HAL
open science

The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments

Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, Tzung-Pei Hong

To cite this version:

Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, et al.. The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. IEEE Transactions on Computational Intelligence and AI in games, 2009. ⟨inria-00369786⟩

HAL Id: inria-00369786

https://inria.hal.science/inria-00369786v1

Submitted on 21 Mar 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Autorisation HAL

HAL Authorization

The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments

Chang-Shing Lee[*], Mei-Hui Wang[*], Guillaume Chaslot[**],
Jean-Baptiste Hoock[***], Arpad Rimmel[***], Olivier Teytaud[***],
Shang-Rong Tsai[****], Shun-Chin Hsu[****], and Tzung-Pei Hong[*****]

[*]Dept. of Computer Science and Information Engineering, National University of Tainan, Taiwan
[**]Dept. of Computer Science, University of Maastricht, Netherlands
[***]TAO, Lri, Univ. Paris-Sud, Inria Saclay-IDF, UMR Cnrs 8623
Bât 490, Université Paris-Sud F91405 Orsay, France
[****]Dept. of Information Management, Chang Jung Christian University, Taiwan
[*****]Dept. of Computer Science and Information Engineering, National University of Kaohsiung, Taiwan

E-mail: leecs@mail.nutn.edu.tw, olivier.teytaud@inria.fr

Abstract

In order to promote computer Go and stimulate further development and research in the field, the event activities, "*Computational Intelligence Forum*" and "*World 9×9 Computer Go Championship*," were held in Taiwan. This study focuses on the invited games played in the tournament, "Taiwanese Go players versus the computer program MoGo," held at National University of Tainan (NUTN). Several Taiwanese Go players, including one 9-Dan professional Go player and eight amateur Go players, were invited by NUTN to play against MoGo from August 26 to October 4, 2008. The MoGo program combines *All Moves As First* (AMAF)/*Rapid Action Value Estimation* (RAVE) values, online "UCT-like" values, offline values extracted from databases, and expert rules. Additionally, four properties of MoGo are analyzed including: (1) the weakness in corners, (2) the scaling over time, (3) the behavior in handicap games, and (4) the main strength of MoGo in contact fights. The results reveal that MoGo can reach the level of 3 Dan with, (1) good skills for fights, (2) weaknesses in corners, in particular for "semeai" situations, and (3) weaknesses in favorable situations such as handicap games. It is hoped that the advances in artificial intelligence and computational power will enable considerable progress in the field of computer Go, with the aim of achieving the same levels as computer chess or Chinese chess in the future.

Keywords: Computational Intelligence, Computer Go, Game, MoGo, Monte-Carlo Tree Search

I. Introduction

Games provide competitive dynamic environments that are ideal for testing computational intelligence theories, architectures, and algorithms [1]. Many studies have identified the developments, challenges, and opportunities for applying computational intelligence methods to games [1][2]. Additionally, Go remains an excellent challenge for computer science research; however, Monte Carlo methods have very recently shown significant promise, especially for small versions of the game such as 9×9 games. Therefore, the Upper-Confidence Tree (UCT) Monte Carlo has considerable potential for application to other games such as Hex, Amazons, and even Shogi [2][39]. Schaeffer and Herik [38][39] noted that work on computer games has resulted in advances in numerous computing areas. Many ideas that developed through game-tree search have been applied to other algorithms. For example, the UCT Monte Carlo algorithm may have important applications to control Non-player Characters (NPCs) in video games such as Quake [1][2]. Moreover, many studies have applied artificial intelligence (AI) and evolutionary computation to games. For instance, Chellapilla and Fogel [3][4] developed an expert program that plays checkers without using human expertise or expert knowledge. Messerschmidt and Engelbrecht [5] developed a competitive learning approach to playing games. Werf *et al.* [6] presented a search-based approach for playing Go on small boards. Bouzy and Cazenave [7] presented an AI-oriented survey of computer Go. Togelius *et al.* [8] applied computational intelligence to racing games. Chen [9] proposed a strategy that maximizes the chance of winning when searching Go game trees. Cutumisu *et al.* [41] advocated the development of adaptive programming as an alternative to current constructive programming techniques, as well as the application of adaptive programming to many domains. Carbonaro *et al.* [42] proposed an interactive story authoring technology that offers students an opportunity to successfully construct interactive game stories. Zahavi *et al.* [40] proposed a new dual search algorithm to improve the chance of reaching a goal fast, meaning that the algorithm does not necessarily visit all states on a solution path.

In chess, humans now need a handicap (in favor of the human) to have a chance of winning against top-level programs. In Go, humans are still heavily favored to win. For example, in 1998 Muller won despite 29 handicap stones against "*Many Faces Of Go*" [11]. Computer Go has, however, made

considerable progress in recent years. Programs are currently competitive at the professional level in 9×9 Go, and MoGo has won with an advantage of "only" 9 handicap stones against top-level human players in 19×19 Go; additionally, CrazyStone won with handicaps of 8 and 7 stones against Kaori Aoba, a Japanese 4th Dan Pro (4P). To strengthen computer Go programs and advocate research, development and application of computer games' related fields, Chang Jung Christian University (CJCU), National University of Tainan (NUTN), and the Taiwanese Association for Artificial Intelligence (TAAI) hosted the "*2008 Computational Intelligence Forum and World 9×9 Computer Go Championship*." This event, held in Taiwan, was to fulfill the purpose of "Enjoying learning through playing computer Go." Event activities were the "*Computational Intelligence Forum*" and "*World 9×9 Computer Go Championship*." The championship was divided into two sections. Section A, which comprised computer program competitions, was won by MoGo which was undefeated. Section B was human *versus* computer competitions.

The recent rapid improvements to computer Go are mainly due to the development and application of the Monte Carlo Tree Search (MCTS) algorithm. The MCTS algorithm and associated algorithms have been applied to computer Go. On the other hand, they have also been applied to several other games, such as Settlers of Catan [12] and Texas Hold'em poker [13], which show that when the branching factor after obvious pruning remains too large, or when no good handcrafted evaluation function exists, algorithms based on the "bandit principle" (compromises between exploration and exploitation) are efficient. Real-time strategy games, which are games with incomplete information, have also been tested [14]. Algorithms using the bandit principle have also been applied to, say, clinical trials with the MCTS algorithm [15], non-linear robust optimization using UCT [16], news selection with a technique based on mixing bandit and change-point detection, which was ranked first in the Nips/Pascal "Online Trading of Exploration and Exploitation Challenge 2006" [17], and optimal sailing using simulations and a tree developed at the point of interest [18]. The new algorithms are remarkably scalable and have considerable computational power in the 19×19 Go game, as they can use supercomputers. This study focuses on the invited games played in the tournament, "Taiwanese Go players *versus* the computer program, MoGo," held at NUTN, Taiwan. Several Taiwanese Go players, including one 9P Go player and eight amateur

players, ranging from 1 Dan (1D) to 6D, were invited by NUTN to play against MoGo from August 26 to October 4, 2008. In particular, Jun-Xun Zhou, a 9P Go player, played 9×9 and 19×19 games against MoGo running on a supercomputer with 800 CPUs, through the Kiseido Go Server (KGS) on September 27, 2008. Zhou, the strongest Go player in Taiwan, won the 2007 World LG Cup. MoGo lost three games to Zhou, including two 9×9 games and one 19×19 game with 7 handicap stones. MoGo had a very favorable situation in the first 9×9 game, but made a significant mistake and lost. The invited eight amateur Go players included a retired professor of NUTN (C. W. Dong, 70 years old, 5D), a Chief Information Officer (CIO) of a software company (C. S. Chang, 50 years old, 6D), the Chief Referee of this championship (S. R. Tsai, 55 years old, 6D), two teachers of Tainan's Go Association (B. S. Luoh, 45 years old, 6D; and W. T. Yu, 50 years old, 3D), and three child members of Tainan's Go Association (Y. S. Huang, 12 years old, 4D; Y. X. Wang, 11 years old, 3D; and S. Y. Tang, 10 years old, 2D). Tournament results indicate that MoGo was roughly 2–3D for 19×19 games and roughly 1P professional for 9×9 games on the Taiwanese scale.

The remainder of this study is organized as follows. Section II briefly describes recent advances in computer Go. The mechanism of MoGo is presented in Section III. The game results of MoGo playing against humans in Taiwan are presented in Section IV. Discussions and conclusions are given in Section V. Finally, comments and properties of MoGo in the Taiwanese tournament are presented in the Appendix.

II. Recent Advances in Computer Go

Minimax and alpha-beta searches are the most common techniques used in computer games. In Go, these algorithms use patterns and/or expert rules to prune search trees. However, they cannot compete with humans in 9×9 or 19×19 Go games. One reason for this is that there is no good function to evaluate a position in Go. The MCTS algorithm has been designed to improve the performance of computer Go programs. This section introduces all the improvement features adopted by MoGo. It is divided into two subsections to briefly examine the MCTS approach. Why Monte Carlo evaluation is successful in Go is described in subsection A. The MCTS algorithm is then presented in detail in subsection B, in which

Monte Carlo evaluation is adaptively biased by statistics from previous simulations. Additionally, the various formulas used in choosing compromises between exploration and exploitation, *i.e.*, choosing which sequences should be studied carefully, are also described in subsection B.

A. Monte Carlo evaluation

Brugmann proposed an original evaluation function based on Monte Carlo exploration [19]. For a given situation s, the evaluation value is the probability of winning when a game is completed by an ad hoc random player p playing both black and white. This evaluation function depends on situation s, random player p, and the number n of simulations. Although this evaluation function proposed by Brugmann is quite generic, it has the following drawbacks:

- The evaluation function relies on a random player p. Designing such a random player is a "dark art" [20]. That is, one can improve the performance of p as a standalone Go player and still have a weak evaluation function built on top of it. State-of-the-art Monte Carlo simulators have been designed by trial and error using the complete algorithm (*i.e.*, random player p is relevant when the *complete* algorithm makes decisions with the evaluation function. Designing a good Monte Carlo simulator therefore involves experiments that last for several months;
- The evaluation function may be very weak when robust evaluation functions exist (e.g., in chess);
- The evaluation function can be very computationally expensive when games are very long.

For Go, only the first drawback is relevant, which may explain the success of the Monte Carlo evaluation. The *All-Moves-As-First* (AMAF) value of a move improves Monte Carlo evaluation. This value is a good heuristic for identifying good moves [19][21]. In the MCTS setting, AMAF values are usually called *Rapid Action Value Estimation* (RAVE) values. The AMAF value of move m for player q (white or black) in situation s with random player p after n simulations is w/N, where (1) N is the number of games in n simulations, in which move m is played by player q before possibly being played by the opponent later in the game; and (2) w is the number of *won* games in n simulations, in which move m is played by player q before possibly being played by the opponent. The important point here is that move m is not necessarily the first move in situation s. This study considers AMAF values in *all* simulations, including move m

played by the player whose turn is to play in situation *s*. These AMAF values are poor quality as the order of moves is not preserved: an AMAF simulation is not necessarily consistent with Go rules, as a permutation of a consistent game is in many cases inconsistent. For each real simulation, this study has a significant amount of AMAF values—one for each move by the same player in a simulation.

B. Combining Monte Carlo evaluation and a tree search

The Monte Carlo evaluation function evaluates a position in random games played starting from this position. Thus, Monte Carlo evaluation can be used as an evaluation function in an alpha-beta engine. However, a recent and considerable improvement is the incremental construction of a tree on top of the Monte Carlo evaluation function. In each iteration, one simulation is launched from the current situation; however, the initial part of the simulation is in the tree (Fig. A3 of Appendix), which grows by adding the first situation of the simulation not yet in the tree. Outside the tree, the simulation uses the default random policy, whereas in the tree, simulation is based on moves that maximize a score combining two criteria: (1) Exploration criterion: moves that have not been simulated often should be simulated frequently; and (2) Exploitation criterion: moves that lead to high probabilities of winning should be simulated often. This implies that (1) Hashtable should be used to retain in memory many situations and statistics for these situations; these statistics can be used to adaptively change simulations using a compromise between exploration and exploitation; (2) The quantitative formulas should be defined to specify these biases. Therefore, the resulting algorithm is called the MCTS algorithm [22][23] whose pseudo-code is shown in Algorithm 1. Additionally, the MCTS algorithm depends on the Monte Carlo player (see Algorithm 2) and quantitative formulas (see Algorithm 3) that introduce bias into random choices.

Algorithm 1: Pseudo-code of a MCTS algorithm applied to a two-player game (typically Go or chess). T is a tree of positions, with each node equipped with statistics (number of wins and losses in simulations starting at this node). Concerning the decision line at the very end of the pseudo-code, the most simulated decision is known as the most reliable criterion; other solutions such as taking the decision with the highest ratio "number of wins divided by the number of simulations" are insufficiently robust, due to the possible small number of simulations. Here the reward at the end of each simulation is binary (win or loss) and deterministic (the reward only depends on the moves and not on random play), but arbitrary distributions of rewards can be used.

Initialize T to a single node, representing the current state.
while Time left > 0 do
- **Simulate one game until a position** L is out of T (thanks to bandit algorithms, see Algorithm 3).

- **Simulate one game from position L until the game is over** (thanks to the random player, see Algorithm 2).
- **Growth of the tree:** add L in T.
- **Update statistics in the entire tree:** In UCT, we have to store in each node how many simulations and how many winning simulations have been performed from this node. In other forms of tree search, some more information is necessary; for example, the Bandit Algorithm for Smooth Trees (BAST) algorithm [24] needs more general information on the size of the tree, and "AMAF-variants" of MCTS (discussed later and presented in [21]) need some more subtle statistics on past simulations.

End while
Return the move which has been simulated most often from the root

Various formulas exist for choosing compromises between exploration and exploitation. This work defines the (1) Upper Confidence Bound (UCB) formula, (2) AMAF values (or RAVE values), (3) Heuristic values, and (4) Progressive unpruning (or Progressive widening), as follows.

1) UCB formula

The most classical formula for choosing compromises between exploration and exploitation is the UCB formula [36][37]. This formula provides a score for each possible move; the move with the highest score is simulated. The score of a move m is the sum of the frequency of simulations won among all simulations starting with move m and a confidence term $c\sqrt{\log(2+sims(s))/sims(s,m)}$. The overall formula is Eq. (1):

$$score(m) = wins(s,m)/sims(s,m) + c\sqrt{\log(2+sims(s))/sims(s,m)} \quad (1)$$

where c is an ad hoc constant, $sims(s, m)$ is the number of simulations starting at s with first move m, and $sims(s)$ is the total number of simulations starting at s. This work then simulates the move with a maximal score (see Algorithm 3).

Some variants of UCB, e.g., UCB-tuned [25], have been proposed and scholars have believed that the UCB formula is key to a successful MCTS. However, in most cases (as in the case of MoGo, as RAVE values are used), using $c=0$, which is surprising, is the best choice, at least when frequency is regularized by, for example, *(number of wins + K1) / (number of simulations + K2)* for some ad hoc constants, $K1 > 0$ and $K2 > 0$, where $K1$ and $K2$ are absolute, and do not depend on the node. No optimized programs exist in which $c>0$ provides significant improvements. We believe $c>0$ is only suitable for preliminary implementations without RAVE values, without progressive widening, without heuristic values learned

from datasets, and without tuning constants. Moreover, in some cases discussed in personal communications, the authors of effective programs with $c>0$ have admitted that the advantage of $c>0$ was unclear. On the other hand, the fact that the constant c is zero has been debated on mailing lists, but never stated clearly is studies. In fact, many studies still claim that UCT is used in MCTS, whereas in MoGo, a modified UCT, namely, $c=0$, is used. UCB has however been quite useful for understanding the algorithm. We could see that the exploration constant $c>0$ was useful in early versions of MoGo whereas it became useless when heuristics were added. By empirically tuning the constant c to 0, we made tree search in computer Go essentially a best-first search (see however the discussion of optimism in front of uncertainty in Section III.B).

When UCB is used, MCTS is called an Upper-Confidence Tree (UCT) [26]. Interestingly, some significant improvements in chess have been achieved with "forced moves" [10]. Forced moves are moves that are almost mandatory for a player. A combination is a sequence of such moves when an opponent can only play forced moves. Via forced moves, one can increase analysis depth. In chess, according to some scholars, this technique is efficient and programs with forced moves can announce checkmate and victory far in advance. In computer Go, forced moves are a difficult concept. Cazenave [27] produced notable results using forced moves; however, forced moves are only forced in the sense that they are necessary for some particular goal, not for complete victory. In Go, strong players never try to keep some group alive. If the opponent can use many stones to kill one group, letting the opponent reach his target is fine, as during that time, influence is extended to another part of the goban, increasing the amount of territory won. Via the MCTS approach, very deep sequences can be produced, allowing computer Go to reach the same complexity as computer chess despite a lack of forced moves.

2) AMAF values

The bandit algorithm, which is based on AMAF, was developed by [21]. Generally, the first statistic, AMAF, is created by permutations of moves in simulations. If a move is often in a winning simulation, it will be considered a possible move. Therefore, the proportion of won simulations in simulations containing move X is a criterion for analyzing move X, as well as the proportion of won simulations in simulations

with X as a starting move. The second statistic is clearly better asymptotically than the first one because it is based on real simulated games; however, the AMAF statistic is relatively much faster. It is much faster because for each simulation, many AMAF simulations (namely, all simulations obtained from the real simulation by permutation of one move with the first move[1]) exist; that is, all simulations obtained from a real simulation after exchanging the first move with another move of the same player later in the simulation. The score for one move m is then a weighted average of the ratio of won simulations (estimated in simulations starting with move m) and the ratio of won AMAF simulations (*i.e.*, simulations in which move m occurs for the same player before occuring for the other player). The score for one move m is formally defined as Eq. (2):

$$\begin{aligned} &Score(move\ m,\ situation\ s) \\ &= \alpha \times ratio\ of\ won\ simulations + (1-\alpha) \times ratio\ of\ won\ AMAF\ simulations \end{aligned} \tag{2}$$

where weight α increases toward 1 as the number of simulations starting at s with move m moves toward infinity and is small for a small number of simulations. The move chosen for simulations is the move with the maximum score (see Algorithm 3).

3) Heuristic values

Adding a term based on the patterns and rules to the scores computed above was proposed by [28][29][30]. Typically, a value is proportional to the frequency of a move m in a pattern p according to a database, plus a coefficient tuned empirically for moves matching some expert rules. The main rules in MoGo are such classical rules as (1) avoidance of big self-atari, (2) avoidance of empty triangles, (3) territory lines, (4) walls, and (5) connect. Other rules implemented in MoGo can be found in [30]. The website (http://senseis.xmp.net) provides the Go definitions required for implementing these rules.

4) Progressive unpruning

Progressive widening [29] and progressive unpruning [23] improve Algorithm 3, which considers only the $K(n)$ "best" moves according to some heuristic at the n-th simulation in a given node, for some

[1] Only moves of the same color are permuted. See [21] for a detailed description of the AMAF method in a Monte-Carlo Tree Search (MCTS).

non-decreasing mapping $K(n)$. The decision to be simulated is that with a maximum score among $K(n)$ moves. However, when AMAF values are implemented, this improvement to Algorithm 3 is minor.

III. The Mechanism of MoGo

This section presents the Monte Carlo player used in MoGo in subsection A, and the formulas for biasing the Monte Carlo simulator, *i.e.*, the bandit formula, also known as the compromise between exploration and exploitation, in subsection B. Subsection C discusses the parallelization of MoGo.

A. The Monte Carlo player

The Monte Carlo player is defined in Algorithm 2. An *atari* occurs when a string representing a group of stones can be captured in one move. Some Go knowledge, such as 3×3 hand-crafted patterns, has been added to play meaningful games [20].

Algorithm 2: Algorithm for choosing a move in Monte Carlo simulations for the game of Go. Some details have been omitted for clarity; see [30] for details (in particular the "Nakade" modification).

```
if the last move is an atari then
    Save the stones which are in atari.
else
    Randomly pick up 6 empty locations on the goban.
    if one of them is empty and the 8 surrounding locations are empty then
        Play in this location.
    else
        if there is an empty location among the 8 locations around the last move which matches a pattern then
            Play randomly and uniformly in one of these locations.
        else
            if there is a move which captures stones then
                Capture stones.
            else
                if there is a legal move which does not fill a friendly eye then
                    Play randomly such a legal move.
                else
                    Return pass.
                end if
            end if
        end if
    end if
end if
```

B. Compromise between exploration and exploitation in MoGo

Combining various scores is classical in computer Go. Some studies have combined offline learning with statistics obtained from professional games and online learning with bandit choosing moves [28][29].

Gelly and Silver, who combined online learning with bandit choice and "transient" learning using AMAF values [21], experimented with the use of off-line learning, *i.e.*, a heuristic value. However, offline learning using Reinforcement Learning and Computer Go (RLGo) was later removed from MoGo as improvement was minor and even negative after tuning. In the current version of MoGo, improvements have been achieved by combining the following:

- on-line learning, *i.e.*, statistics such as those in a UCT, but with $c = 0$ (Section II.B.1);
- transient learning, *i.e.*, AMAF values (Section II.B.2);
- off-line knowledge, *i.e.*, expert rules and statistics in a database (Section II.B.3);
- progressive unpruning (small improvement).

The compromise between these values is as follows:

- for a small number of simulations, off-line knowledge is extremely important;
- for a high number of simulations, transient learning RAVE values becomes essential;
- after additional simulations, the "standard" statistics, ratio of won simulations, become the most important term.

Algorithm 3 presents the detailed pseudo-code for the compromise between exploration and exploitation in MoGo. Notably, $\alpha + \beta + \gamma > 1$, but converges to 1 (equivalent to 1/log(number of simulations of this move)) as the number of simulations goes to infinity. Therefore, values of $\alpha + \beta + \gamma > 1$ can be used for moves that are moderately explored—this ensures diversity and it is the only part of MoGo which has such a diversity criterion, i.e. it is the only part which is not a completely best-first approach. This is a form of optimism in front of uncertainty for the values of unvisited nodes.

Algorithm 3: Algorithm for choosing a move in the tree, for a two-player game with binary reward (extensions to arbitrary distributions are straightforward). *sims(s, m)* (resp. *wins(s, m)*) is the number of simulations (resp. won simulations) in which move *m* has been chosen in situation *s*. The total number of simulations at situation *s* is $sims(s) = sims(s, m_1) + sims(s, m_2) + \ldots + sims(s, m_n)$ where m_i's are the possible moves. The prefix "amaf" holds for statistics on AMAF-simulations instead of standard simulations (see section II.B.2). For each simulation with moves $m_1, m_2, \ldots, m_k$, we consider the corresponding AMAF-simulations: in the *i-th* AMAF-simulation associated to this simulation, m_1 is replaced by m_i (for *i* odd, as both stones must be of the same color).

Function decision = Bandit(situation *s* in the tree).
For *m* in the set of possible decisions do
    p1(m, s) = wins(s, m) / sims(s, m)

p2(m, s) = amaf-wins(s, m) / amaf-sims(s, m)
p3(m, s) = heuristic value of *m* in situation *s*.
Score(m,s) = α×p1(m, s) + β×p2(m, s) + γ×p3(m, s),
where
- *α goes to 1 as sims(s, m) goes to infinity (and α negligible for sims(s,m) small) :*
- *β is negligible for sims(s,m) small, and then increases, and later (for sims(s,m) large) goes to 0;*
- *γ starts at a value >1 and then decreases to 0 as sims(s, m) goes to infinity;*

Decision = $argmax_d$ Score(s, m)

End for

C. Parallelization in MoGo

Parallelization of MoGo is described in detail in [31]. Essentially, two types of parallelization exist.

- Multicore parallelization for shared memory, which exists in most MCTS implementations. Basically, this parallelization runs simulations independently on each core of a machine, and each core updates the same tree T.
- Message-passing parallelization without shared memory. This parallelization has one MCTS working independently on each computation node. At a frequency of 3Hz, all computation nodes merge their trees as follows:
  (1) all nodes of all the trees with > 5% of the total number of simulations at a depth < 10 are shared (only those nodes);
  (2) for each shared node, all statistics for win/loss/AMAF-win/AMAF-loss are averaged among all computation nodes.

Message-passing parallelization is presented in the Appendix (Fig. A3).

IV. Game Results of MoGo Playing Against Human Players in Taiwan

This study constructed a platform for the GO games held at NUTN, Taiwan, from August 26 to October 4, 2008 (http://go.nutn.edu.tw/). Table I lists the profiles of all Go players who competed against MoGo. Table II lists the Chinese rule adopted and related game parameters. During the tournament, MoGo ran on a Dell PowerEdge R900 machine with 16 cores and supercomputer "Huygens," which was provided by Dutch research organizations, Stichting Academisch Rekencentrum Amsterdam (SARA) and National Computer Facilities (NCF). The MoGo program was allowed to use at most 25 of the 104 nodes of the

supercomputer, *i.e.*, 800 cores at 4.7GHz, with a floating point processing power of 15 Teraflop (>1000 times that of Deep Blue). The game was played over the KGS Go server platform when MoGo was run on the Huygens cluster with different numbers of cores.

Table I. Profiles of all the Go players competing with MoGo.

Title	Name	Age	Sex	Dan grade
Mr.	Jun-Xun Zhou	28	Male	9P Professional
Mr.	Biing-Shiun Luoh	45	Male	6D Amateur
Prof.	Shang-Rong Tsai	55	Male	6D Amateur
Mr.	Cheng-Shu Chang	50	Male	6D Amateur
Prof.	Cheng-Wen Dong	70	Male	5D Amateur
Child	Yu-Shu Huang	12	Female	4D Amateur
Child	Yu-Xin Wang	11	Male	3D Amateur
Mr.	Wen-Tong Yu	50	Male	3D Amateur
Child	Sheng-Yu Tang	10	Male	2D Amateur

Table II. Parameters of the game.

Game Board	Komi	Time per side (min)
9×9	7.5 unless otherwise stated (some games with Komi 6.5)	30
19×19	7.5	45

Tables III and IV list the information related to the 9×9 and 19×19 game results MoGo played against nine Taiwanese Go players, respectively. All Smart Go Format (SGF) files of the game records (Tables III and IV) are available on the website, (http://go.nutn.edu.tw/eng/result_eng.htm). In Tables III and IV, the first column shows the game number and the second column lists MoGo performance. Performance is represented by *XD*+ or *XD*- with $X = L - H$, where L is the rank of a player and H is the handicap level. When MoGo won, its performance was *XD*+; otherwise, its performance was *XD*-. The level Xkyu corresponds to −(X−1) Dan. As the Dan number increases, player proficiency increases. The 9P player Zhou (9D on the pro scale) is assumed equivalent to a 10D player (10D on the amateur scale). Luoh, a Go teacher and 6D amateur, and Tsai, the chief referee of the tournament and a 6D amateur, were invited to comment on game results. Their comments on the 9×9 and 19×19 games are provided in the next two subsections, respectively.

A. Comments on the 9×9 games

Games Nos. 15 and 16 were very interesting *9×9* games. In these two games, MoGo played against Zhou (the 9P Go player). Figures 1 and 2 show the boards for these two 9×9 games, respectively. According to the comments of Tsai and Zhou, game No. 15, shown in Fig. 1, was worth studying because

MoGo had a chance to win. However, MoGo was tricked by Zhou with White 20 and lost the game. Zhou analyzed that if time per side was extended, then MoGo would have taken the advantage. Figure 3 indicates that the probability of playing a good move (E9) instead of one of the two bad moves exceeded 50% after 5 minutes×computation cores, which quickly reached parallelization. The probability of playing D8, E2 or E9 depends on computational effort. Bad moves D8 and E2 are likely to be played when MoGo has little time, and the probability of playing E9 increases as computational effort increases. The 9×9 Go game is the first field to which MCTS methods have been applied. However, game results for MoGo in Taiwan were not good against top-level human players; MoGo lost most of its 9×9 games—two games against Zhou and three games against Luoh. Additionally, the first game against Zhou was difficult. The professional player predicted during the game that MoGo was likely to win prior to its big mistake (see Fig. 1). Nonetheless, MoGo won one of two games against 6D Tsai. Figures 4 and 5 show the outcomes of games Nos. 3 and No. 4, respectively. Figure 6 shows the outcome of game No. 10.

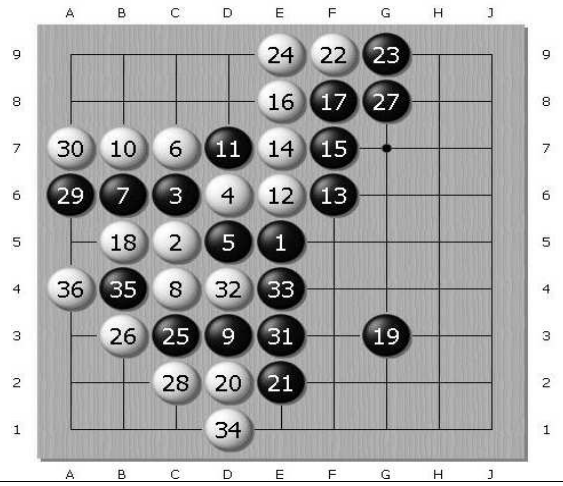
Table III. Related information about results of the 9×9 games that MoGo played against humans in the tournament.

No	Performance	Date	Setup	Environment	White	Black	Result
1	9×9 5D+	08/26/2008	9×9	Huygens with 150CPUs	**MoGo**	Dong	W+0.5
3	9×9 6D+	08/26/2008	9×9	Huygens with 150CPUs	Tsai	**MoGo**	B+Resign
4	9×9 6D-	08/26/2008	9×9	Huygens with 150CPUs	MoGo	**Luoh**	B+Resign
10	9×9 6D-	09/25/2008	9×9	Huygens with 320CPUs	**Luoh**	MoGo	W+Resign
11	9×9 6D-	09/25/2008	9×9	Huygens with 320CPUs	MoGo	**Luoh**	B+Resign
15	9×9 10D-	09/27/2008	9×9	Huygens with 800CPUs	**Zhou**	MoGo	W+Resign
16	9×9 10D-	09/27/2008	9×9	Huygens with 800CPUs	MoGo	**Zhou**	B+Resign
24	9×9 2D+	10/04/2008	9×9	R900 machine	Tang	**MoGo**	B+0.5

Table IV. Related information about results of the 19×19 games that MoGo played against humans in the tournament.

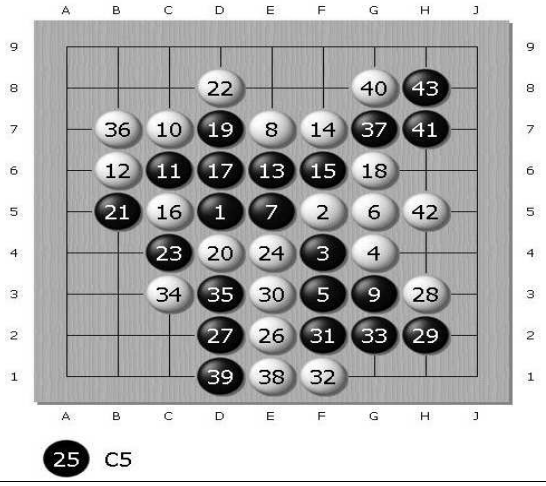
No	Performance	Date	Setup	Environment	White	Black	Result
2	1kyu+	08/26/2008	19×19 H5	Huygens with 150CPUs	Dong	**MoGo**	B+0.5
5	2 kyu+	09/24/2008	19×19 H6	R900 machine	Dong	**MoGo**	B+Resign
6	1D+	09/24/2008	19×19 H4	R900 machine	Dong	**MoGo**	B+Resign
7	1D+	09/25/2008	19×19 H4	R900 machine	Dong	**MoGo**	B+Resign
8	1D+	09/25/2008	19×19 H4	R900 machine	Dong	**MoGo**	B+Resign
9	1D+	09/25/2008	19×19 H4	R900 machine	Dong	**MoGo**	B+0.5
12	2D-	09/25/2008	19×19 H4	Huygens with 320CPUs	**Luoh**	MoGo	W+Resign
13	1D+	09/27/2008	19×19 H5	Huygens with 480CPUs	Tsai	**MoGo**	B+1.5
14	1D+	09/27/2008	19×19 H5	Huygens with 480CPUs	Chang	**MoGo**	B+1.5
17	3D-	09/27/2008	19×19 H7	Huygens with 800CPUs	**Zhou**	MoGo	W+Resign
18	3D+	10/02/2008	19×19	R900 machine	**MoGo**	Yu	W+11.5
19	2D+	10/02/2008	19×19 H4	R900 machine	Luoh	**MoGo**	B+7.5
20	1D-	10/03/2008	19×19 H5	R900 machine	**Tsai**	MoGo	W+Resign
21	1D-	10/03/2008	19×19 H5	R900 machine	**Tsai**	MoGo	W+Resign
22	4D+	10/04/2008	19×19	R900 machine	Huang	**MoGo**	B+0.5

23	3D+	10/04/2008	19×19	R900 machine	Wang	**MoGo**	B+2.5



Result:
The 9×9 game was played against Zhou (9P). MoGo was black and lost the game. The komi was 6.5, whereas MoGo had hard-coded the first moves for komi 7.5.
Comments by Tsai: MoGo was black. A good move was 20 for White (Zhou). Black answered with 21 E2, whereas E9 would have resulted in a win for black.
Comments from Luoh: MoGo could have played C3 as a reply to D2.
A posteriori analysis by MoGo of the situation after move 20: (1) MoGo inferred that it was likely to win with move E9 (65% probability of winning, estimated after a few seconds). (2) MoGo did not see clearly that E2 was a bad move (MoGo computed the probability of winning for a moment and generated an estimation of roughly 50%). (3) MoGo was likely to play good move E9, but could also play moves D8 (a loosing move) or E2. The probability of a good move increases as computational effort increases (Fig. 3). In many cases, MoGo simulated the 50% bad move for a long time, and did not explore the good moves sufficiently such that it converged to 65%. However, if MoGo is forced to spend 50% of its time on each move, it will choose the good move. By forcing MoGo to consider both moves is not a solution for the general case because deciding which moves should be considered is difficult.

Fig. 1 Game No. 15.



Result:
The 9×9 game was played against Zhou (9P). MoGo was white and lost the game. The komi was 6.5, whereas MoGo had hard-coded the first moves for komi 7.5.
Comments by Tsai: White (MoGo) played a bad move, move 16 (C5).
A posteriori analysis by MoGo of the situation before move 16 (C5):
With limited time per move, MoGo was likely to play bad move C5 with 50% probability of winning, and played G6 based on the other 50% probability. Interestingly, MoGo, when playing C5, was aware that this move did not lead to a good situation. However, it did not find a move with a relatively better probability of winning. Some methods, such as "distributing computational power over several moves" (e.g., parallelization in which the first move is fixed and different for each node) when the situation seems very good may be a good idea; however, this idea has not been implemented.

Fig. 2 Game No. 16.

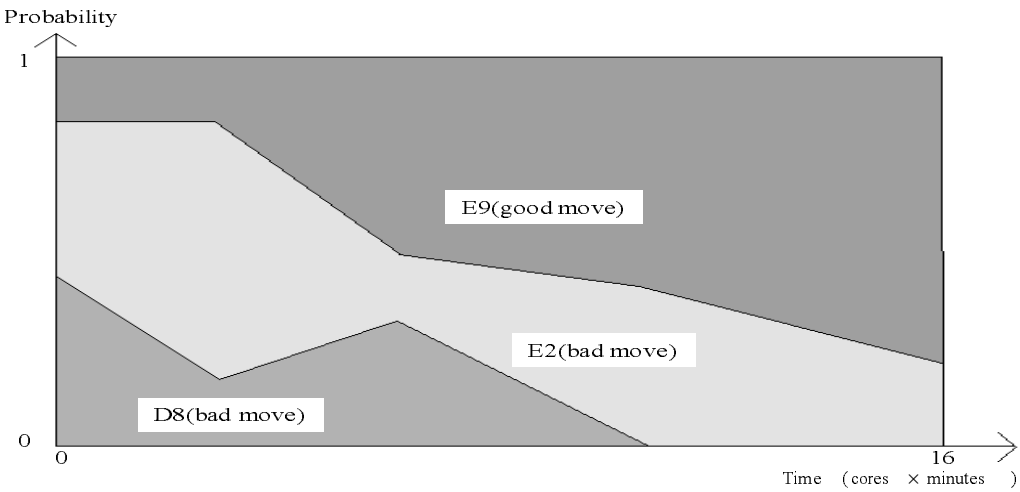
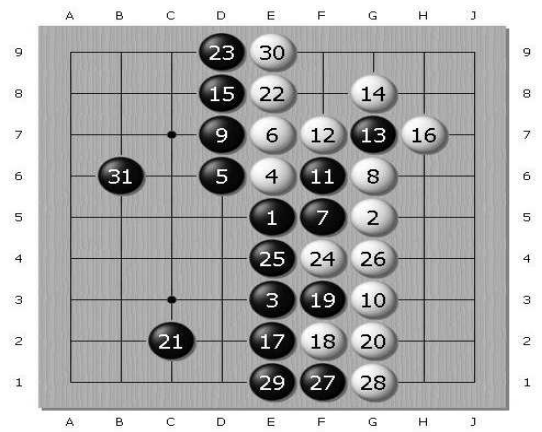
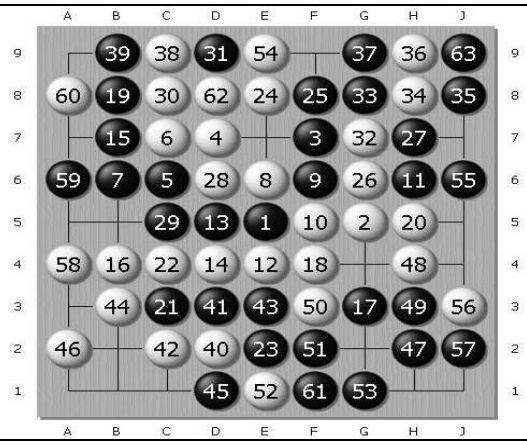


Fig. 3 Probability of playing the good move (E9), where the X-axis is time from 0 to 16 cores×minutes and the probabilities are estimated on 30 independent runs for each abscissa.



Result:
9×9 game won by MoGo (Black) against Tsai (6D)
Comments by Tsai: With Black (MoGo) playing good moves 11, 13 and 15, MoGo shows a good yose technique. Therefore, Black gets yose at 17.

Fig. 4 Game No. 3.



Result:
9×9 game lost by MoGo (White) against Luoh (6D)
Comments by Tsai: This game focused on complex fights. Therefore, there were so many variations in the game that it was difficult to analyze.

Fig. 5 Game No. 4.

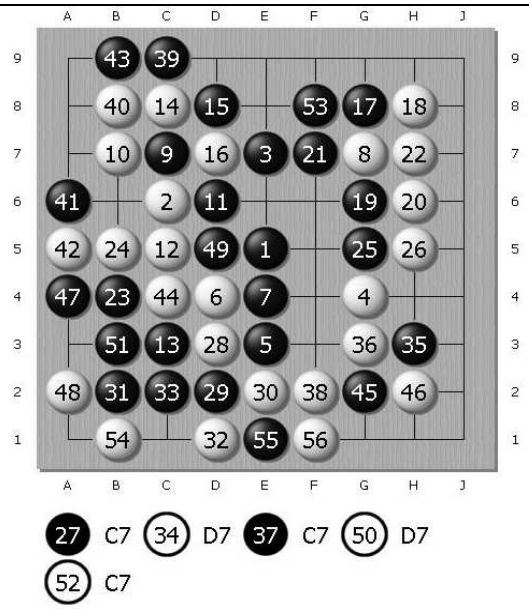
B. Comments on the 19×19 games

This subsection discusses the performance of MoGo in the 19×19 games. The following four features are of particular interest: (1) the main weakness of MoGo, namely, corners; (2) scaling over time; (3) the behavior in handicapped games of MoGo; and, (4) the primary strength of MoGo in contact fights.

1) Weakness in corners

The weakness of MoGo in corners was evident in the game against Zhou, in which MoGo lost its

advantage in all corners. Figures 7 and 8 display the outcomes of games Nos. 17 and 21, respectively. Game No. 17 with seven handicap stones clearly demonstrates the weakness of the MoGo program. That is, life-and-death conditions in the corners were not correctly assessed by MoGo. The reason was that the Monte Carlo simulator did not properly estimate semeai situations. A group is in "semeai" when this group survives if and only if a given opponent group dies; "semeai" situations involve a different reasoning based on counting the liberties of groups. Other games, such as game No. 21 (Fig. 8) show the same weakness.



Result:
9×9 game lost by MoGo (Black) against Luoh (6D)
Comments by Luoh: If MoGo (Black) had played 37 at G2 instead of C7, Luoh (White) would have played at H4. But unfortunately, Black played 37 at C7 not G2, so Black lost the game.

Fig. 6 Game No. 10.

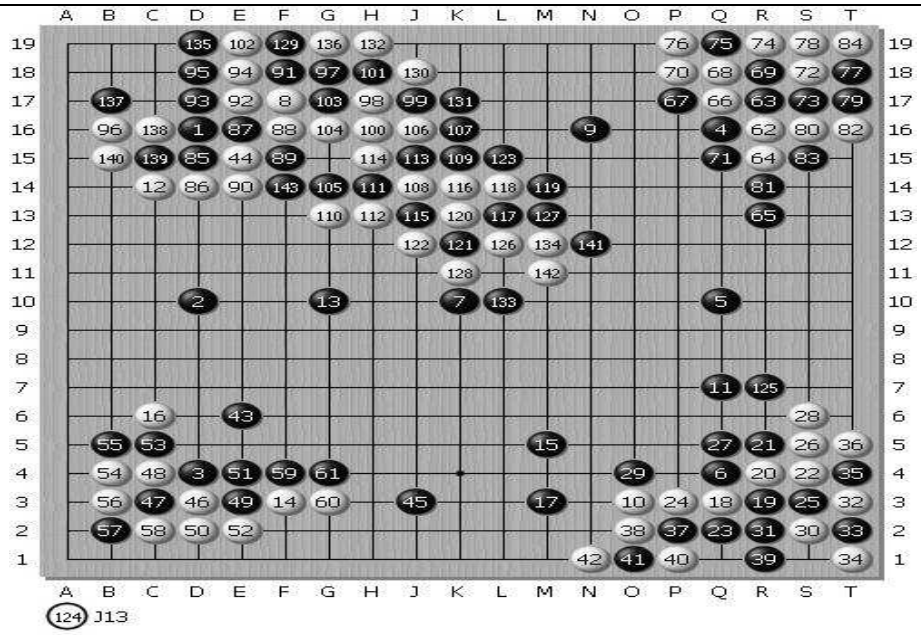
2) Scaling over time

Notably, MoGo always needs considerable time to reach its best level. In particular, the 8P Go player, Kim, won against MoGo with 11 handicap stones by setting a short time limit for moves. In this game, MoGo ran on the Huygens supercomputer with only 45 minutes per side. On the other hand, MoGo won against Kim at the 2008 US Congress in Portland with 9 handicap stones and 90 minutes per side. Moreover, Kim stated that MoGo would likely win with only 8 handicap stones. Although humans also improve over time, the game results (Table V) show that the human improvement is not as significant as that for computers. However, humans can spend a long time on particular moves, whereas the MCTS

programs typically spend the same amount of time on each move. Criteria for determining when to increase the time spent on a move are needed.

Table V. Time-setting effect on Human and MoGo.

Time per side	Won by Kim (8P)	Won by MoGo (running on Huygens, 800 cores, 4.7 GHz)
10 / 15 minutes	9 - 11 stones	12 stones
45 / 90 minutes	7 stones	9 stones



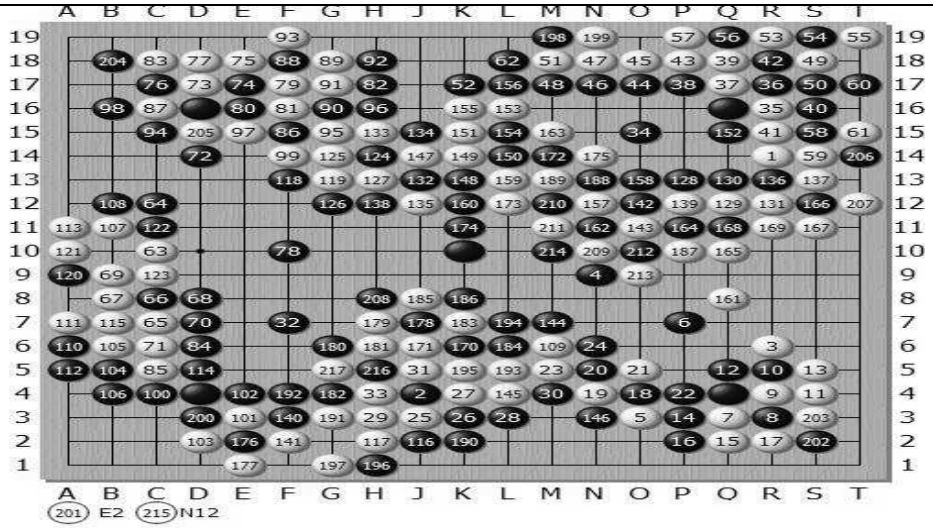
Result:
Game against Zhou (9P), with 7 handicap stones. MoGo was Black and lost the game.
Comments by Tsai: This game was a bad game which MoGo played in this championship. White (Zhou) profited from the four corners which meant that MoGo might not be good at processing the corners in the game. So, White quickly won the game after Black (MoGo) lost points at the four corners.

Fig. 7 Game No. 17.

3) MoGo in handicapped games

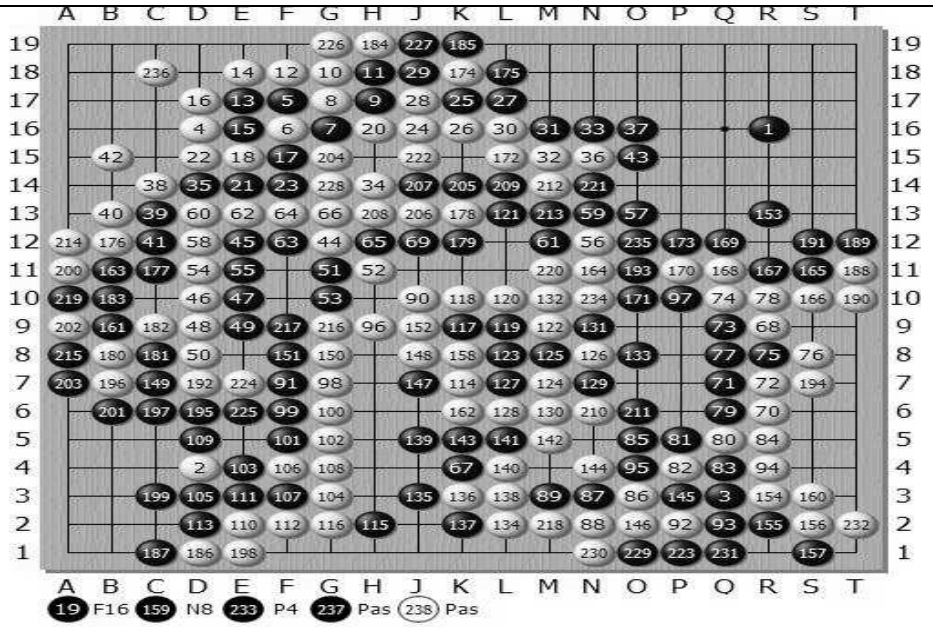
MoGo, like other MCTS algorithms, is based on best-first searching. Hence, at the start of games with high handicaps, MoGo only studies a few moves and keeps simulating each move to ensure they keep the probability of winning high. As all moves have a high probability of winning at the start of a game, based on underlying assumption of equal strength between two players (an essential assumption in MCTS algorithms), MoGo keeps simulating only initial moves. Consequently, MoGo plays its first moves almost randomly. This is in contrast with the case of an equilibrated situation without a handicap, in which MoGo will spend considerable time on various moves until it finds a move with a high probability of winning.

Interestingly, the same situation occurs in games in which MoGo has an advantage (see comments for the first 9×9 game lost against Zhou, shown in Fig. 1). This is illustrated by the successes of MoGo in non-handicapped games (Figs. 9–11), and by statistics given in Section IV.C.



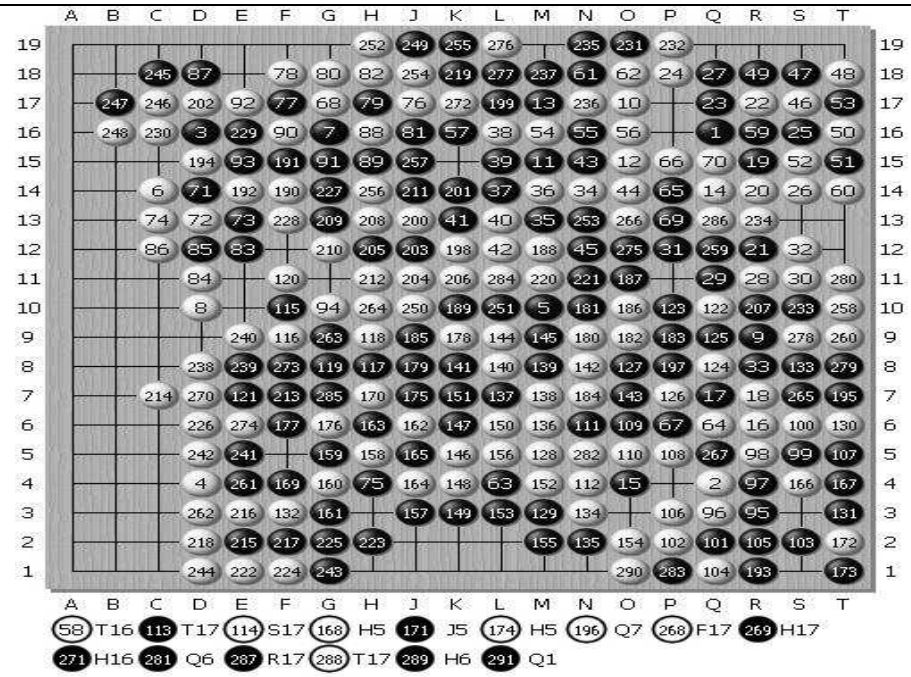
Result:
Game against Tsai (6D), with 5 handicap stones. MoGo was Black and lost the game.
Comments by Tsai: In this game, Black (MoGo) made a mistake on the right upper corner so that Black lost the game. After playing with Black for some games, Tsai thought that Black had made such a mistake many times. This was not a good move.

Fig. 8 Game No. 21.



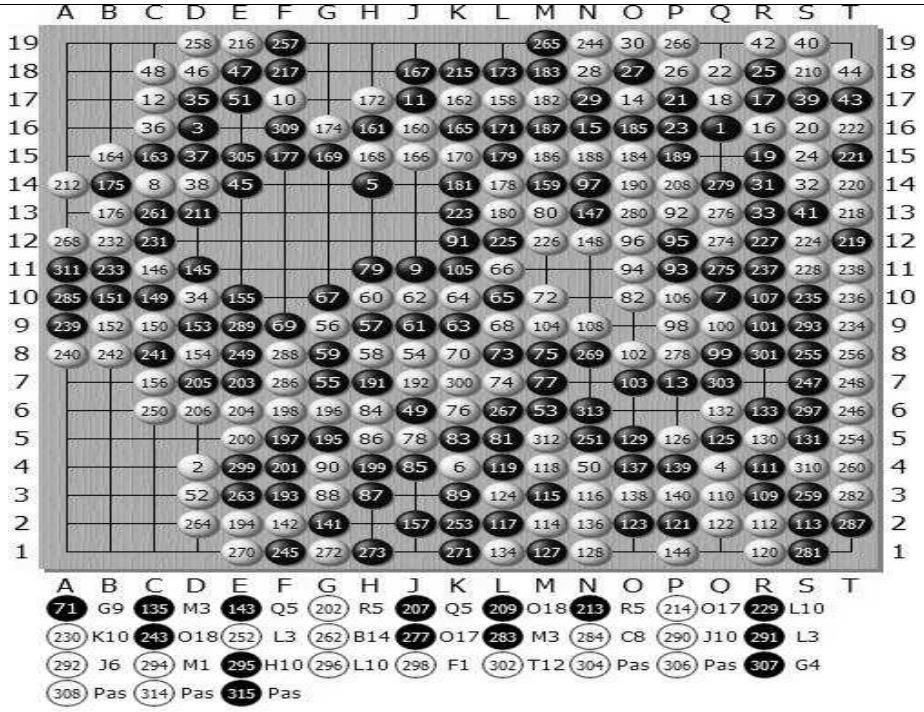
Result:
Game against Yu (3D), without handicap stones. MoGo was White and won the game.

Fig. 9 Game No. 18.



Result:
Game against Huang (4D), without handicap stones. MoGo was Black and won the game. MoGo successfully attacked the white Q4 group and therefore managed to win the game.

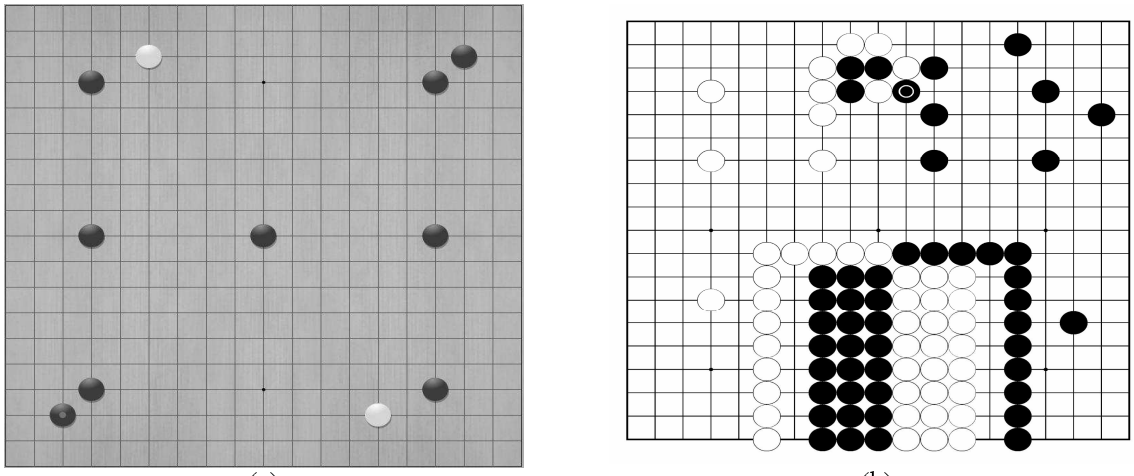
Fig. 10 Game No. 22.



Result:
Game against Wang (3D), without handicap stones. MoGo was Black.

Fig. 11 Game No. 23.

A useful solution for MoGo in handicapped games is to avoid corner trouble. The first moves can be hard coded to ensure nothing occurs in corners. Figure 12(a) presents defensive moves for black in corners when the handicap is seven stones. Thus, black plays defensive moves to ensure that two corners will remain black, and does not try to protect the other two corners. This solution, however, is ad hoc and based on human expertise. Of course, a solution based on deep improvements in the MCTS would be more elegant. Cases of poor behavior by MoGo in corners often involve "semeai," which are very common in corners. However, such a "semeai" can be simulated in an artificial situation, shown in Fig. 12(b). Figure 12 (b) is an extreme case of a very simple situation in which MoGo poorly evaluates the situation. As white, MoGo played a move that captured three stones in the upper part of Fig. 12(b), whereas it should have played in the very large "semeai" in the lower part which decides the winner. The "semeai" situation is very simple for humans, but involves much more different reasoning than other Go situations. That is, one must count "liberties" (free locations around groups)—this is obvious for a Go player but not for a Monte Carlo algorithm. The MoGo is not equipped with such intelligence, and is not encoded in any computer (this study tested without success several available programs).

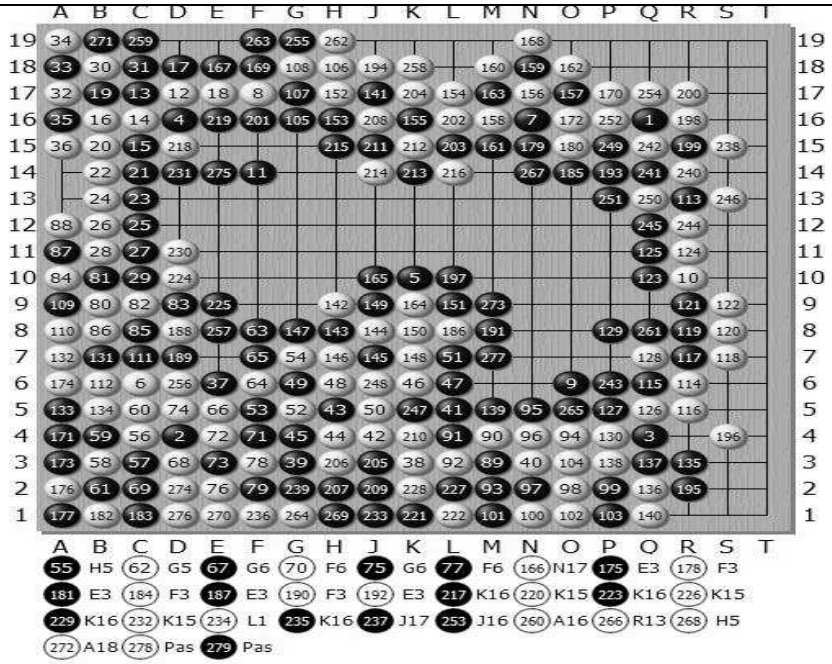


(a) (b)

Fig. 12 (a) Defensive moves for black in the corners with 7 handicap stones. (b) Artificial "semeai" situation: here the random player cannot play the right sequence and therefore estimates the probability of winning the semeai at 50% independently of the first moves.

4) Strength of MoGo in contact fights

Notably, MCTS algorithms are very effective in local fights (see Figs. 13–15).

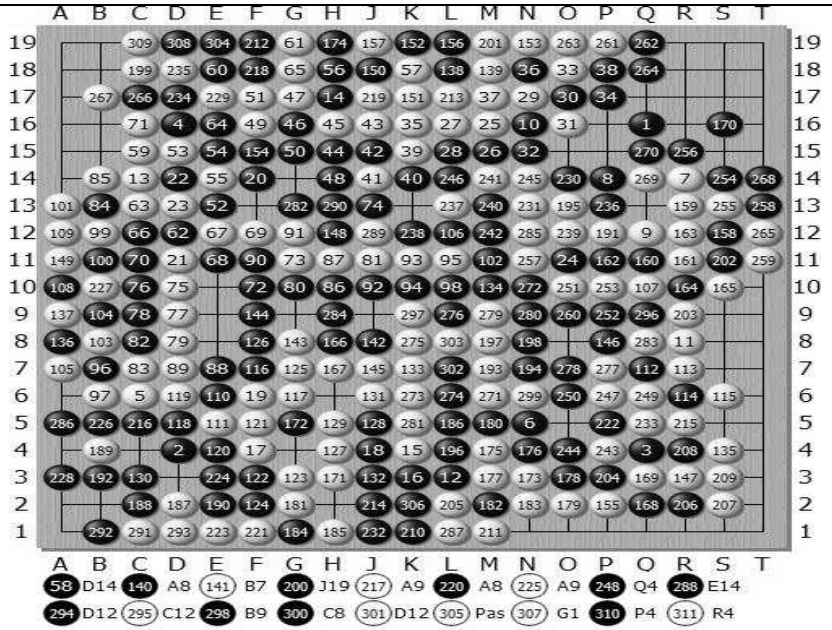


Result:

Game against Tsai (6D), with 5 handicap stones. MoGo was Black and won the game.

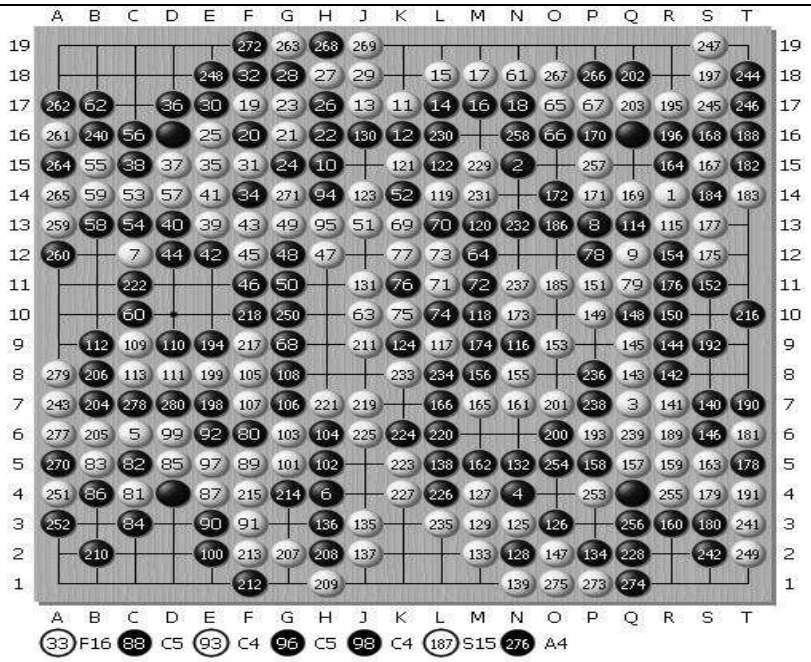
Comments by Tsai: Originally, White (Tsai) should have had a great chance to win in the middle of the game. When White played 142 to attempt to break into Black's territory, Black played 143 and 145 to cut White's stones. Meanwhile, White made another mistake. That is, White played a bad move at 146. If White had played 146 at G8 instead of H7, White would have successfully intruded into Black's territory to get more than 10 points. From the board of this game, Tsai said that MoGo had a good performance on the contact fight. In spite of the fact that Black (MoGo) also lost some points at the corners in this game, White ended up losing the game because of this vital mistake. Another key point of this game was the ko fight at 156 and 157. From the result of ko fight, Black also performed ko fight well in this game.

Fig. 13 Game No. 13.



Result:
Game against Luoh (6D), with 4 handicap stones. MoGo was Black and lost the game.
Comments by Tsai: Black (MoGo) played well in the beginning of the game, but made big mistakes at the end of the game. The key point of this game was at the fight located at the left side. Most of the time, Black played reasonable moves, but critical mistakes, such as Black 68, caused Black to lose the game. Anyway, Black had a good performance in this game.

Fig. 14 Game No. 12.



Result:
Game against Luoh (6D), with 4 handicap stones. MoGo was Black and won the game.
Comments by Luoh: Black (MoGo) played the locally optimal move when White (Luoh) played 25, which caused White to play in a difficult situation later. Therefore, Black won the game. From the result of this game, Luoh said that Black could not only detect the locally optimal move but also had a strong center territory performance. But, Black performed poorly in managing the edges and corners of the board (this point is further developed in section IV.B.1).

Fig. 15 Game No. 19.

C. *Numerical analysis of performance*

A classical formula for likelihood used in the Internet Go Server (IGS) rating system (http://www.pandanet.co.jp/English/ratingsystem/) estimates the level of players by likelihood maximization, which is based on the following formulas. The probability of losing against the L Dan player with a handicap of H stones, if one's level is B, is estimated by evaluating the following:

- Effective advantage of opponent $A = L - H - B$;
- Likelihood = $1 - (3/4)^{2A}/2$ if $A > 0$;

- Likelihood = $(3/4)^{2A}/2$ if $A<0$.

The level of MoGo can then be determined by maximizing overall likelihood, *i.e.*, the product of all likelihoods. Confidence intervals can be given that consider values with likelihoods that are at least half of maximum likelihood. The result is shown below.

(1) The level of MoGo in all games was slightly less than 2D (1.6D) and it differs depending on the machine used by MoGo:

- Games played by the R900 machine (16 cores; 3GHz): 2.5D (confidence interval, 1.7–3.3),
- Games played by the Huygens cluster: 1.7D (confidence interval, 0.8–2.7).

This is quite surprising at first view, as the Huygens cluster is a powerful machine and its acceleration is very good. The Huygens machine was used in games against Zhou and 6D players, who had a large number of handicap stones; thus, the Huygens machine performed poorly. This can be contrasted with the fact that the R900 machine was tested against 1–4D players. This introduces a bias that exceeds computational power. In particular, strong players always defeat MoGo in corners regardless of the handicap; this cannot be solved by increasing computational power.

(2) The level of MoGo in games with at most 4 handicap stones (MCTS algorithms do not handle handicaps properly) was 5D (5.3D; confidence interval, 3.8–7.8). This is surely too high as an estimate. The game MoGo won against a 4D player involved large fights, a situation that favored MoGo. Additionally, MoGo may have lost in other situations—changing just one game has a significant impact on the estimate as the number of games was small.

We conclude that the level of MoGo is estimated at 2–3D. Additionally, MoGo has (1) good fighting skills, (2) weaknesses in corners (especially in semeai situations) and (3) weaknesses in favorable situations such as in handicapped games.

V. Discussion and Conclusion

In this study, the advances in computational intelligence of MoGo are revealed from Taiwan's computer Go tournaments. The MoGo program combines AMAF/RAVE values, online values, offline

values extracted from databases, and expert rules. These techniques have three-level learning, including offline learning from a dataset, online learning with the MCTS, and transient learning with AMAF/RAVE values. In the game results in the tournament it appears that MCTS does not properly handle semeai situations. That is, the counter example—the very simple semeai for which MoGo cannot find a good answer—is a very clear example of the weakness of MCTS in Go. This is an important open problem, which is now being addressed and paid considerable attention. Additionally, the difference in levels depending on whether one considers handicapped games or non-handicapped games is highly significant — almost all games with a low handicap generate better results than almost all handicapped games.

From the game results, it is known that the level of MoGo was poorly estimated for 9×9 games; that is, changing the outcome of just one game strongly changes the overall outcome. The situation is slightly better for 19×19 games, including handicapped games, but still very imprecise for 19×19 games without handicaps, as changing the outcome of just one game markedly changes the estimated level. Despite these limitations due to a finite set of games, the following conclusions are held. (1) MoGo is weak in semeai situations. This is validated by human players and a very clear artificial semeai not solved by MoGo, which can be solved by a Go beginner. (2) MoGo is the strongest in games without handicaps. (3) MoGo is strong in contact fights; this finding is not based on statistics, but rather the unanimity among human players. This set of games is the largest set of games MoGo has played against human players with levels validated by the Go federation. Confidence intervals were used in this study: confidence intervals are built with a likelihood ratio of 0.5. Finally, according to the comments made by the Go players against MoGo in Taiwan's computer Go Tournaments, MoGo is roughly 1P professional and 2–3D amateur for 9×9 and 19×19 games on the Taiwanese scale, respectively. However, it is hoped and expected that thanks to future advances in artificial intelligence and computational power, the field of computer Go will progress in the future. Possibly, trying to combine the expert knowledge such as ontology [32][33] with the MCTS [34][35] is a way to improve the performance of MoGo.

Acknowledgements

The authors would like to thank the National Science Council of Taiwan for financially supporting this research under the grant NSC97-2221-E-024-011-MY2 and the Pascal Network of Excellence for its support. Additionally, the authors would like to thank the support of Professor Hsiu-Shuang Huang (President of National University of Tainan, Taiwan), the Computer Center of National University of Tainan, especially Mr. Yuan-Liang Wang & Miss Meng-Chun Wang, and the Tainan Go Association, especially Mr. Biing-Shiun Luoh. Finally, the authors would also like to thank the anonymous referees and Professor Simon Lucas for their constructive and valuable comments.

References

[1] S. M. Lucas and G. Kendall, Evolutionary computation and games, *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 10-18, 2006.

[2] S. M. Lucas, "Computational intelligence and games: challenges and opportunities," *International Journal of Automation and Computing*, vol. 5, no. 1, pp. 45-57, 2008.

[3] K. Chellapilla and D. Fogel, "Evolving neural networks to play checkers without expert knowledge," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1382-1391, 1999.

[4] K. Chellapilla and D. Fogel, "Evolving an expert checkers playing program without using human expertise," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 422-428, 2001.

[5] L. Messerschmidt and A. P. Engelbrecht, "Learning to play games using a PSO-based competitive learning approach," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 280-288, 2004.

[6] E. van der Werf, J. van den Herik, and J. Uiterwijk, "Solving Go on small boards," *International Computer Games Association (ICGA) Journal*, vol. 26, no. 2, pp. 92-107, 2003.

[7] B. Bouzy and T. Cazenave, "Computer Go: an AI-oriented survey," *Artificial Intelligence Journal*, vol. 132, no. 1, pp. 39-103, 2001.

[8] J. Togelius and S. M. Lucas, and R. de Nardi, "Computational intelligence in racing games," Advanced Intelligent Paradigms in Computer Games, Springer Series on Studies in Computational Intelligence, vol. 71, pp. 39-69, 2007.

[9] K. H. Chen, "Maximizing the chance of winning in searching Go game trees," *Information Science*, vol. 175, no. 4, pp. 273-283, 2005.

[10] T. Anantharaman, M. S. Campbell, and F. Hsu, "Singular extensions: adding selectivity to brute-force searching," *Artificial Intelligence*, vol. 43, no. 1, pp. 99-109, 1990.

[11] M. Müller, "Computer Go," *Artificial Intelligence,* vol. 134, no. 1-2, pp. 145-179, 2002.

[12] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo tree search: a new framework for game AI," The 4th Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, 2008.

[13] R. Maîtrepierre, J. Mary, and R. Munos, "Adaptive play in texas hold'em poker," The 18th European Conference on Artificial Intelligence (ECAI'08), Patras, Greece, 2008.

[14] M. Chung, M. Buro, and J. Schaeffer, "Monte Carlo planning in RTS games," The 2005 IEEE Symposium on Computational Intelligence and Games (CIG05), Colchester, UK, 2005.

[15] J. Y. Audibert, R. Munos, and C. Szepesvari, "Tuning bandit algorithms in stochastic environments," The 18th International Conference on Algorithmic Learning Theory (ALT07), Sendai, Japan, 2007.

[16] A. Auger and O. Teytaud, "Continuous lunches are free and the design of optimal optimization algorithms," *Algorithmica*. (DOI: 10.1007/s00453-008-9244-5)

[17] C. Hartland, S. Gelly, N. Baskiotis, O. Teytaud, and M. Sebag, "Multi-armed bandit, dynamic environments and meta-bandits," NIPS 2006 Workshop on Online Trading of Exploration and Exploitation, Whistler, Canada, 2006.

[18] L. Peret and F. Garcia, "On-line search for solving markov decision processes via heuristic sampling," Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04), Valencia, Spain, pp. 530-534, 2004.
[19] B. Brügmann, Monte Carlo Go, 1993. Online at http://www.ideanest.com/vegos/MonteCarloGo.pdf.
[20] Y. Wang and S. Gelly, "Modifications of UCT and sequence-like simulations for Monte-Carlo Go," Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games (CIG07), Hawaii, USA, pp. 175-182, 2007.
[21] S. Gelly and S. Silver, "Combining online and offline knowledge in UCT," Proceedings of the 24th International conference on Machine learning, New York, USA, pp. 273–280, 2007.
[22] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," The 5th International Conference on Computers and Games, Turin, Italy, 2006.
[23] G. Chaslot, M. Winands, J. Uiterwijk, H. J. van den Herik, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007), Salt Lake City, USA, pp. 655–661, 2007.
[24] P. A. Coquelin and R. Munos, "Bandit algorithms for tree search," The 23rd Conference on Uncertainty in Artificial Intelligence (UAI'07), Vancouver, Canada, 2007.
[25] J. Y. Audibert, R. Munos, and C. Szepesvari, "Use of variance estimation in the multi-armed bandit problem," NIPS 2006 Workshop on Online Trading of Exploration and Exploitation, Whistler, Canada, 2006.
[26] L. Kocsis and C. Szepesvari, "Bandit-based Monte-Carlo planning," *Lecture Notes in Computer Science*, vol. 4212, pp. 282–293, 2006.
[27] T. Cazenave, "Metaprogramming forced moves," Proceedings of the 13th European Conference on Artificial Intelligence (ECAI '98), Brighton, UK, pp. 645-649, 1998.
[28] B. Bouzy and G. Chaslot, "Bayesian generation and integration of knearest-neighbor patterns for 19×19 Go," Proceedings of the 2005 IEEE Symposium on Computational Intelligence in Games (CIG05), Colchester, UK, pp. 176-181, 2005.
[29] R. Coulom, "Computing elo ratings of move patterns in the game of Go," Computer Games Workshop 2007, Amsterdam, Netherlands, 2007.
[30] L. Chatriot, S. Gelly, J. B. Hoock, J. Perez, A. Rimmel, and O. Teytaud, "Including expert knowledge in bandit-based Monte-Carlo planning with application to Computer Go," The 10th European Workshop on Reinforcement Learning (EWRL08), Lille, France, 2008.
[31] S. Gelly, J. B. Hoock, A. Rimmel, O. Teytaud, and Y. Kalemkarian, "The parallelization of Monte-Carlo planning," Proceedings of the 4th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Madeira, Portugal, pp. 198-203, 2008.
[32] C. S. Lee, Z. W. Jian, and L. K. Huang, "A fuzzy ontology and its application to news summarization," *IEEE Transactions on Systems, Man and Cybernetics Part B*, vol. 35, no. 5, pp. 859-880, 2005.
[33] C. S. Lee, M. H. Wang, and J. J. Chen, "Ontology-based intelligent decision support agent for CMMI project monitoring and control," *International Journal of Approximate Reasoning*, vol. 48, no. 1, pp. 62-76, 2008.
[34] B. Bouzy, "Associating domain-dependent knowledge and Monte Carlo approaches within a Go program," *Information Sciences*, vol. 175, no. 4, pp. 247-257, 2005.
[35] L. Ralaivola, L. Wu, and P. Baldi, "Svm and pattern-enriched common fate graphs for the game of Go," Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium, pp. 485–490, 2005.
[36] P. Auer, N. C. Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235-256, 2002.
[37] T. L. Lai, H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in Applied Mathematics*, vol. 6, no. 1, pp. 4-22, 1985.
[38] J. Schaeffer and H. J. van den Herik, Chips challenging champions: games, computer and artificial intelligence, Elsevier, Amsterdam, 2002.
[39] J. Schaeffer and H. J. van den Herik, "Games, computers, and artificial intelligence," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 1-7, 2002.
[40] U. Zahavi, A. Felner, R. C. Holte, J. Schaeffer, "Duality in permutation state spaces and the dual search algorithm," *Artificial Intelligence*, vol. 172, no. 4-5, pp. 514-540, 2008.
[41] M. Cutumisu, C. Onuczko, M. Mcnaughton, T. Roy, J. Schaeffer, A. Schumacher, J. Siegel, D. Szafron, K. Waugh, M. Carbonaro, H. Duff, and S. Gillis, "ScriptEase: a generative/adaptive programming paradigm for game scripting," *Science of Computer Programming*, vol. 67, no. 1, pp. 32-58, 2007.
[42] M. Carbonaro, M. Cutumisu, H. Duff, S. Gillis, C. Onuczko, J. Siegel, J. Schaeffer, A. Schumacher, D. Szafron,

and K. Waugh, "Interactive story authoring: a viable form of creative expression for the classroom," *Computers & Education*, vol. 51, no. 2, pp. 687-707, 2008.

Biography

Chang-Shing Lee is a professor at the Department of Computer Science and Information Engineering, Director of the Computer Center in the National University of Tainan (NUTN), Taiwan. He received the Ph.D. degree in Computer Science and Information Engineering from the National Cheng Kung University, Taiwan, in 1998. He is the ETTC Chair of the IEEE Computational Intelligence Society (CIS) in 2009 and ETTC Vice-Chair of the IEEE Computational Intelligence Society (CIS) in 2008. Additionally, he is also the Committee Member of the IEEE CIS International Task Force on Intelligent Agents and the member of the IEEE SMC Technical Committee on Intelligent Internet System (TCIIS). He also serves as an Editor Board for the Applied Intelligence, the Journal of Advanced Computational Intelligence and Intelligent Informatics (JACIII), and Open Cybernetics and Systemics Journal and a guest editor for Applied Intelligence Journal, Journal of Internet Technology (JIT), and International Journal of Fuzzy Systems (IJFS). His major research interests are in Ontology Applications, Knowledge Management, Capability Maturity Model Integration (CMMI), Meeting Scheduling, and Artificial Intelligence. He is also interested in Intelligent Agent, Web Services, Fuzzy Theory & Application, Genetic Algorithm, and Image Processing. He also holds several patents on Ontology Engineering, Document Classification, Image Filtering, and Healthcare. He served as the Program Committee of the 2009 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2009), in Korea, August, 2009 as well as the Technical Committee of the IEEE World Congress on Computational Intelligence (WCCI) in Hong Kong, June 2008. He is also one of International Advisory Board for the ISDA2009 and the Program Committee for the IEEE CIIA 2009, IEEE CIIP 2009, KEOD2009, IEEE SMC 2008, IEA/AIE'08, SCIS & ISIS 2008, ICMLC 2008, ICSC2007, BIBE2007, ISIS2007, ICMLC2007, IEEE SMC2007, CIIP2007, IEA/AIE'07, SCIS & ISIS 2006, and so on. He is a Member of IEEE for CI society and SMC society, Taiwanese Association for Artificial Intelligence (TAAI), and Software Engineering Association Taiwan (SEAT).

Mei-Hui Wang received the B.S. degree in BioMedical Engineering from the Chung Yuan Christian University, Chung-Li, Taiwan, in 1993, and the M.S. degree in Electrical Engineering from the Yuan Ze University, Chung-Li, Taiwan, in 1995. From July 1995 to June 2005, she worked for the Delta Electronics, Inc., Chung-Li, Taiwan, as a senior firmware engineer. Now she is a researcher at the Ontology Application & Software Engineering (OASE) Lab. of the Department of Computer Science and Information Engineering, National University of Tainan (NUTN), Taiwan. Her research interests include intelligent agent, ontology engineering, and image processing.

Guillaume Chaslot is currently working as a third-year Ph.D. student at the University of Maastricht, Netherlands. His research topic is Monte-Carlo Tree Search techniques.

Jean-Baptiste Hoock was born in Creil, France in 1983. He received the master in computer science and image processing from Caen University and engineering diploma from the ENSICAEN Engineer School, Caen, France, in 2006. He is currently working as engineer at University of South Paris, France. His research interests include Monte-Carlo Tree Search with the application in the game of Go.

Arpad Rimmel was born in Paris, France in 1983. He received a master degree in computer science at the University of South Paris in 2005. He is currently working on a PHD in Artificial Intelligence at the University of South Paris. His research interests include reinforcement learning and Monte-Carlo tree search with application to the game of Go.

Olivier Teytaud is a researcher in TAO, Inria Saclay-IDF, Cnrs, Lri, University Paris-Sud. He works in artificial intelligence, statistical learning, evolutionary algorithms, and games.

Shang-Rong Tsai is a professor at the Department of Information Management, Chang Jung Christian University, Tainan, Taiwan. His research interests include Web technologies, distributed systems, and operating systems. He received the B.S. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 1976, and the Ph.D. degree in Electrical Engineering from National Cheng Kung University in 1987. He is an amateur Go player with the skill about 6 dan.

Shun-Chin Hsu was born in Tainan, Taiwan in 1950. He received the B.S. degree and M.S. degree in Electrical Engineering from National Taiwan University, Taiwan, in 1973 and 1980, respectively. He was a lecture at the Department of Electrical Engineering, National Taiwan University, Taiwan. He also was once a lecture, an associate professor and a professor at the Department of Computer Science and Information Engineering, National Taiwan University, Taiwan. In 1982, he retired from National Taiwan University and now he is a professor at the Department of Information Management, Chang Jung Christian University, Taiwan. His research interests include Programming Language, System Programming, and Artificial Intelligence for computer games, especially in Chinese chess and computer Go.

Tzung-Pei Hong received his B.S. degree in chemical engineering from National Taiwan University in 1985, and his Ph.D. degree in computer science and information engineering from National Chiao-Tung University in 1992.

From 1987 to 1994, he was with the Laboratory of Knowledge Engineering, National Chiao-Tung University, where he was involved in applying techniques of parallel processing to artificial intelligence. He was an associate professor at the Department of Computer Science in Chung-Hua Polytechnic Institute from 1992 to 1994, and at the Department of Information Management in I-Shou University (originally Kaohsiung Polytechnic Institute) from 1994 to 1999. He was a professor in I-Shou University from 1999 to 2001. He was in charge of the whole computerization and library planning for National University of Kaohsiung in Preparation from 1997 to 2000 and served as the first director of the library and computer center in National University of Kaohsiung from 2000 to 2001, as the Dean of Academic Affairs from 2003 to 2006 and as the Vice President from 2007 to 2008. He is currently a professor at the Department of Computer Science and Information Engineering and at the Department of Electrical Engineering.

He has published more than 300 research papers in international/national journals and conferences and has planned more than fifty information systems. He is also the board member of more than ten journals and the program committee member of more than sixty conferences. His current research interests include parallel processing, machine learning, data mining, soft computing, management information systems, and www applications.

Dr. Hong is a member of the Association for Computing Machinery, the IEEE, the Chinese Fuzzy Systems Association, the Taiwanese Association for Artificial Intelligence, and the Institute of Information and Computing Machinery.

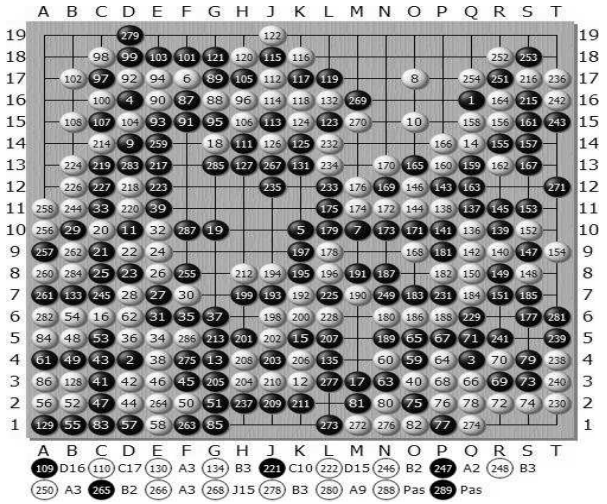
Appendix. Comments of Games for MoGo in Taiwan

In this appendix, some comments made by five Go players against MoGo in Taiwan's Computer Go tournaments are listed in Table AI. Additionally, Figs. A1(a), A1(b), A1(c), A1(d), A1(e), and A1(f) display the outcomes of games Nos. 2, 5, 6, 7, 8, and 9 with handicap 5, 6, 4, 4, 4, and 4 stones, respectively. Nonetheless, MoGo won all the games against 5D Dong shown in Fig. A1. Based on Go players' dan grade, the statistics in the game results are also presented in Fig. A2. Figure A2 shows that MoGo won most games when the Go players are ranked from 2D to 5D and the handicap has not changed the fact that the game results are much better for level lower than 5D.

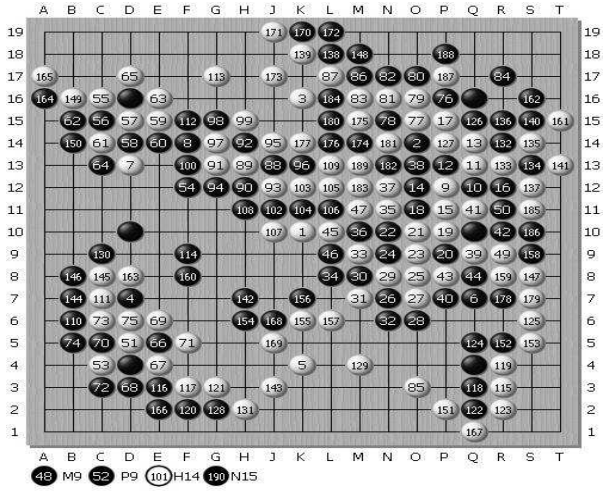
Table AI. Comments about games played by MoGo in Taiwan.

Player No	Player Name	Comments
1	Mr. Zhou (9P)	Extending time per side would benefit MoGo because when time is insufficient, MoGo is forced to play even when it has not yet finished analyzing a board.
2	Mr. Luoh (6D)	1. MoGo can play local optimization such that it attains the level of a 2–3D amateur in Taiwan. However, MoGo cannot play global optimization; thus, its assessment of territory is extremely difficult. 2. MoGo should be ranked as a 2–3D amateur, meaning that MoGo has almost the same ranking as a 6D Go player for 19×19 games with a handicap 4 stones. Thus, the game winner can be MoGo or a Go player. 3. MoGo has a good control of the center territory but has poor control at the four corners and edges; that is, its ability to surround corners is poor, resulting in MoGo mostly playing in the board center. 4. During a game, MoGo can quickly identify its opponent's weaknesses. Thus, MoGo should be with the quite level of rank. 5. The ability of MoGo to fight and attack is acceptable; however, MoGo typically makes sacrificial plays. That is, MoGo typically loses more points than it gains after a fight or attack. 6. For 9×9 games, MoGo has reached the level of professional and is a good tool for players practicing fight and attack strategies. 7. When MoGo cannot kill its opponent, it surrounds the territory and makes sacrificial moves to achieve its goal. In some cases, after a fight or attack, the points MoGo loses exceed those it gains. 8. The time to play for each side is key to MoGo victory. Compared with human Go players, MoGo needs more time to compensate for its weak rank.
3	Prof. Tsai (6D)	1. The ranking of MoGo is unstable; that is, MoGo in the best case is a 3D player, and in other cases a 1D player. Thus, the average rank of MoGo is roughly 2D. 2. The performance of MoGo was beyond the expectations of Tsai. He was also surprised that MoGo can respond to most reasonable moves. 3. MoGo has the ability to identify the weaknesses of its opponent and gain points. 4. In the latter half of a game, MoGo can surround a territory. 5. The number of poor moves by MoGo is significantly less than that by the computer Go program. 6. At the start of a game, MoGo often loses points fighting in corners.

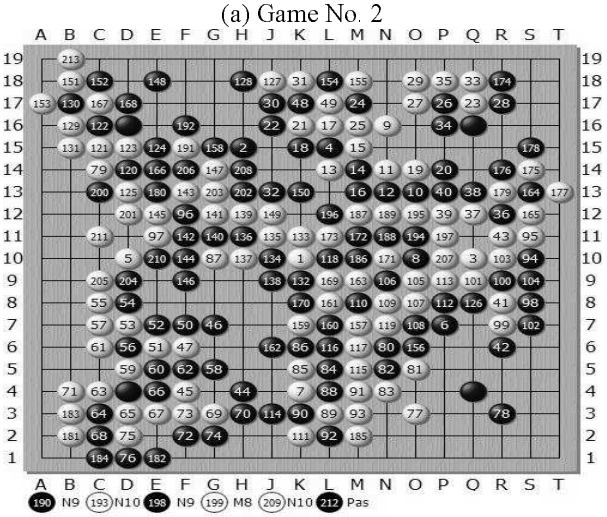
		7. When a game is nearly over, if a Go player recognizes that MoGo is protecting its territory, MoGo likely wins. Conversely, if a Go player is aware that MoGo is playing abnormally, such as when it makes poor moves, the Go player likely wins. 8. Generally, MoGo is weak at the start of a game, and becomes stronger as the game progresses. 9. MoGo may not fully understand the problems in a ko fight. A ko fight may have some complicated math problems.
4	Mr. Chang (6D)	1. MoGo was 4D in the 19×19 games and 1P in the 9×9 games. 2. MoGo has the capability to analyze. 3. When MoGo determines it will definitely win, it will handicap until it wins 0.5 points. 4. MoGo can beat a 1D Go player without handicap stones. 5. MoGo cannot play well in corners.
5	Prof. Dong (5D)	1. The rank of MoGo was beyond Dong's expectations. 2. MoGo is "intelligent." 3. MoGo can defend and fight. 4. The ability of MoGo in 19×19 games is still poorer that of humans.



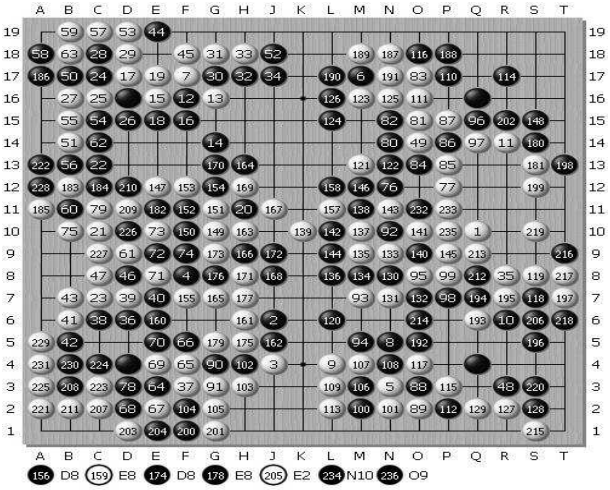
(a) Game No. 2



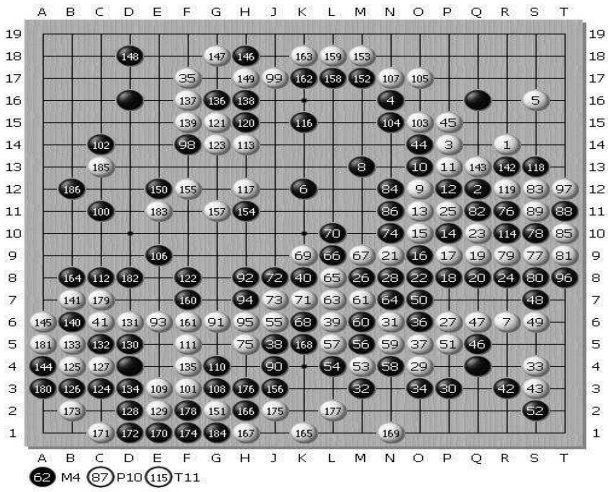
(b) Game No. 5



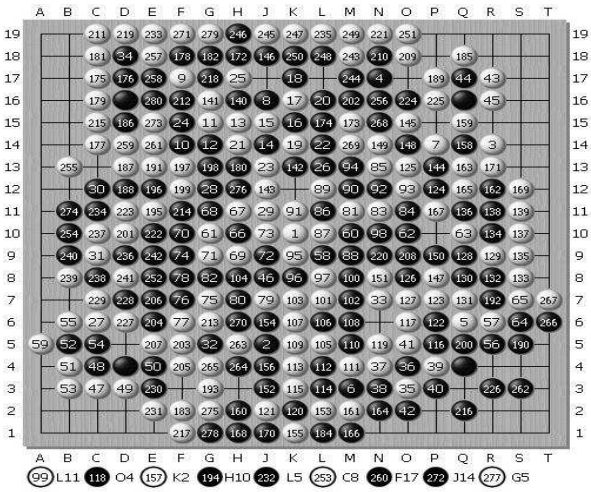
(c) Game No. 6



(d) Game No. 7



(e) Game No. 8



(f) Game No. 9

Fig. A1 Outcomes of games Nos. (a) 2, (b) 5, (c) 6, (d) 7, (e) 8, and (f) 9.

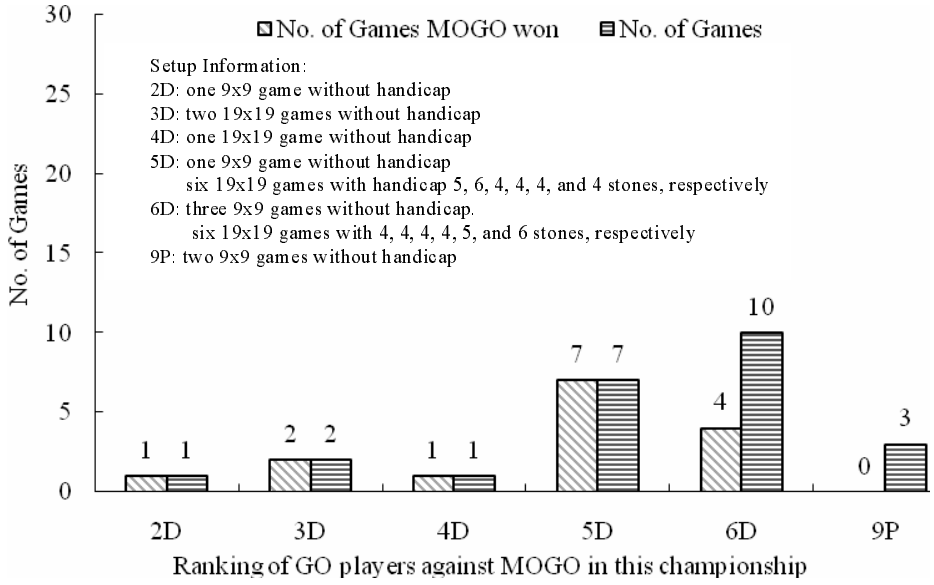


Fig. A2 Statistics in game results based on Go players' dan grade.

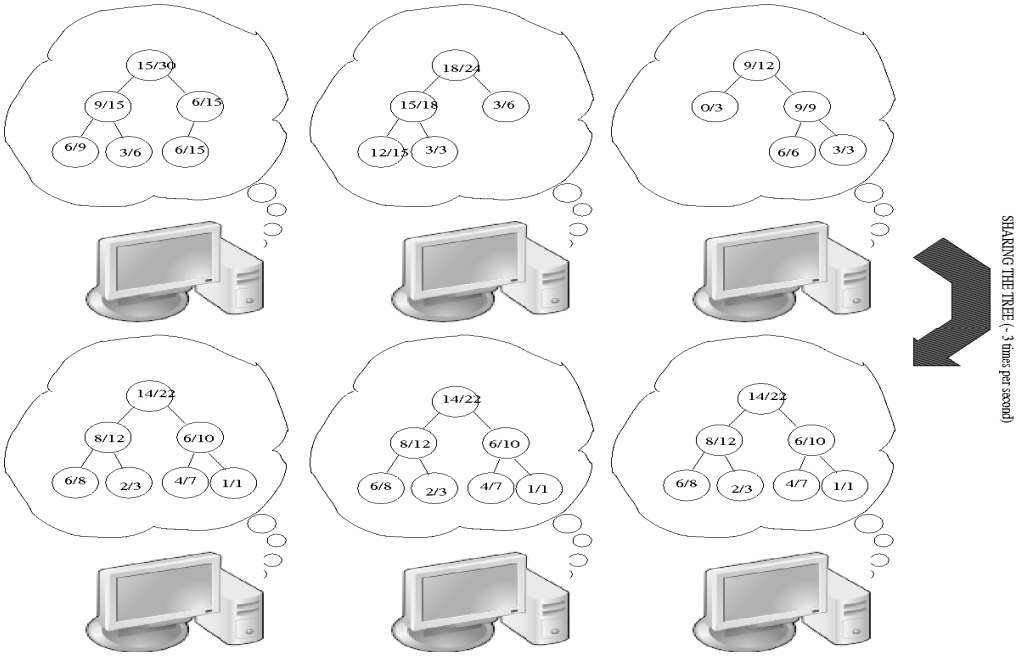


Fig. A3 One step of sharing in the MPI-parallelization of MoGo. Each computation node builds his own tree in memory, and 3 times per second all nodes "share" their tree with other nodes: all statistics are averaged, and thereafter all nodes have the same tree in memory, for the upper part (nodes with low number of simulations, i.e. less than 5% of the number of simulations of the root, or too deep in the tree, i.e. depth more than 10, are not shared).