Computer Go: a Research Agenda

Martin Müller

ETL Complex Games Lab
Tsukuba, Japan
mueller@etl.go.jp

Abstract. The field of Computer Go has seen impressive progress over the last decade. However, its future prospects are unclear. This paper suggests that the obstacles to progress posed by the current structure of the community are at least as serious as the purely technical challenges. To overcome these obstacles, I develop three possible scenarios, which are based on approaches used in computer chess, for building the next generation of Go programs.[1]

1 A Go Programmer's Dream

In January 1998, I challenged the readers of the *computer-go* mailing list [27] to discuss future directions for Computer Go:

> Assume you have unlimited manpower at your hand (all are 7-Dan in both Go and programming) and access to the fanciest state of the art computers. Your task is to make the strongest possible Go program within say three years. What would you do?

Many contributors to the ensuing discussion severely criticized all currently used approaches, and advocated the development of revolutionary new techniques. I don't think such wholesale criticism is justified. In this paper I take a close look at the current state of Computer Go, and propose a more systematic use of already available, proven techniques. I claim that this will already lead to substantial progress in the state of the art. Writing programs for Go has turned out to be much more complex than for other games. The way Go programs are developed must adapt accordingly: it is necessary to scale up to larger team efforts.

The paper is organized as follows: Section 2 analyzes the state of Computer Go, identifies some of the strengths and weaknesses of the current generation of programs, and outlines a plan which draws on existing technology but still promises substantial progress within a few years. In section 3, I introduce three development models that have successfully been used in chess, and discuss how to adapt them to Computer Go. Finally, section 4 introduces some promising topics for long-term research.

[1] This article is adapted from a paper in the proceedings of CG'98 [22]

2 The State of Computer Go

To the casual observer, Computer Go may seem to be in fine shape. However, a number of problems threaten the future prosperity of the field. Many of these problems are rooted in the current structure of the Computer Go community.

2.1 The Computer Go Community

Recent years have seen many developments in Computer Go. Good progress has been made in the tournament scene and the internationalization of the field. However, in several ways the field has remained immature: programs are constructed on an ad hoc basis, results are held back for commercial reasons, and the lack of support for new researchers willing to enter the field is a severe problem.

Actually, the situation may be quite similar to other games. Almost ten years ago, John McCarthy wrote about the computer chess community [17]:

> By and large, they come in two flavors: sportsmen and businessmen (makers of commercial chess machines). Neither group is primarily motivated to produce scientific papers explaining how the results were achieved and how others might build upon them for further improvement. Because of these factors, computer chess has not succeeded in its *Drosophila* role and I must entreat the community that computer chess be made more scientific: publication is what we need.

The same kind of critical remarks would be justified for characterizing today's Computer Go scene. However, let's start by looking at some of the bright spots first:

Plus: Active Tournament Scene Several yearly tournaments have been established. The Ing foundation's International Computer Go Congress, started in 1985, continues to attract the elite of Go programs from all over the world. Japanese researchers and companies have organized a number of big international tournaments, such as the FOST cup. The yearly European and North American Go congresses host smaller, local computer championships. On the internet, the Computer Go Ladder [23] allows Go programmers from all over the world to compete with a wide variety of opponents.

Plus: An International Activity Computer Go has become a truly international activity, with serious programs being developed in at least a dozen countries. It is not unusual to see the first five places in a tournament taken by competitors from as many different countries. The total number of programs to participate in tournaments easily exceeds a hundred.

There is an active English language *computer-go* mailing list [27]. The *Computer Go Forum (CGF)* with more than 60 members runs its own Japanese language mailing list and issues a newsletter. The American Go Association's

Computer Go web pages [20] provide an archive of relevant information, including history and game records, and can serve as a starting point for exploring the growing number of web sites devoted to the topic.

There seems to be renewed interest in Go research from the general AI and the computer games community. After the world championship-level performances of programs for chess, checkers, backgammon, Othello and many other games, eyes have turned to Go as the 'final frontier' of computer game research.

Minus: Lack of Support for New Researchers Few individuals or institutions have enough resources to subscribe to a full-scale Go programming effort. Indeed most new Go programmers have to start almost from scratch. Because of the overhead in getting started, it is very hard for a smaller project, such as a masters thesis, to make a significant contribution.

Given the complexity of the task, the supporting infrastructure for writing Go programs should offer more than for other games such as chess. However, it is far inferior. The playing level of publicly available source code [8, 15, 20], though improved recently, lags behind that of state of the art programs. Quality publications are scarce and hard to track down. Few of the top programmers have an interest in publishing their methods. Whereas articles on computer chess or general game tree search regularly appear in mainstream AI journals, technical publications on Computer Go remain confined to hard to find proceedings of specialized conferences. The most interesting developments can be learned only by direct communication with the programmers and never get published.

2.2 Current State of Go Programs

Computer Go constitutes a formidable technical challenge. Existing programs suggest the following difficulties:

- A competitive program needs 5-10 person-years of development.
- A typical program consists of 50-100 modules.
- The weakest of all these components determine the overall performance.
- A number of standard techniques have emerged. However, no single program incorporates a large fraction of all currently existing successful Computer Go methods.
- The best programs *usually* play good, master level moves, but their performance level over a *full* game is much lower because of the remaining blunders.

Let me discuss the last two problems in some detail:

Minus: Incompleteness of Existing Programs Unfortunately, there is no single program that incorporates most of the currently existing successful Computer Go techniques. A next generation program would need to recreate and integrate most of these individual capabilities.

It is easy to see why the currently predominant single-person projects are really inadequate for Go: the sheer number of necessary components. Even assuming only one month for each module, components of a reasonably complete program take four to five years to build, plus considerable time for system-level testing and integration.

Minus: Disappointing Sustained Performance The difference in first-play versus sustained long-run performance of programs against human players is drastic: programs typically do well in their first game against an opponent inexperienced in playing computers. For example, the *Handtalk* program has won Ing's challenge matches taking a 11 stone handicap against top amateur players, and has beaten a 1-dan player on even in an exhibition game at the FOST cup. These games are impressive achievements. I urge the sceptics among the readers to carefully study the games, which are available on the AGA site [20].

The other side of the story is that, if allowed a few practice games, humans soon spot a program's weaknesses and become able to exploit them. The same program that once beat the 1-dan regularly loses to a well-prepared 5-kyu player, even with huge handicaps of up to 20 stones.

What to Blame? The Model or its Implementations? What is the reason behind the uneven performance of today's programs? How can Go programs look so good in one game and so pathetic in the next? One theory is that there is a fundamental problem with the underlying models. In this view, current Go programs are not able to capture the true spirit of Go: they may play good-looking moves, but do so without any *real understanding* of the game, which inevitably shows sooner or later. The alternative view is that the current model is basically sound and sufficient, but programs suffer from incomplete or buggy implementations. I will describe some experiments designed to test these theories later. First, I will briefly outline a model for current Go programs. A more in-depth treatment can be found in the long version of this article [22].

2.3 A Model of Go Program Components

David Fotland's *Computer Go Design Issues* [11] lists about sixty components of current Go programs, and can be considered as defining a standard model encompassing most current state of the art programs. A program implementing this model as complete and technically accurate as possible would serve as an interesting milestone by itself. Furthermore, such a program would allow a more meaningful analysis of the model's strengths and weaknesses than is currently possible. I will briefly discuss each of the following development tasks for a Go program:

- Theoretical foundations and game tree search methods
- Knowledge representation and data structures
- Search methods

- Global move decision
- Software engineering and testing
- Automatic tuning and machine learning

Theoretical Foundations and Game Tree Search Methods Theoretical techniques applicable to Computer Go range from abstract mathematics for group safety, endgame calculation and *ko* evaluation [6, 7, 25] to Go-specific knowledge such as the *semeai formula*. A detailed survey of theories relevant for Computer Go is given in [19].

Standard game tree-searching methods are well established for goal-oriented tactical search in Go. In addition, new search methods such as proof-number search [1] have been successfully applied in at least one commercial program. The many potential benefits offered by theory have only partially been exploited in current programs.

Knowledge Representation and Data Structures Most programs use a hierarchical model for board representation. Low-level concepts are blocks of adjacent stones and connections or *links* between stones. Chains, groups and territories are higher-level concepts built from the primitives. Pattern matching is used to find candidate moves. Knowledge representation has been the main focus of Computer Go research to date, and has reached a sophisticated level.

I expect that the quality of knowledge incorporated in programs will gradually be refined. The quantity of knowledge is rising dramatically due to large-scale pattern learning methods, which are becoming increasingly popular [9, 14, 13, 29, 24]. However, it is unclear how computer-generated pattern databases can approach a quality comparable to human-generated ones. For comparison, it would be fascinating to develop a large corpus of human Go knowledge, to try to identify and encode the pattern knowledge of Go experts.

Search Methods [2] Three types of search are commonly used in Computer Go: single-goal, multiple-goal, and full-board search. Specialized searches that focus on achieving a tactical goal constitute some of the most important components of current Go programs. One use of goal-directed search is to propose locally interesting moves to the global move decision process. Because of the complex evaluation and high branching factor of Go, full-board search has to be highly selective and shallow.

Global Move Decision There is a great variety of approaches to the problem of global move decision in Go. No single paradigm, comparable to the full board minimax search used in most other games, has emerged. Most programs use a combination of the following methods:

- Static evaluation to select a small number of promising moves

[2] This topic is discussed in much greater detail in [22]

- Selective search to decide between candidate moves
- Shortcuts to play some *urgent* moves immediately
- Recognition and pursuit of temporary goals
- Choice of aggressive or defensive play based on a score estimate

I expect experimentation to continue, without any clear preference or standard method emerging. Methods based on combinatorial game theory have the potential to replace more traditional decision procedures.

Software Engineering and Testing A competitive Go program is a major software development project. Software quality can be improved by using standard development and testing techniques [18]. A wide variety of game-specific testing methods are available, including test suites, auto-play and internet-based play against human opponents. However, many tournament games are still decided by blunders that seem to originate from programming errors.

It is hard to judge objectively, but I suspect there is a lot of room for improvement in this area. Most leading programs have been in continuous development for ten or more years. Many of these programs may be reaching a level of internal complexity where it is difficult to make much progress. Originally designed for machines a thousand times smaller and slower, programs have grown layer upon layer of additions, patches and adjustments. Some programs have been rewritten from scratch in the meantime, but this is a daunting and extremely time-consuming task [28].

Automatic Tuning and Machine Learning Machine learning techniques are used more and more in Go programs [9, 10], with the previously mentioned large-scale pattern learning methods as maybe the most prominent example. I predict that the applications of machine learning techniques in Computer Go will increase further, for example for fine-tuning the performance of complex programs with many components.

3 A Research Plan for Computer Go

What is the real limit that current models impose on the performance of Go programs? Should research focus on developing better models, or on improving the implementation of current ones? To answer these important questions, I propose the following three lines of research and development:

- Detailed analysis of current programs' errors and limitations
- A *Dreihirn* experiment
- Large scale Go programming projects

The first two methods are designed to better understand the practical problems of current programs. The third proposal addresses testing the inherent limits of current technology.

3.1 Detailed Error Analysis

Detailed error analysis of current programs can draw upon a wealth of available game records [20]. Many classifications of mistakes can be made. One possible distinction is between *lack of basic understanding* and *lack of efficiency*. Lack of basic understanding can be defined as the failure to identify the current focus of a game. Examples are attacking or defending the wrong group of stones, ignoring threats or double threats, making wrong life and death judgments, or playing bad shape because of a lack of pattern knowledge.

Efficiency errors are less drastic individually but have a large cumulative effect. Mistakes belonging to this category are: making overconcentrated shapes, taking *gote* when a *sente* move is available, or achieving the correct main goal without optimizing secondary effects as well.

Research should aim to develop automatic methods for performing such error analyses by using statistical techniques and developing suitable test suites.

3.2 A Dreihirn Match for Go

In Ingo Althöfer's series of *Dreihirn* chess experiments [2, 16], a team of two chess computers supervised by a human *boss* has achieved strong results against chess grandmasters and programs. The team played markedly better than each individual program, even though the boss was a relatively much weaker player. The human supervisor was able to select a promising overall direction of play and avoid some dubious computer moves. Althöfer has performed similar experiments with one or more computers running in k-best mode, and a human operator selecting a move from that list [2, 3, 5, 4].

In [4], Althöfer concludes:

> The match win against Yusupov demonstrated once more that in chess an interactive man-machine system may perform much better than each of its components separately.

I propose performing a Dreihirn experiment in Go, with a team of several Go programs supervised by a *strong* human player. At each move, the human selects one of the moves proposed by the programs. This team is tested against a variety of opponents, including other programs and humans of different strengths. Such a test can serve to establish an upper limit of current program performance, and show whether the uneven play is due more to individual bugs in the implementations or due to more fundamental limitations of all current programs. If a series of games is played, the test would also show if human opponents can adapt as quickly to such a system as they seem to adapt to each individual program.

3.3 Outline of An Architecture for Large Scale Go Projects

From the beginning, most Computer Go projects have consisted of a single programmer, with occasional assistance from scientists and Go experts. In recent years, a few commercial programs have been developed on a slightly bigger scale,

with small teams of programmers and managers working on the Go engine and user interface.

I believe that the scale of these projects is not large enough, and that projects an order of magnitude larger are necessary to produce a qualitative jump in performance. Section 2 has identified a list of tasks required to implement a complete Go program based on the current standard model. However, implementing a successful large scale Go project requires a series of preliminary steps:

- Secure an existing state of the art program to build on, including an easy to use basic Go toolkit.
- Modify the program to increase its usability in a multi-programmer environment.
- Describe the model underlying the program in detail.
- Extensively document and structure the source code.
- Define an effective communication method between team members.
- Implement a well-defined process for subtask assignment, code integration and testing.

I think that these steps represent necessary preparation for all three types of large scale projects discussed below.

3.4 Three Proposals for Large Scale Go Projects

In chess, three approaches have been taken in recent years that may serve as an inspiration for Go:

- Large company funded teams (Deep Blue)
- Public domain source code (GNU chess, Crafty)
- University projects (many)

Plan 1: Large Scale Commercial Project The Deep Blue chess project represents a large-scale effort, one order of magnitude larger than typical competitive chess programs. Its success rests on two pillars: on the technical side, it is a complete, mature system, the result of skilful engineering firmly based on a large amount of previous research. On the organizational and financial side, the Deep Blue project was backed by a large company with an interesting new marketing strategy. Computer chess was chosen as an advertising vehicle because it represents an attractive topic that is tied to deep myths about human and machine intelligence.

Would a similar alliance of research and big business make sense in Go? Who would be a potential sponsor, and what would be their interest? In my view it would be a world-class company with a strong interest in the Asian market, and an ambition to create or reinforce their image as an intellectual leader. The company would profit mainly from the publicity generated by exhibition games, not from sales of Go software. Given the high regard for Go as an intellectual sport, it seems possible to attract a level of attention comparable to that of the

chess matches, at least in East Asia. In Go, what is an achievable goal that will fascinate the masses? World championship level play still seems far in the future. Yet a program playing at a sustained 1-dan level, which can beat professional players on 9 stones handicap, will be perceived as an intellectual achievement at least equal to that of the chess machines. Is it possible in the near future? Let's try!

Plan 2: Public Domain Go Project Source code for more than a dozen chess programs is readily available on the internet [26]. The two best-known of these programs, GNU chess and Crafty, have active user groups which are testing, discussing or directly improving the program.

In Go, several public domain projects have been attempted over the years. Until recently, none of these have resulted in a tournament level program. Perhaps fueled by the open source/GNU/Linux revolution, there is now a renewed interest in such projects [15, 8].

The characteristics of a public domain Go program are quite different from a funded project and include:

- Greater fluctuation of team members, unpredictable team size
- Driven by group dynamics rather than financial incentive
- Low development cost
- Difficult moderation and integration tasks

The project goal could be to develop a noncommercial, research-oriented tool. The program structure should allow small or medium-scale experiments, for example in machine learning, to make use of a state of the art Go engine in a reasonably straightforward manner. A less ambitious approach would aim at developing only a library of commonly used functions.

Plan 3: University Research Project Many of the strongest chess programs are developed at universities. The situation in Go is comparable: about half of the current top 20 Go programs have started as student projects. An advantage of student projects is that relatively little funds are required, and students can combine programming work with their research.

The main challenge of this approach is to assemble a large group of talented students and keep their efforts coordinated over a number of years. Given the current distribution of Go players, a large-scale university Go project would probably be feasible only in an Asian country. On the other hand, it is easy for a student or university researcher to contribute to a Plan 2 public domain project, so this may be a more viable approach for most.

4 Some Issues for Long-term Research in Computer Go

Compared to the complex reasoning processes of human Go experts, the models incorporated in current Go programs are severely limited. A goal of long-term

research could be to narrow this gap, either by building more sophisticated models or by deriving human-like reasoning capabilities from simple models. Such research could include modeling the high level full-board plans of human players, or advanced Go concepts such as *aji*, *korikatachi* or *sabaki*. Another direction for research is evaluation from first principles, using only search and learning, without relying on human-engineered heuristics.

Long-term machine learning topics are automatic derivation of sophisticated Go concepts from first principles, or the learning of patterns along with suitable contexts for their application.

Yet another research topic are applications of combinatorial game theory. As a framework for Computer Go, combinatorial game theory has several advantages compared to the standard minimax game-playing model. However, the finer points of this theory are as good as unknown outside the small combinatorial games community. Several of the tools provided by this theory are well suited for analyzing Go, and should be used in more Go programs. For example, the combinatorial game-theoretical method of *thermography* is able to very naturally model fundamental Go concepts such as *sente*, *one-sided sente* and *gote* [7]. Recent progress in theory and in practical algorithms for thermography [7, 25, 21] also provides effective and sound methods for comparing the relative values of *ko* and non-*ko* moves, and the evaluation of *ko* threats.

Further work is needed for handling incomplete local game trees and developing selective search strategies within a combinatorial game framework [12]. An important research problem is to generalize the precise concepts of combinatorial game theory to work in a heuristic setting, in analogy to the minimax-based heuristic game tree search used in other games.

5 Summary

Computer Go has enjoyed a boom in recent years, but its progress is hampered by problems in the structure of the Computer Go community. An analysis of the current state of Computer Go and a comparison with computer chess methods indicates promising directions for research, both short-term and long-term. To overcome the lack of critical human resources, Computer Go would benefit from large scale projects, which have already succeeded in chess.

References

1. L.V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, Maastricht, 1994.
2. I. Althöfer. On timing, permanent brain and human intervention. In J. van den Herik, editor, *Advances in Computer Chess*, volume 7, pages 285–297. University of Limburg, Maastricht, 1994.
3. I. Althöfer. A symbiosis of man and machine beats grandmaster Timoshchenko. *ICCA Journal*, 20(1):40–47, 1997.
4. I. Althöfer. List-3-Hirn vs. grandmaster Yusupov part 2: Analysis. *ICCA Journal*, 21(2):131–134, 1998.

5. I. Althöfer. List-3-Hirn vs. grandmaster Yusupov part1: the games. *ICCA Journal*, 21(1):40–47, 1998.
6. D.B. Benson. Life in the game of Go. *Information Sciences*, 10:17–29, 1976. Reprinted in Computer Games, Levy, D.N.L. (Editor), Vol. II, pp. 203-213, Springer Verlag, New York 1988.
7. E. Berlekamp. The economist's view of combinatorial games. In S. Levy, editor, *Games of No Chance: Combinatorial Games at MSRI*. Cambridge University Press, 1996.
8. D. Bump. Gnugo. http://www.gnu.org/software/gnugo/gnugo.html, 1999.
9. T. Cazenave. *Systeme d'Apprentissage Par Auto-Observation. Application au jeu de Go*. PhD thesis, University of Paris, 1997. www-laforia.ibp.fr/~cazenave/-papers.html.
10. M. Enzenberger. The integration of a priori knowledge into a Go playing neural network. cgl.ucsf.edu/go/Programs/NeuroGo.html, 1996.
11. D. Fotland. Computer Go design issues. www.usgo.org/computer/text/-designissues.text, 1996.
12. K.Y. Kao. *Sums of Hot and Tepid Combinatorial Games*. PhD thesis, University of North Carolina at Charlotte, 1997.
13. T. Kojima. *Automatic Acquisition of Go Knowledge from Game Records: Deductive and Evolutionary Approaches*. PhD thesis, University of Tokyo, 1998.
14. T. Kojima, K. Ueda, and S. Nagano. An evolutionary algorithm extended by ecological analogy and its application to the game of go. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence(IJCAI-97)*, pages 684–689, 1997. www.brl.ntt.co.jp/people/kojima/research.
15. J. Lim. Baduki. http://soback.kornet.net/~artist/baduk/baduki.html, 1999.
16. C. Lutz. Report on the match 3-Hirn vs. Christopher Lutz. *ICCA Journal*, 19(2):115–119, 1996.
17. J. McCarthy. Chess as the drosophila of ai. In T. A. Marsland and J. Schaeffer, editors, *Computers, Chess, and Cognition*, pages 227–237. Springer Verlag, New York, 1990.
18. S. McConnell. *Rapid Development*. Microsoft Press, 1996.
19. M. Müller. Game theories and Computer Go. In *Proc. of the Go and Computer Science Workshop (GCSW'93)*, Sophia-Antipolis, 1993. INRIA.
20. M. Müller. American Go Association Computer Go Pages. www.usgo.org-/computer, 1997.
21. M. Müller. Generalized thermography: A new approach to evaluation in Computer Go. In H. Iida, editor, *Proceedings of IJCAI-97 Workshop on Computer Games on Using Games as an Experimental Testbed for AI Research*, pages 41–49, Nagoya, 1997.
22. M. Müller. Computer Go: a research agenda. In J. van den Herik and H. Iida, editors, *Computers and Games. Proceedings CG'98*, number 1558 in Lecture Notes in Computer Science, pages 252–264. Springer Verlag, 1998.
23. E. Pettersen. The Computer Go Ladder. cgl.ucsf.edu/go/ladder.html, 1994.
24. S. Sei and T. Kawashima. Move evaluation tree system. Complex Games Lab Workshop. http://www.etl.go.jp/etl/divisions/~7236/Events/workshop98/, 1998.
25. W. Spight. Extended thermography for multiple kos in go. In J. van den Herik and H. Iida, editors, *Computers and Games. Proceedings CG'98*, number 1558 in Lecture Notes in Computer Science, pages 232–251. Springer Verlag, 1998.
26. P. Verhelst. Chess program sources. www.xs4all.nl/~verhelst/chess/sources.html, 1997.

27. L. Weaver. COMPUTER-GO Mailing List Archive. www.hsc.fr/computer-go, 1998.
28. B. Wilcox. Chess is easy. Go is hard. Computer Game Developers Conference. home.sprynet.com/sprynet/vrmlpro/cgdc.html, 1997.
29. H. Yoshii. Move evaluation tree system. Complex Games Lab Workshop. http://www.etl.go.jp/etl/divisions/~7236/Events/workshop98/, 1998.