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 A Go Programmer's Dream

In January 1998, I challenged the readers of the *computer-go* mailing list [23] 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 of the answers I received 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 been used in chess, and discuss how to adapt them to Computer Go. Finally, section 4 introduces promising topics for long-term research.

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.

The Tournament Scene Several yearly tournaments have been established. Two world championships, the Ing foundation's International Computer Go Congress and the newer FOST cup, continue to attract the elite of Go programs from all over the world. The yearly European and the North American Go congresses host smaller, local computer championships. The internet-based Computer Go Ladder [21] allows Go programmers from all over the world to compete with a wide variety of opponents.

An International Activity While human Go players are still concentrated mainly in Asia, 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. Several strong new commercial programs have been developed, and the total number of programs to participate in tournaments easily exceeds a hundred.

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

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 [18] is far below 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 the proceedings of specialized conferences. The most interesting developments can be learned only by direct communication with the programmers and never get published.

2.2 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.
- It is the weakest of all these components which determine the overall performance.
- 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.
- A number of standard techniques have emerged. However, there is no single program which incorporates most of the currently existing successful Computer Go techniques.

Let me discuss the strengths and weaknesses of programs in some detail.

Special Strengths of Current Programs Several of the leading programs do one specific task better than the others. For example, *Handtalk* excels in overall integration, playing good shape, and in knowledge about group attack and defense. *Go 4++*, on the other hand, is most efficient in taking territory, plays the fewest unnecessary 'wasted' moves per game, and has an extensive special purpose *joseki* book for opening move sequences. Further, *Go Intellect* is a mature program with strong tactical fighting and overall Go knowledge, while *GoTools* [25] is a high dan-level Life and Death solver specialized for completely surrounded areas.

Incompleteness of Existing Programs Unfortunately, there is no one 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 single-person projects are inadequate: the sheer number of necessary components. Even assuming only one month for each module, a reasonably complete program will take four to five years to build, even without considering testing and system integration.

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, world champion Handtalk has won Ing's 11 stone challenge matches against high dan-level human players, and has beaten a 1-dan player without handicap in an exhibition game at the FOST cup. However, if allowed a few practice games, humans soon spot and exploit a program's weaknesses. The same program that once beat the 1-dan regularly loses against a well-prepared 5-kyu player, even when receiving huge handicaps of up to 20 stones.

2.3 What to Blame? The Model or its Implementations?

What is the reason behind the uneven performance of programs? How can Go programs look so good on one day and so pathetic on the next? One theory is that there is a fundamental problem in 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.

2.4 A Model of Go Program Components

Fotland's 'Computer Go design issues' lists about sixty components of current Go programs [8], and can be considered as defining a 'standard model'. State of the art programs contain many of the modules described in Fotland's list. I will use the following simplified classification of development tasks for a Go program:

- Mathematical foundations and Go theory
- Knowledge representation and data structures
- Search methods
- Global move decision
- Software engineering and testing
- Automatic tuning and machine learning

I will briefly discuss the issues for each group of tasks, describe the current state of their implementation, and point out promising areas for short-term research and development. A program implementing this 'standard model' as completely and technically accurate as possible would serve as an interesting milestone and allow a more meaningful analysis of its strengths and weaknesses than is currently possible.

Mathematical Foundations and Go Theory Theoretical techniques applicable to Computer Go range from abstract mathematics for group safety and endgame calculation [3, 4] to Go-specific knowledge such as the *semeai formula*. [16] gives a detailed discussion.

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 applied 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 focus of the majority 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 [5, 13, 12]. However, it is unclear how computer-generated pattern databases can reach 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 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 are some of the most important components of current Go programs. A major advantage of goal-directed search over full-board search is that evaluation consists only of a simple test, which is much faster than full territory evaluation. One use of goal-directed search is to propose locally interesting moves to a selective global move decision process. I expect the use of goal-directed minimax searches to expand widely in volume and scope.

Single-goal search uses standard game tree searching techniques for finding the tactical status of blocks, chains, groups, territories, or connections. Knowing this status improves the board representation and is a precondition for creating a meaningful scoring function.

Examples for the targets of single-goal search are given in the table below. Current programs implement many but not all of these goal-oriented searches. A complete implementation of all basic single-goal searches seems to be a straightforward development task.

Target	Reference
Single block capture	[11]
Life and death	[25, 14]
Connect or cut	[7]
Eye status	-
Local score	(Goliath)
Safety of territory	[20]
Semeai	[7]

Multiple-goal search tries to achieve a combination of basic goals, such as capturing at least one of a set of blocks. The implementation of such tasks is rudimentary in most programs. Simple double threats such as *double atari* are usually built in as special cases. However, many standard Go strategies can be understood as more general double threats [17]. The following table lists a few common themes. [9] describes a more sophisticated architecture for multipurpose strategic planning.

Target(s)	Goal(s)
Multiple blocks	save all blocks
Territory boundary	capture *or* break through
2 groups	splitting and leaning attacks
Group	attack *and* make territory
Group	live locally *or* break out
Group	make eyes *or* win semeai

Searching each possible goal combination leads to combinatorial explosion in the number of searches. Heuristics can be used to select promising goal combinations for search. I expect a lot of progress on this problem over the next few years.

Full board search seeks to maximize the overall evaluation. Because of the complex evaluation and high branching factor of Go, full-board search has to be highly selective and shallow. I expect the use of full board evaluation to increase steadily along with improving hardware, but without playing the same dominating role it has played for other games.

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
- Selective search to decide between candidate moves
- Shortcuts to play some 'urgent' moves immediately
- Recognition and following 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 model' emerging. Methods based on combinatorial game theory (section 4.1) 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 [15]. A wide variety of game-specific testing methods are available, including test suites, auto-play and internet-based play against human opponents.

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 [24].

Automatic Tuning and Machine Learning Machine learning techniques are currently used in only a few programs [5, 6]. However, parameter tuning and book learning techniques seem to be in more frequent use. I predict that the applications of machine learning techniques in Computer Go will increase, 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 on the performance of Go programs imposed by current models? Should research focus on developing new 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
- A *Dreihirn* experiment
- Large scale Go programming projects

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

3.1 Detailed Error Analysis

Detailed error analysis of current programs can draw upon a wealth of available game records [18]. Many classifications of mistakes are possible. For our purposes, it may be sufficient to assign errors to one of two broad groups: 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, ignoring threats or double threats, or making wrong life and death judgments.

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 considering secondary effects. An example of the latter kind of problem is saving a group by connecting it on a neutral point, instead of living more profitably by surrounding enough territory to make two eyes. Another example of efficient play is *kikashi*: playing profitable forcing moves before going back to make a neccessary but unattractive defense.

Research along these lines will aim to develop automatic methods for performing such analyses, by using statistical techniques and developing suitable test position collections.

3.2 A Dreihirn Match for Go

In the *Dreihirn* chess games [2], a team of two chess computers supervised by a human 'boss' has achieved strong results against chess grandmasters. 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 typical computer missteps.

I propose a comparable 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 adapt to each individual program's weaknesses.

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 either scientists or 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 long 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

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 sound 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 [22]. 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. So far, none of these has resulted in a tournament level program. Recently, there seems to be renewed interest in such a project, which has generated a large amount of messages on the mailing list.

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

- Greater fluctuation of team members
- Less individual commitment, lower work intensity
- 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 profit from a state of the art Go engine. 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 their parts of the overall 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.

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 close this gap, either by building more sophisticated models or by deriving human-like reasoning capabilities from simple models.

Another fascinating topic is modeling the high level full-board plans of human players, or advanced Go concepts such as *aji*, *korikatachi* or *sabaki*. Therefore, one 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, addressed in the next subsection, is the application of combinatorial game theory to Computer Go.

4.1 Combinatorial Game Theory for Computer Go

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 ouside the small combinatorial games community. Several of the tools provided by this theory are well suited for analyzing Go and should be used more in Go programs.

For example, the method called *thermography* is able to model fundamental Go concepts such as *sente* and *gote* very naturally [4]. Thermography computes the temperature of each local situation, which is a measure of move urgency.

Comparing the thermographs before and after a move yields an optimal temperature range for each move. These ranges may differ dramatically for both players, for example in the case of *one-sided sente* moves. Using such an analysis, programs will be able to follow the standard Go strategy of keeping *sente* moves in reserve as potential *ko* threats. The theory is able to determine precisely how long the *ambient* temperature of a game remains high enough to prevent an opponent's *reverse sente* move.

Another important concept from combinatorial game theory is *reversibility*, which allows a player to make many moves based on a local consistency argument, without any full-board analysis. The computational advantages of this idea are immediate. Thermography introduces the stronger notion of *thermographic reversibility* by which a further reduction of search can be achieved [4].

Recent research by Kao addresses handling incomplete local game trees and selective search strategies within a combinatorial game framework [10]. An important research question is to generalize the precise concepts of combinatorial game thoery to work in a heuristic setting, in analogy to the heuristic game tree search based on minimax used in other games.

4.2 Handling of Ko Fights

Ko fights are considered the most complex phase of the game, and are handled poorly by current programs. Progress in theory and in practical algorithms for thermography [4, 19] provides effective and sound methods for comparing the relative values of *ko* and non-*ko* moves. This framework also allows the evaluation of possible *ko* threats.

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 indicates promising directions for research, both short-term and long-term. To overcome the lack of critical human resources, Computer Go would benefit from the same kind of larger scale projects that have 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. A symbiosis of man and machine beats grandmaster Timoshchenko. *ICCA Journal*, 21(1):40–47, 1997.
3. 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.

4. 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.
5. 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.
6. M. Enzenberger. The integration of a priori knowledge into a Go playing neural network. cgl.ucsf.edu/go/Programs/NeuroGo.html, 1996.
7. D. Fotland. Knowledge representation in The Many Faces of Go. Report posted on internet newsgroup rec.games.go, 1993. Available by ftp from igs.nuri.net.
8. D. Fotland. Computer Go design issues. www.usgo.org/computer/text/-designissues.text, 1996.
9. S. Hu and P. Lehner. Multipurpose strategic planning in the game of Go. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1048–1051, 1997.
10. K.Y. Kao. *Sums of Hot and Tepid Combinatorial Games*. PhD thesis, University of North Carolina at Charlotte, 1997.
11. A. Kierulf. *Smart Game Board: a Workbench for Game-Playing Programs, with Go and Othello as Case Studies*. PhD thesis, ETH Zürich, 1990.
12. T. Kojima. *Automatic Acquisition of Go Knowledge from Game Records: Deductive and Evolutionary Approaches*. PhD thesis, University of Tokyo, 1998.
13. 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.
14. J. Kraszek. Heuristics in the life and death algorithm of a Go-playing program. *Computer Go*, 9:13–24, 1988.
15. S. McConnell. *Rapid Development*. Microsoft Press, 1996.
16. M. Müller. Game theories and Computer Go. In *Proc. of the Go and Computer Science Workshop (GCSW'93)*, Sophia-Antipolis, 1993. INRIA.
17. M. Müller. *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory*. PhD thesis, ETH Zürich, 1995. Diss.Nr.11.006.
18. M. Müller. American Go Association Computer Go Pages. www.usgo.org-/computer, 1997.
19. 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.
20. M. Müller. Playing it safe: Recognizing secure territories in Computer Go by using static rules and search. In H. Matsubara, editor, *Proceedings of the Game Programming Workshop in Japan '97*, pages 80–86, Computer Shogi Association, Tokyo, Japan, 1997.
21. E. Pettersen. The Computer Go Ladder. cgl.ucsf.edu/go/ladder.html, 1994.
22. P. Verhelst. Chess program sources. www.xs4all.nl/~verhelst/chess/sources.html, 1997.
23. L. Weaver. COMPUTER-GO Mailing List Archive. www.hsc.fr/computer-go, 1998.
24. B. Wilcox. Chess is easy. Go is hard. Computer Game Developers Conference. home.sprynet.com/sprynet/vrmlpro/cgdc.html, 1997.
25. T. Wolf. Investigating tsumego problems with RisiKo. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2*. Ellis Horwood, 1991.