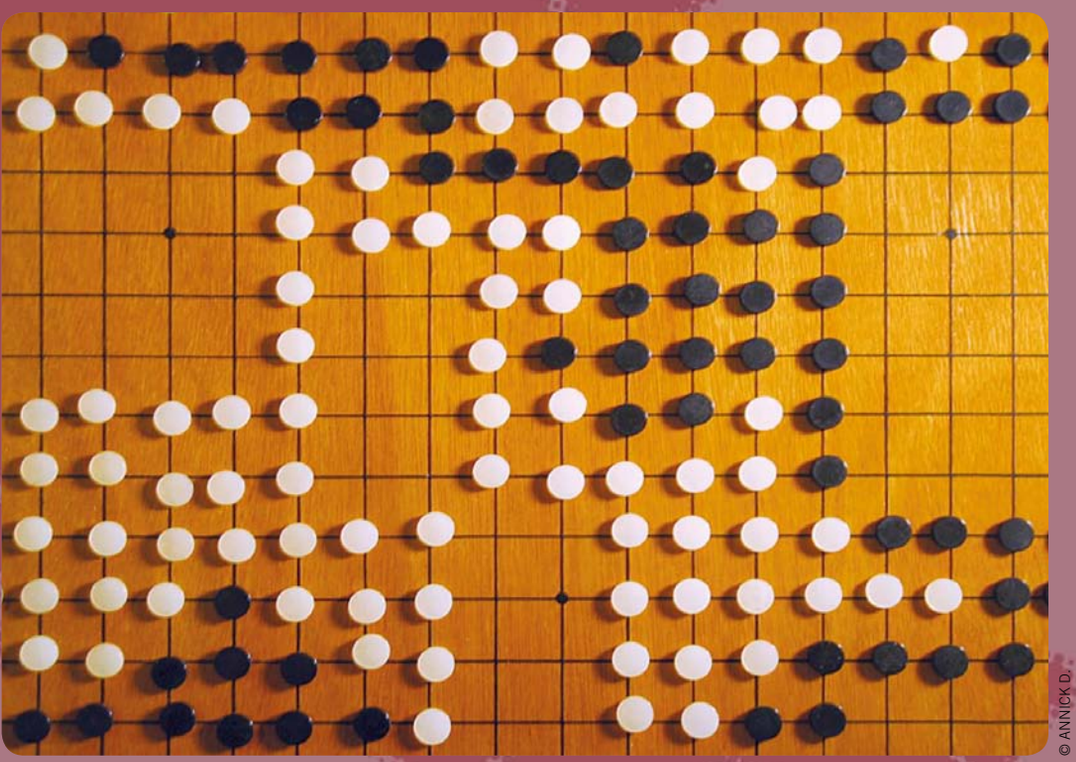
JEAN-YVES AUDIBERT*, SÉBASTIEN BUBECK****, VINCENT DANJEAN*****, SYLVAIN GELLY*, THOMAS HÉRAULT**, JEAN-BAPTISTE HOOCK*, CHANG-SHING LEE******, RÉMI MUNOS****, JULIEN PÉREZ*, ARPAD RIMMEL*, MARC SCHOENAUER*, MICHÈLE SEBAG*, OLIVIER TEYTAUD*, MEI-HUI WANG******, YIZAO WANG***

* Equipe TAO, LRI, ** LRI, *** Projet Willow, Certis, Enpc, **** Inria Futurs, équipe Sequel, ***** LIG, UMR 5217, Université de Grenoble, Ensimag, ****** National University Of Tainan, Taiwan ******* University of Maastricht

GOTHIQUE



© ANNICK D.

In 1997, for the first time ever, a computer beat Garry Kasparov, the world chess champion. In the game of Go, on the other hand, humans retain the upper hand. Go is more complex than chess, with the number of different games possible exceeding 10^{600} greater than the number of particles in the Universe. Go is therefore a remarkable school for learning strategy. Computers cannot easily compete with the best humans, but new algorithms are changing all that.

The game of Go **(Inset 1)**, is little known in France but very popular in large countries such as China, Japan and Korea. It presents a real challenge to computer scientists. The number of possible combinations is much greater than for chess, and human expertise is extremely difficult to emulate in effective computer programs. This game thus requires a form of intelligence open to humans, but, until recently, largely inaccessible to computers. The ancient game of Go provides an excellent standard for evaluating progress in artificial intelligence. Since 2006, Bandit-Based Monte-Carlo* Planning (BBMCP) technology, developed from fundamental research into "bandit problems" and exploration/exploitation dilemmas, has revolutionised planning. During the Go tournament organised in Paris in March 2008 by the French Go Federation (FFG), the artificial intelligence engine MoGo, the fruit of collaboration between our TAO team from LRI and groups from INRIA, the CNRS and the Ecole Polytechnique (CMAP), scored the first officially validated victory over a human Go master. MoGo is a computer program that makes use of BBMCP to play Go. The MoGo project was built on expertise in parallel computing and makes use of a classical form of parallel computing involving "clusters" supplied by the Grid'5000 project and Bull. One of the key features of MoGo is its ability to meet the challenge posed by the game of Go using very little specific information about the game. Indeed, many of its developers know very little about the game beyond its basic rules. MoGo has provided an opportunity to develop general methods not specific to Go and, thus, to evaluate and understand the power and limitations of Monte-Carlo planning. MoGo has thus developed from a combination of basic, experimental and applied research. The potential domain of application of these methods is vast, and includes problems with too many parameters to be dealt with by classical techniques. The planning of energy production is a typical example. The optimal reconciliation of highly diverse, but limited resources perfectly illustrates such "multiple dimension" problems.

Just like the casino

One of the key elements of MoGo is the "one-armed bandit" problem, named after the slot machines in casinos. Imagine that you are faced with two slot machines, one on the left and one on the right. Each machine provides a particular probability of winning, but you have no prior knowledge of the probability of winning with either machine. You play the machine on the right and you win. You then play the machine on the left and you lose. You then go back to the machine on the right, but this time you lose. So, you have one win in two tries for the machine on the right and a single unsuccessful try for the machine on the left. This leaves you with a dilemma: is it best to act on the information you already have (play the machine on the right, which has a better success rate so far), or is it best to explore and acquire new information (play the machine on the left, with an outcome that is less well known)? This dilemma, the so-called "bandit problem", requires you to find a strategy that maximises gain by finding a judicious balance between exploitation and exploration. This problem can be generalised in several ways: with more than two machines, the gains are no longer binary (win or lose) but quantitative (how much can I win?) depending on the number of times an arbitrary arm is pulled. Many exploitation/exploration algorithms have been developed to maximise the gain in different situations.

An algorithm is satisfactory if the optimal machine (the one giving the highest gains) is the machine most frequently played. As the ability of each machine to provide gains must be assessed by testing (i.e. by playing the machine), the key element in the algorithm is playing the least optimal machines only rarely, typically just often enough to determine that the machine is not optimal. The problem may also be limited in time: for example, it may be possible to play only 100,000 times. In such cases, an asymptotic analysis of the exploration/exploitation dilemma, based on playing an unlimited number of times, is not optimal. The algorithm can be improved, by taking into consideration the limitations on playing time.

The goal is thus to bias the algorithm so that it tends to play certain machines, presumed to provide the greatest gains, more frequently. However, it is far from trivial in this case to guarantee that the algorithm will always find the best solution, even assuming that sufficient calculating power is available. The bias must be sufficiently strong to work properly, but sufficiently weak to ensure that it can still find the optimal solution even if its initial assessment of the machines is wrong.

1 The Game of Go

The game of Go is played by two people, who take turns placing black or white stones on a board, the goban, on which a grid is engraved. The grid consists of 19 horizontal and 19 vertical lines, resulting in 361 intersections. A smaller number of lines may be used, most often 13x13 or 9x9 for rapid games or for learning the rules of the game. The player with the black stones goes first. Stones surrounded by the opponent's stones are removed from the board, and the player whose stones cover the most territory on the board at the end of the game is the winner. The rules are very simple; but the game has an immense number of moves and can prove addictive.

The tree of possible moves

In 2006, several European publications substantially increased the range of bandit problems by extending them to problems that can be represented in the form of a tree. The tree symbolises all possible positions and every available path to reach them. Imagine a situation in a two-player game in which you have to choose your next move. You draw a tree, the root of which represents the current situation. The number of branches corresponds to the number of possible moves. Each branch leads to a node corresponding to a new position. Each of these nodes leads to a new set of branches corresponding to new moves in response to your opponent's moves, and so on. If you could follow each branch until the final position in the game — win or loss — was reached, then you could choose the best move with certainty.

2 Acknowledgements

MoGo would like to thank Jean-Yves Audibert, Eric Caudal, Bertrand Chardon, Louis Chatriot, Rémi Coulom, Vincent Danjean and MOAIS, Frédéric Donzet, Alain Facélina, David Fotland, Sylvain Gelly, Jean-Baptiste Hoock, Bernard Helmstetter, Thomas Hérault, Marc Jégou, Jean-François Méhaut, Rémi Munos, Vincent Néri, Julien Perez, Arpad Rimmel, David Silver, the team at Tao, Olivier Teytaud, Clément Trung, Yizao Wang, the mailing-list computer-go, KGS, Cgos, Linux and Récitsproque.

Unfortunately, your calculating power is limited and insufficient to develop a complete tree. You therefore need to limit the depth of construction, but you don't necessarily have to use the same depth for all moves. You can stop searching along certain branches right from the first move if they are clearly bad, whilst pursuing the most promising branches for up to 30 moves. But how do you decide which nodes to develop? That is where the bandit technique comes in, making it possible to find a compromise between exploring and exploiting. Simulations of games are carried out, beginning at an individual node and continuing until completion of the game. Such simulations are carried out for each node. If a simulated game ends in victory, then a "win" is associated with the node from which that game began. If the simulated game ends in defeat, a "loss" is associated with the node. This results in a situation similar to that for one-armed bandits in casinos, and we can use the same type of algorithm to optimise the search for a favourable branch: nodes associated with a higher frequency of victories are developed further (exploited), but nodes that have been less well searched are also considered (exploration) **(figure1)**. A large number of simulations are carried out over the space of, say, one minute, and the most promising move is actually played.

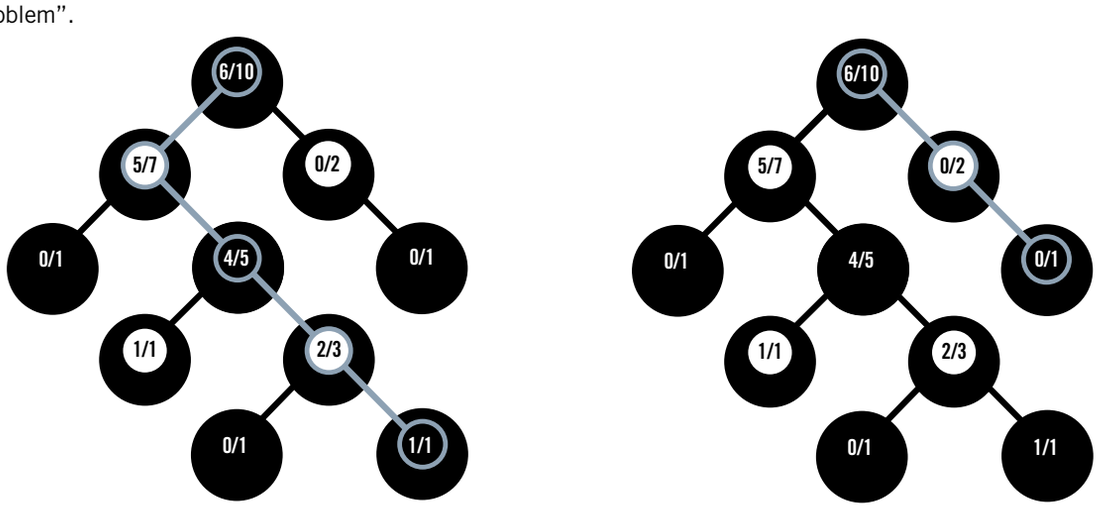
Optimizing the power of the algorithm

The classical solution to the problem of program performance being limited by the computing power available would be to run the program simultaneously on several computers in parallel. This is

FIGURE 1

The exploration-exploitation compromise

Each node correspondes to the ratio a/b, where b is the number of simulations passing through the node and a is the number of simulations resulting in a win. Either the branch leading to the maximum number of victories (left) or the least explored branch (right) may be selected. This is what is meant by the "exploration/exploitation compromise", otherwise known as a "bandit problem".



the principle behind parallel computing*. The tree approach used by BBMCP (an "in depth-first scan") is difficult to adapt to a parallel computing system. Nevertheless, using a few tricks and approximations, we have been able to make use of the two main types of parallel computing: shared memory and the passing of messages. In a parallel system based on shared memory, several computers, known as cores, read and write in the same memory. There is no need for explicit communication (each unit sees the entire memory), but these systems become physically difficult to construct once they include more than about ten computing units. In a parallel system based on the passing of messages, several computers, or computing nodes, work together, but each uses its own memory. Messages are passed between units for the sharing of information. These messages must be explicitly written in the programs, complicating the programming process. Nowadays, it is easy to imagine having thousands of computing units working together. Parallel computing with shared memory naturally leads to deeper reasoning (even to a deeper game), whereas parallel computing based on the exchange of messages naturally leads to broader reasoning (more moves considered). Intriguingly, Go players often describe the different styles of game playing in these very terms.

Our planning algorithm gradually identifies nodes corresponding to possible future positions and stores them in memory. In shared memory systems, the parallel approach is thus quite natural: each core undertakes its own simulations and the results are used to enrich the same memory. However, many obstacles remain. If several cores write to the same memory space simultaneously, it is hardly surprising that the results are far from ideal! There is therefore a need to ensure that no particular core is faced with a memory space the contents of which change unexpectedly due to modification by another core. Certain segments of the code must therefore be "protected" by serialisation, that is, by prohibiting the execution of certain operations in parallel. It is also important to avoid prevent the cores slowing each other down because they have arrived at the same memory space or at the same time in the serialised parts of the program. In other words, the real difficulty is minimising serialisation: too little and the program may function erroneously, too much and the program will not make the best use of the available computing power.
When memory is not shared, how do you "parallelise" the algorithm? Sending messages between machines is highly time-consuming, whereas simulations are very rapid. A single computing core may execute 15,000 operations per second. If we are not careful, communication time may rapidly increase to several orders of magnitude above the calculation time. The trick involves not communicating the result of each calculation, but instead bundling the results of multiple calculations together through statistics. Each computer constructs its own tree of possible positions and then, 20 times per second, for example, all the computers combine their results. This operation can be applied over a large scale, because communication time increases little with increasing number of computers. Indeed, it increases as a logarithm of the number of machines.

Grid'5000

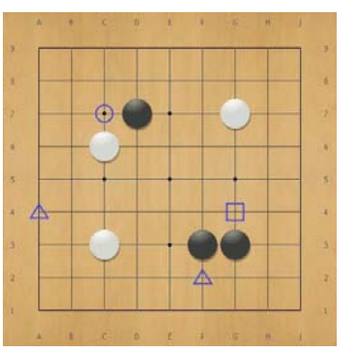
One of the major problems is the need to access an effective set of computers that are simultaneously multinode (several computers sending messages efficiently to each other) and multicore (several computing cores sharing memory relating to each node). We achieve this by making use of the resources provided by the Grid'5000 project **(Figure 2)**, an experimental computing grid* linking together nine locations in France, including Université Paris-Sud at Orsay, and providing access to a network of thousands of computing units. Grid'5000 supplies a large range of resources to computer scientists developing new algorithms for computing grids and parallel computers or studying the behavior of these machines. In particular, Grid'5000 includes a machine in Grenoble with 16 computing cores, a network of 46 eight-core machines in Lille, and a large number of clusters used in the many experiments required to improve the performance of MoGo.

A statistical simulation method with a pinch of human expertise

The BBMCP chooses the moves to explore as a function of the exploration/exploitation dilemma. But what should you do when you happen upon an enti-

FIGURE 2

The position indicated with a circle is considered by MoGo to be a priority move, because it occurs twice on the third line and corresponds to a "wall" pattern. The triangles correspond to moves considered by MoGo to be less interesting : the triangle at A4, because it is an edge; and the triangle at F2, because it corresponds to an "empty triangle" motif. Finally, the G4 square is more complicated. This would normally be considered a bad move, because it corresponds to an empty triangle pattern. However, it occurs on the third line, which is considered to be good. This move is therefore generally considered to be a weak move.

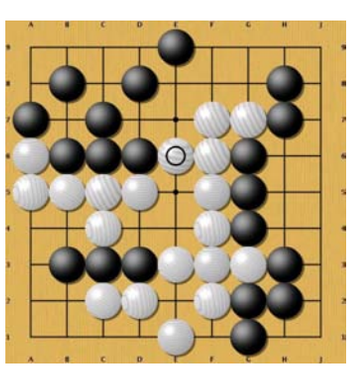


rely unprecedented situation: which moves should be given priority for exploration? A simple solution would be to select a move at random. This is the principle behind the Monte-Carlo method. However, this selection at random does not mean each possible move has the same probability. A distribution of probabilities judiciously selected on the basis of specific knowledge about the problem, in this case the game of Go, can be employed. This principle, long studied at the Universities of Paris 5 and 8, remains elusive. Clearly, the use of the Monte-Carlo algorithm that makes the best moves does not necessarily result in the best program. It is important to maintain a certain diversity in strategies, and it is better to play poorly but robustly and consistently, than to play well but with fatal weaknesses at times. Put another way, it is better to anticipate every reasonable move rather than to play excellently 90% of the time but get it badly wrong the remaining 10% of the time.

BBMCP has rapidly become established as an excellent approach to Go, despite including very little expert knowledge of the game. The best programs, however, now include some human expertise. In particular **(Figure 3)**, opponents of MoGo have discovered weaknesses in the program concerning certain positions called "Nakade". For a long time, MoGo played with insufficiently varied simulations, lacking moves of this type. It was only during games against human players that these weaknesses were eventually discovered and corrected. Go players have also suggested that MoGo should focus more on some moves and less on others: avoiding moves known as "empty triangles" and instead focusing on moves known as "walls". Finally, like human players, MoGo now studies the Tsumego problems s: positions in which it is known that one and only one move can win the game. Biasing the method of constructing trees until the right decision is one way of taking Tsumego problems into consideration.

FIGURE 3

Final position in the game won by MoGo against Catalin Taranu (5th Dan professional) on a 9x9 board, during the IAGO challenge in 2008. MoGo played white.



And the computer wins !

During the tournament in Paris in 2008, RécitSproque and the French Go Federation (FFG) organised the IAGO challenge: a game between a professional Go player and a computer on a 9x9 board (goban). The 9x9 Go game is easier for computers than the traditional 19x19 grid, but humans are still largely superior (the number of possible moves is much greater than in a game of chess). MoGo went up against Catalin Taranu, professional 5th Dan master and winner of the 2008 Shusaku cup. The series consisted of three matches on the 9x9 board on March 22nd and an exhibition game played on the 19x19 board on March 23rd. The first match was unfortunately disrupted by a technical problem. The cluster made available to MoGo by Bull, on which the program performs best, had to be replaced by a standard computer. MoGo lost and Catalin Taranu considered it to be an easy win. During the second game, the cluster was working and MoGo won **(figure 4)**. Catalin Taranu admitted that he had made some major blunders that were astutely exploited by MoGo. The third game took place without technical incident, and Catalin was on top form and beat MoGo. Finally, some of the players wanted to try play MoGo outside of tournament play, and MoGo went unbeaten in these exhibition games against highly ranked opponents.
MoGo also played an exhibition game on the 19x19 board against Catalin, but the human won, despite a nine-stone handicap. There was a problem with the connection to the Bull cluster from the start of the match, and so a standard computer was used for part of the match, until it could be replaced by the Bull cluster. Catalin said that MoGo played at a level close to a Dan player and made some brilliant moves. MoGo finally lost because of a blunder at the end of the game, but the game nonetheless lasted a long time. We thank the French Go Federation and RécitSproque for providing us with an opportunity to demonstrate our program.

Next challenge in Portland and at the National University of Tainan (Taiwan)

MoGo was then invited to play in August 2008, against Kim Myungwang, 8th Dan professional from the formidable Korean Federation, with a nine-stone handicap. Following several defeats in speed games, MoGo won a game in standard time, notably after a beautiful local victory in the bottom right corner **(figure 6)**. Emboldened by these successes at the university, MoGo went on to play against other computers in Taiwan and other human players. Although MoGo again lost against a leading player, despite a seven-stone handicap, MoGo won for the first time without a handicap against a 4th Dan player from the Taiwanese Go Federation **(Figure 7)**.

A promising future

Let us begin by considering what Go has brought to BBMCP and the amazing opportunity it has provided for demonstrating these alternative planning techniques for multiple dimensions. Go provides a suitable challenge: nobody can doubt the performance of BBMCP after its achievements in this domain. Opportunities for applications in other domains are scarce but rapidly increasing, and the example of this application for Go will play a major role in the development of other applications, by providing proof of principle that BBMCP can work through its clear victory over classical methods in the challenging game of Go. This work on Go has also provided an excellent opportunity to visualise event in a BBMCP program. The tree constructed by BBMCP reveals the pitfalls to be avoided: loss of time due to a large number of aberrant simulations or, conversely, loss of diversity. Finally, this work has favoured the development of parallel computing. The idea of running BBMCP algorithms in parallel did not occur to anyone until we realised the extent to which this could increase the efficacy of this method. Few studies have considered the use of other planning methods in parallel systems, but we strongly believe that this technology could be generalised. In particular, the use of parallel systems involving message transfer, although not necessarily intuitive, could easily be applied to situations other than Go, and even to situations beyond BBMCP. Generalised approaches (heuristics) not specific to Go are also possible. The notion of studying permutations of a simulation, for example, which was used for each simulation carried out in MoGo, could potentially be applied to other domains, such as robotics.
Other perspectives for the future, such as the automatic parameterisation of MoGo, should also be mentioned. MoGo uses a large number of constants that are difficult to select. The parameters are optimised, but a huge amount of computing time is required. MoGo could learn by playing games against itself or others in the future, but how can it learn from its errors without human intervention? Humans know how to estimate their confidence in a move, something that MoGo does not yet know how to do. When in doubt, human players may devote half the time in a match to a single decisive move, but MoGo spends about the same amount of time on each move. How can the confidence in a decision be assessed and, if necessary, how can computing power be increased in situations of doubt? Beyond Go, a few potential applications already exist, particularly in the domain of resource management. Still others are under consideration. We strongly believe that this technology has a number of advantages (efficient parallelisation, compatibility with problems involving many dimensions, the possiblity of introducing human experience) and has a promising future. Go is simply the first step! ■

Glossary

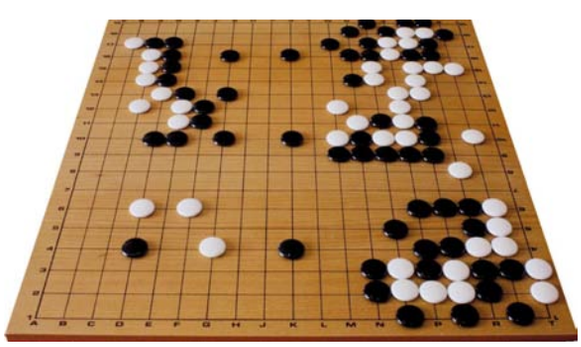
Computing grid :
A computing grid makes distributed calculations possible : it uses the computing power (CPUs, memory, etc.) of thousands of computers to give the impression of a super powerful virtual computer. This system can thus solve major problems that would take too long to solve in a "classical" environment.

Monte-Carlo Method :
Methods for calculating a numerical value through procedures dependent on chance (i.e. probabilistic techniques) are called Monte-Carlo methods. The name was inspired by the games of chance played in the casinos of Monte Carlo.

Parallelisation :
Parallelisation involves the execution of independent tasks simultaneously to minimise the time required to complete all the tasks. In computing, parallelisation often involves breaking down tasks into atomic operations, executed independently of each other. Parallelisation may be achieved on a single computer or by distributing tasks between different computers.

FIGURE 4

MoGo played black and won.
Kim, champion of the formidable US Open 2 days after his defeat by MoGo, later played with a 7-stone handicap and won.



PHOTO

Professor Dong (5th Dan), assessing MoGo's rank. MoGo won all four games against Professor Dong, who played with a 4-stone handicap.