

Appendix II

On the Frontlines of Computer Go: An Interview with Olivier Teytaud of MoGo

By Peter Shotwell
© 2008

At the beginning of the year, I posted an appendix to my Cognitive Psychology article in the Bob High Library on the AGA website. It ended with a discussion of how human learning studies spurred on the Monte Carlo method of programming, which is now being successfully used in many fields. Like the game of Battleship is played, in go the position of the next move is largely based on the win-lose results at the end of the playouts of almost random searches of future moves. These choices are modified by introducing a certain amount of previous 'history,' meaning the factoring in of the proximity to the last move, adding off-line pattern values taken from thousands of pro games, and estimating the amounts that can be captured or escape. If a certain move leads to 80% wins or more in the search tree, it is played, and can lead to usually impressive (or, sometimes disastrous) results.

Invented during the Manhattan Project of World War II and now using enormous search speeds and sometimes multi-core or message-passing parallelization, the Monte Carlo approach is much faster than the 'simple' memory-crunching computers that have beaten the best chess players. It also began outdistancing the 'classical' human knowledge-based go programs when the Upper Confidence Trees (UCT) algorithm was added in 2006.

In using UCT, different moves within the game tree search are treated like 'one-armed bandit' slot machines that return random results—but, unlike in casinos, where the machines are finely tuned, the results on some bandits are better than others. Because of this, the famous Exploration-Exploitation Dilemma arises and choices have to be made between making what seems to be a good move now (by pulling the arm of a previously high-paying bandit), or spending time playing other bandits which might (or might not) produce a better pay-off.

In humans, the process of resolving the Dilemma is called Temporal Difference Learning, which combines the Monte Carlo method with

dynamic programming (DP) ideas (the ‘programming’ is mathematical optimization and has nothing to do with computer programming). In Temporal Difference Learning, MRI studies show how different parts of the brain start working against each other to narrow down our choices about future actions. However, it is important to note that in these studies, the emphasis is not on how the brain makes the best judgment—the traditional approach—but how it reduces its ‘regret’ for making wrong choices.

In the world of ‘thinking’ machines, algorithms called Rapid Action Value Estimations (RAVE) were developed to solve the problem of the Dilemma. And in the business world, Behavioral Targeting algorithms try to figure out profitable ‘moves’ for companies by predicting the behavior of opposing customer ‘players’ who are making choices about such things as what to buy or where to go on the Internet.

Dr. Olivier Teytaud of the University of Paris is one of the developers of MoGo, the first program to beat a professional on 9x9 and the first to achieve a 1-*dan* rank on KGS in 19x19 ten-minute-a-side games. He generously responded to my email questions about what is happening on the front lines of computer go.

MoGo improves everyday. The version of yesterday evening wins with probability 57% against the version of yesterday morning (OK, it's not so nice everyday :-).

Most of the MoGo team are not go players and there is very little math involved. All the efficient programs, as far as I know, use the following elements:

- *A Monte-Carlo simulator for evaluating roughly the quality of a position. This method simulates a random game several times, from an initial position, in order to have a preliminary idea of its value.*
- *An incremental building of a tree of situations, representing possible future states; a main part of the algorithm is the so-called ‘bandit-part,’ which triggers the compromise between exploration (trend to analyze more carefully the situations which are less analyzed than other situations) and exploitation (trend to analyze more carefully the situations which are the most likely).*
- *In all successful programs, the Monte-Carlo part is biased through patterns. In fact, one of the main recent improvements consists in reducing the bias—as we improve the computational power (thanks to parallelization), we must reduce the bias induced by the patterns of the Monte-Carlo part.*

- *In all successful programs also there are correlations between one move and the next move in the Monte-Carlo part. This seems to be absolutely necessary. Some programs include a dependency with respect to the two previous moves. This is an important achievement of the early MoGo team, interestingly analyzed in the Ph.D. thesis of Sylvain Gelly.*
- *At least Leela, MoGo, CrazyStone, and probably some others have an SMP parallelization (the Symmetric Multiprocessing of many computers).*
- *In some programs, some bias is introduced in the tree exploration; we a priori prefer moves which match some known patterns, or some rules (e.g., is it worth considering the possibility of a ko, or an atari?); from this point of view MoGo was quite weak until a recent date, but now it is much better. Mango and CrazyStone are top-level programs from this point of view.*

- *The following elements are less widely used but are also important tricks:*

- *In some programs (e.g. MoGo), a so-called RAVE heuristic approximates the value of a sequence of moves by the value of a permutation of an already analyzed sequence of moves. This part is very efficient in MoGo, and is probably a main reason for the efficiency of MoGo on small boards—for small boards, I guess MoGo is currently the best program whenever we do not use the MPI-parallelization (the Message Passing Interface which keeps the Central Processing Units [the CPUs] from interfering with each other's work).*
- *Some programs also use a parallelization without shared memory. This is far less intuitive but quite efficient, particularly in 19x19. Thanks to this parallelization, MoGo can be quite strong in 19x19; it recently reached 1-dan on KGS with 64 quad-core machines on blitz games (10 minutes per side). We have not yet experimented with longer time settings—the priority for the moment is the ‘merging’ of all the improvements developed by various people into only one version of MoGo.*
- *In 9x9, openings are very important. This part is not yet stable in MoGo, but already provides significant improvement, in particular for fast games. For long time settings, the improvement is negligible, but we're working optimistically on it!*

Many elements are not well understood. Essentially, very important modifications in the Monte-Carlo methods come from trial and error. Even

the UCT part, which was the most clearly understood, is now outperformed by new methods. A particularly disappointing point is that we cannot estimate the quality of our moves: we don't know when we are sure that MoGo is going to choose a good move. This is quite important, as this could provide a tool for choosing the time used for a given move: when we are not sure of a move, we might want to spend a few more minutes on it. Humans spend a lot of time on some moves, and not at all on some other moves—whereas MoGo spends almost exactly the same time on all moves—there is just a regular decrease of time per move during the game. This is a strong weakness in all current implementations.

Some recent directions are as follows:

- First, MoGo is sufficiently strong for benefiting from classical opening books in 19x19. This was false until a recent date.*
- Second, we can now tune MoGo, and probably other strong programs as well, so that it solves Tsumego. This is much easier than tuning it by self-play, because self-play is highly time-consuming, and makes sense only now—when MoGo was 10-kyu, it was meaningless to try to make it solve difficult Tsumego.*
- As well as many programs, MoGo was playing a so-called 'cosmic,' center-oriented style; now, it is much more classical (and stronger). But another trouble, which still holds, is that MoGo is too aggressive. It is quite efficient in killing groups, but it also tries this against groups which are clearly alive. Modifying the Monte-Carlo part might be a good solution for that.*
- Large-scale parallelization is possible in 19x19. MoGo would be much, much, much stronger if we had access to huge machines (technically speaking, to big clusters of SMP with good switches) and this looks like a possibility early this fall.*

To explain further, a multi-core machine has several cores accessing the same memory and there is no need for messages between cores, whereas a cluster is made of processors like Disk Sectors, each of them working on its own memory. Multi-core parallelization is a priori easier, but the number of cores on the same machine is limited to a few cores sharing the same memory because they interfere with each other, whereas a cluster can be made of thousands of cores. The parallelization on a cluster is much more difficult and much more recent in Monte-Carlo go, and can be applied efficiently with much bigger machines. This is the main strength of

MoGo; MoGo was the first code with a very efficient parallelization on clusters. I guess many programs will have such a parallelization soon. MoGo combines both parallelizations producing a cluster of multi-core machines. For example, the cluster used for games against the professional Catalin Taranu was made of 32 machines, each of them having eight cores. The eight cores of each machine work on the same memory, but in order to work together, the 32 machines have to communicate by sending messages. This is very slow on standard networks, but some specialized networks, based on fast cards and fast switches, are much better.

As for how all this developed, as far as I know from the published papers, the basic idea of mixing bandit techniques and Monte-Carlo simulations came from Rémi Coulom (the author of CrazyStone); UCT comes from MoGo and is also now in CrazyStone; Progressive widening (the 'unpruning' of possible moves that had been discarded, sometimes because threats have been uncovered from other searches) has been introduced in CrazyStone and Mango.

The RAVE heuristic comes from MoGo and is incorporated in Leela, but not in CrazyStone or Mango; Correlations between successive moves in the Monte-Carlo part were initially introduced in MoGo and is now also used in CrazyStone, Leela, Mango and all successful algorithms; as mentioned, Multi-node Message-Passing Parallelization has been introduced in MoGo and not yet in CrazyStone, Leela, or Mango; Expert knowledge as a bias in the bandit was first introduced in CrazyStone and is not yet very developed in MoGo, but we are working on it.

I want to point out that we do not consider MoGo as only the result of our work. Instead, it's the result of plenty of contributors, including go players, computer-scientists, mathematicians, from several countries, the authors from Mango, CrazyStone and UCT, and members of the University of Alberta in Canada.

Lastly, MoGo is not just a program for computer-go because we are developing several other applications with the same algorithm. For example, the same techniques as those employed in MoGo can be used in resource management, such as deciding which resource should be used for producing electricity. Choosing the next move in go is analogous to choosing the next reservoir to be used, (such as hydroelectricity, nuclear plants, etc.), and winning a go game is analogous to the saving of money and the reduction of the environmental impact of electricity management.

** * * * **

A free version of MoGo can be downloaded at <http://www.lri.fr/~gelly>

Leela can be bought at <http://www.sjeng.org/leela.html>

See also: <http://www.pascal-network.org/article4.pdf> and
<http://www.lri.fr/~teytaud/crmogo.en.html>

For games of MoGo vs. humans, see <http://www.lri.fr/~teytaud/iago>

For a game between MoGo and CrazyStone, see http://project-oz.com/public_html/article.php?story=20080127044215866

Dr. Teyaud has recently updated the Wikipedia article on computer go at http://en.wikipedia.org/wiki/Computer_Go

For a good, detailed explanation of UCT that I became aware of too late to utilize in this paper, along with a visualization of its playing Othello 'knowing' nothing but the rules, see
<http://www.hvergi.net/2008/06/visualizing-gameplaying-algorithms>

For the names and authors of the almost 100 go programs running on KGS, go to <http://www.weddslist.com/kgs/names.html>

For Temporal Difference Learning, see the Wikipedia article at http://en.wikipedia.org/wiki/Temporal_difference_learning

For MRI research on human learning, see, for example,
<http://neurodudes.com/2006/06/22/softmax-rule-for-exploration-exploitation/>

For a dictionary of computer terms, go to www.webopedia.com

To follow what is happening in the field, subscribe to the computer go email group at
<http://hosting.midvalleyhosting.com/mailman/listinfo/computer-go>