ICGA Journal **30** (2007) 35–42

[23] Audouard, P., Chaslot, G., Hoock, J.B., Perez, J., Rimmel, A., Teytaud, O.: Grid coevolution for adaptive simulations; application to the building of opening books in the game of go. In: Proceedings of EvoGames. (2009)

[24] Coulom, R.: Lockless hash table and other parallel search ideas (2008) Post on the computer-go mailing list.

[25] Enzenberger, M., Müller, M.: A lock-free multithreaded Monte-Carlo tree search algorithm. In: Proceedings of Advances in Computer Games 12. (2009)

[26] Gelly, S., Hoock, J.B., Rimmel, A., Teytaud, O., Kalemkarian, Y.: The parallelization of monte-carlo planning. In: Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO 2008). (2008) 198–203 To appear.

[27] Chaslot, G., Winands, M., van den Herik, H.: Parallel Monte-Carlo Tree

Search. In: Proceedings of the Conference on Computers and Games 2008 (CG 2008). (2008)

[28] Cazenave, T., Jouandeau, N.: On the parallelization of UCT. In: Proceedings of CGW07. (2007) 93–101

[29] Kato, H., Takeuchi, I.: Parallel monte-carlo tree search with simulation servers. In: 13th Game Programming Workshop (GPW-08). (2008)

[30] Cazenave, T., Helmstetter, B.: Combining tactical search and monte-carlo in the game of go. IEEE CIG 2005 (2005) 171–175

[31] Auger, A., Teytaud, O.: Continuous lunches are free plus the design of optimal optimization algorithms. Algorithmica (2009)

[32] de Mesmay, F., Rimmel, A., Voronenko, Y., Püschel, M.: Bandit-based optimization on graphs with application to library performance tuning. [39] 92

[33] Sturtevant, N.R.: An analysis of UCT in multi-player games. [40] 37–49

[34] Cazenave, T.: Multi-player go. [40] 50–59

[35] Szita, I., Chaslot, G., Spronck, P.: Monte carlo tree search in settlers of catan. In: Proceedings of 12th Advances in Computer Games Conference (ACG12). (2009)

[36] Winands, M.H.M., Björnsson, Y., Saito, J.T.: Monte-carlo tree search solver. [40] 25–36

[37] Cazenave, T.: A phantom-go program. In van den Herik, H.J., chin Hsu, S., sheng Hsu, T., Donkers, H.H.L.M., eds.: ACG. Volume 4250 of Lecture Notes in Computer Science., Springer (2006) 120–125

[38] Cazenave, T., Borsboom, J.: Golois wins phantom go tournament. ICGA Journal **30** (2007) 165–166

[39] Danyluk, A.P., Bottou, L., Littman, M.L., eds.: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009. In Danyluk, A.P., Bottou, L., Littman, M.L., eds.: ICML. Volume 382 of ACM International Conference Proceeding Series., ACM (2009)

[40] van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M., eds.: Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 - October 1, 2008. Proceedings. In van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M., eds.: Computers and Games. Volume 5131 of Lecture Notes in Computer Science., Springer (2008)