AI Techniques Used in Computer Go

Jay Burmeister and Janet Wiles
Schools of Information Technology and Psychology
The University of Queensland, Australia
jay@it.uq.edu.au
http://www.psy.uq.edu.au/~jay/

Abstract

This paper surveys the most competitive Computer Go programs. It is targeted generally at the Cognitive Science community as an introduction to Computer-Go, given the growing importance of Go as a domain for Cognitive Science research, and specifically at Computer Go programmers (or prospective Go programmers). We survey the best current Computer Go programs, Handtalk, Go4^{++}, Many Faces of Go, Go Intellect, and Explorer. We summarise the AI techniques used, key challenges that must be faced, and issues involved in game tree search, showing why Computer Chess techniques do not translate well to the Go domain.

1. Competitive Go Programs

Go[1] is one of the last formal game domains in which computer performance is not competitive against even moderately good human players. There are several annual Computer Go tournaments, notably the FOST Cup which promises JPY 2,000,000 (around AUD $23,000) for 1st place, as well as the unclaimed Ing Prize TWD $40M (around AUD $1.9M) for the first Go program to beat a professional player in a best-of-seven match without handicap.

The earliest work in Computer Go using Go as a research domain was in 1962 although the first complete game played by a program was in 1968 (Zobrist, 1970). Computer Go became established as a field in the 1980's when Computer Go tournaments began and the first commercial programs were released, and has since flourished in the 1990s. The top programs currently active in Computer Go competitions include Explorer, Go Intellect, Go4^{++}, HandTalk, and The Many Faces of Go, and are currently ranked in the range of approximately 4-8 kyu.

2. Game-Tree Search in Go

The typical AI approach to playing 2 player perfect information games is to search the game-tree in order to decide what move to play. Standard game-tree search comprises four components: 1. representation of game states, 2. generation of possible moves, 3. determination and recognition of goal states, and 4. a static evaluation function to determine the relative merit of states. An effective means of pruning the game-tree (e.g., alpha-beta) enhances the performance of programs.

The game-tree approach can be very successful, as shown by chess programs which are based on full-width game-tree search typically using alpha-beta pruning, and challenge even the world champion[2]. In this section we examine the four components of game-tree search from a Computer Go perspective.

2.1 State Representation

From a perfect information perspective, a Go board is a 19x19[3] grid with each intersection point being either empty or occupied by either a black or a white *stone*. The size of the state-space (i.e., the number of possible positions) can be estimated[4] at 3^{361} (or 10^{172}) compared to approximately 10^{50} for chess and 10^{30} for othello (Allis, 1994). The size of the game-tree (i.e., the number of possible games) is estimated[5] to be between 10^{575} and 10^{620} compared with estimates of 10^{123} for chess and 10^{55} for othello (Allis, 1994).

Solely representing the state space in terms of the 19x19 grid is too low-level for either humans or machines to use effectively, due to the combinatorial size of the space. The next level of description is to form orthogonally adjacent stones into *strings* (or *chains*). All programs collect strings into larger units, however,

1. Go is a very popular board game known as Igo in Japan, Wei-Ch'i in China and Taiwan, and Baduk in Korea.
2. Deep Blue beat the world champion Garry Kasparov in a best-of-six series in May 1997.
3. Go is also played on 9x9 and 13x13 boards.
4. Not all positions would be legal; also a history must be maintained for each board position to avoid illegal ko repetitions.
5. The estimate of 10^{575} is based on an average branching factor of 200 and average game length of 250 ply (i.e. 200^{250}). The size of the game-tree may be as high as 10^{620} if the average game is considered to be 300 ply (Burmeister & Wiles, 1995).

there is no generally accepted process - even for expert human players - for grouping strings into larger units. Depending on their theory of Go, programmers develop their own heuristics for assessing when strings are effectively linked together (called variously *groups* or *blocks*).

In addition, the appropriate level of representation can vary depending on the sub-task being performed at the time, for example, *tactical* analysis, *life-and-death* analysis, or assessing *territory.*

2.2 Move Generation

Players take it in turns to place a stone on an empty intersection point (including corners and edges), subject to a rule prohibiting suicide (self capture) and position repetition (ko). Like chess, only a subset of all legal moves in Go make sense in the context of a given position. The average branching factor in Go is large, being around 6 times that of chess (200 compared to 35, Burmeister & Wiles, 1995).

Note that this branching factor takes into account the whole board. In some situations, only local considerations are important. For example, goal-directed search is used to read out *ladders*, which usually have only 1-2 possible moves, but may be up to 60 ply[6] deep.

Move generation in practise is a complex issue: see section 3.4.

2.3 Goal States

The ultimate goal of Go is to acquire more territory than the opposing player. There are two methods for acquiring territory: by surrounding empty points with walls of stones, and by *capturing* opponent strings by surrounding them with stones. In practice it is difficult to determine goal states because territory is gained by accumulation (unlike chess where the final goal of checkmate is sudden-death and is focussed on one piece). Since an accurate determination of territory is difficult until late in the game, heuristic estimations are usual. Such heuristics usually incorporate components or sub-goals (e.g., material advantage in chess) which are indicative of the potential to secure territory (e.g., life-and-death of groups, and *influence*).

The game ends when both players pass in turn. Players usually pass when no move will increase their score and/or every move will decrease their score. In practice, determining the end of the game (i.e., when to pass) is difficult. In human games, disputes over the life-and-death of groups during scoring are settled by continuing play until resolution is reached. In Computer Go tournaments, if programs have a disagreement over the score which cannot be solved by their operators, the game is scored by a tournament official.

2.4 Evaluation Function

The life-and-death status of groups is an important consideration in judging the merit of Go positions. Determining life-and-death is time consuming and is typically achieved by tactical search (see Section 3.5) or life-and-death search (see Section 3.6). Another interesting complication to evaluating the life-and-death of a group is that it may be necessary to evaluate the entire board position: if a group is alive in ko (i.e., it must win a ko to stay alive), it is necessary to evaluate the number and size of *ko threats* a player has in comparison to the opponent to determine whether the group is dead or alive. Ko analysis can also be further complicated by the existence of double or triple ko situations.

The results of evaluation are sometimes uncertain because definite determination of life-and-death may not be possible within the constraints placed on the tactical search i.e., an absolute life-and-death answer may lie beyond the search horizon of the tactical or life-and-death search.

It is not surprising that heuristics are used, since from a complexity class analysis, determining the winner from an arbitrary position is P-space hard (Lichtenstein & Sipser, 1980) and deciding whether a player can force a win is exponential-time complete (Robson, 1983). These theoretical results indicate that no deterministic polynomial time algorithms is likely to exist for determining territory from an arbitrary position.

3. Game-Tree Search and AI Techniques in Competitive Go Programs

The competitive Go programs which are currently active in Computer Go tournaments include Explorer (*EX*) by Martin Muller (1995), Go Intellect (*GI*) by Ken Chen (Chen, 1989; 1990; 1992), Go4^{++} (Go4) by Michael Reiss, HandTalk (*HT*) by Chen Zhixing, and The Many Faces of Go (*MFG*) by David Fotland. We report the details of the programs with respect to game-tree search as discussed in Section 2 and also with respect to Go specific AI techniques: tactical search, life-and-death search, and influence functions.

3.1 Position Representation

All programs represent stones, strings and groups, typically using pattern-based heuristics to determine

6. A ply is a single move by either Black or White, as distinct from a game turn, consisting of a Black and White move.

connectivity between strings in order to determine the strings which belong to individual groups. *Armies* (or *groups*) are also represented in EX and GI. The heuristics used to determine armies include influence (GI), and zones of territory (EX).

Important attributes of the objects represented on the board (e.g., groups, armies) include their life-and-death status (also referred to as *safety* or *strength*), territory value, number of eyes, and influence. The values of attributes in some cases are determined by tactical search.

Some components of MFG’s representation are controlled by the evaluation function (e.g., groups, connections or links, eyes, territory, and influence). Go4's representation of the board is simply the representation formed by the evaluation function (e.g., groups, eyes, safety, territory).

3.2 Candidate Move Generation

Typically, candidate moves are heuristically generated by patterns or more generally, rule-based expert systems. Board points suggested as good moves are assigned a value with all suggested values being summed (linearly or by weighted sum) for each board point at the end of the move generation process. Typically the highest scoring board points are selected as moves for full-board evaluation.

The number of candidate moves which are evaluated at a full-board level varies between the programs: up to 12 in GI (Chen, 1997); up to 10 in MFG; and a minimum of 50 in Go4. The number of rules contained in a program varies: around 100 for EX; around 200 for MFG. GI contains around 20 move generators which are based on either pattern libraries, goal oriented search, or heuristic rules (containing perhaps dozens of rules).

Patterns usually contain both low-level and high-level information. The low-level information pertains to the position of the black and white stones, which points must be empty, and the points whose occupancy are not of interest. The high-level information pertains to such things as number of liberties, safety, eyes, and territory. Pattern matching consists of matching not only the stone configurations but also any high-level requirements on stones or strings within the pattern. Pattern matching a large number of patterns can become computationally expensive, and is further complicated due to the symmetry possible on a Go board. This problem has lead to the development of special algorithms to overcome the problems associated with pattern matching (e.g., hashing functions in MFG, string matching in EX).

Knowledge is incorporated into programs in different ways: Some programs work almost entirely from first principles, others match the current position against stored patterns. The number of patterns varies greatly between the programs: around 15 in Go4; around 1,200 in MFG; and around 3,000 in EX. Some programs also contain a database of opening move patterns (*joseki*) (e.g., MFG contains around 45,000 joseki patterns).

3.3 Goals

In most cases, a large amount of territory is more desirable than a small amount and similarly for influence although there sometimes exists a trade-off between territory and influence (particularly in the opening and middle game). However, although heuristic estimations of territory are possible, territory is not always the best indicator of the merit of a position. In the early part of the game, possession of a large amount of territory may indicate an over-concentrated position which may be detrimental with respect to securing territory later in the game. Maximising influence rather than territory at the beginning of the game generally results in acquiring more territory at the end of the game. Influence is an example of a sub-goal which may be useful in determining the merit of a position.

There seems to be no consensus in Computer Go about the relative priority of the various sub-goals associated with determining a position's merit. Typically life-and-death status (safety) of groups and territory are included as goals or sub-goals. In HT, the emphasis is on tactical fighting as in MFG which also concentrates on connectivity, eyes, and group strength. Go4 concentrates almost exclusively on connectivity; almost everything is eventually derived (directly or indirectly) from the connectivity probability map.

3.4 The Evaluation Process

Evaluating Go positions is a slow process (e.g., MFG evaluates less than 10,000 full-board positions for the entire game at less than 10 evaluations per second compared to 10,000-100,000 evaluations per second for chess programs). Limits are typically placed on the number of full board evaluations carried out due to tournament time constraints (e.g., MFG performs no more than 100 full-board evaluations in selecting its next move).

In Go4, each of the 50 candidate moves are evaluated by a 6 step process: 1. A connectivity probability map is generated. For each black and white stone on the board, the probability of connecting it to a friendly stone (real or hypothetical) at the 32 connectable points[7] is computed (generating a large amount of data). Determining connectivity involves tactical search; 2. Groups are determined from the connectivity map and

7. There are 32 possible points to which a stone can be connected using the 6 types of links defined in the Go literature.

tactical search; 3. Eyes are determined (using patterns) from the connectivity and group data; 4. A group's safety is determined based on the number of eyes it has; 5. The safety of each stone is radiated in proportion to its connectivity probability map and summed over all stones; 6. Black and White territory is estimated from the radiated values. The difference between Black and White territory is returned as the evaluation for a given move.

In MFG, evaluation is a multiple pass process and relies heavily on tactical search. All strings with less than 4 liberties and some strings with 4 liberties are examined by tactical search in order to determine dead strings. Tactical search is also used to identify connections and eyes. At this stage, strings are formed into groups. Group strength is determined based on life-and-death considerations (e.g., connections, eyes) and used to determine the amount of control exerted by white and black on each board point (in a range between -50 and +50). Territory is determined based on the sum of the values for each point and gives the final evaluation value. Quiescence search up to 6 ply can be performed and sometimes uses a special set of patterns to suggest local moves that make the position quiet.

In GI, evaluation is used during global search. If the value of one of the candidate moves is significantly better than the values of any of the others, it is played as the next move. However, if several of the candidate moves are approximately equivalent, global search, facilitated by evaluation, is used to determine which one to select. During evaluation, the safety of armies is used as a basis for assigning a value to each board point between -64 and +64 indicating the degree of Black and White control. The values are summed for all points to return the evaluation value. The global search examines no more than about 6 to 7 moves and searches no deeper than about 6 ply.

3.5 Tactical Search

Tactical search is selective goal-oriented search and is used for a variety of purposes including determining whether strings are dead or alive (Go4, MFG, EX, GI), whether connections are safe or can be cut (Go4, MFG), whether eyes can be formed (MFG), candidate move generation (GI), and determining the life-and-death of groups (EX). Just as at the full-board level, tactical search also requires move generation and evaluation. Tactical evaluation differs from full-board evaluation in that it is with respect to the goals of the search (e.g., the number of liberties of a string). Due to time constraints, tactical search is typically limited in terms of the number of nodes, branching factor, and ply depth. Hence, although problems such as life-and-death are usually thought of as being tactical, stones may be strategically dead, even if they cannot be tactically captured. Thus, as in all areas of evaluation in Go, the tactical search is just one more heuristic device.

MFG provides a good illustration of the operation of tactical search (by the tactician): Every string with 3 or less liberties and many strings with 4 liberties are examined by the tactician. Each string is examined twice, once with White moving first and once with Black moving first. The tactician determines whether a string is captured (i.e., can not live even if it moves first), threatened (i.e., it lives if it moves first and dies if it moves second), or stable (i.e., lives regardless of who moves next). The tactician relies on simple heuristics (e.g., the number of liberties and connectivity).

The tactician has two separate move generators; one to generate attacking moves and one to generate defensive moves. The moves suggested by the move generators are sorted according to criteria which include second order liberties (liberties of liberties), cut points, and simple eye shapes. Once sorted, an alpha-beta depth-first search is employed with the performance of the search depending on the quality of the move sorting. Move generation and sorting accounts for most of the time spent in tactical search.

Tactical searches are goal-directed and are limited to a maximum number of nodes. For string captures this limit is around 100, however, when only one move is suggested, it is not counted towards the node limit. In this way, the winner of a *ladder* can be determined without difficulty. The number of nodes for a search are allocated to the branches according to the value given to the moves by the first ply move generator and thus different branches may end at different depths. The branching factor at each successive ply is progressively constrained by the tactician; the branching factor falls from 5 at the first ply to 1 or 2 by the fifth ply.

In the course of a game, MFG performs around 100,000-150,000 tactical searches, visiting around 1.5-2 million nodes at around 2,000-2,500 nodes per second. On average, each tactical search visits around 10-20 nodes although many searches visit less than 5 nodes with a few searches being terminated due to the node limit.

3.6 Life-and-Death Search

Not all programs explicitly perform life-and-death analysis; many use a generic tactical search for that purpose. MFG perform life-and-death analysis in a way that is similar to its tactical search procedure except that it involves life-and-death analysis on groups rather than tactical analysis on strings.

A static life-and-death evaluator determines the life-and-death status of every group in a multiple pass process without using lookahead by progressively creating complex structures from simple structures. The tactician is used by the static life-and-death evaluator and is called at least twice for each appropriate string in a group.

A life-and-death search is goal directed (e.g., to save or kill a group). If the goal of the search is not achieved at a particular node, appropriate moves are generated by the life-and-death search engine's own move generators and the search is continued. The static life-and-death evaluator is called at each node during a life-and-death search in order to determine whether the goal has been achieved. The life-and-death search engine uses depth-first alpha-beta search and the depth to which a particular branch is searched is heuristically determined. The size of the search tree is constrained and is typically around 20 nodes in size and up to 7 plies deep. Life-and-death search is slow and full trees are cached to reduce the overhead associated with researching. The slowness of life-and-death search also means that it is not used during full board evaluation.

3.7 Influence Functions

Influence is a Go concept which provides an indication of the potential control exerted by the stones of each player on empty points. By ensuring that the placement of stones is not over-concentrated at the beginning, a player maximises the chance of acquiring territory later in the game.

Influence is computationally modeled by influence functions (Zobrist, 1969; Ryder, 1971; Chen, 1989). Typically, stones radiate influence (black and white stones radiating opposite values) to surrounding board points with the radiated influence values from all the stones being summed at each board point (Black and White influence is kept separate in MFG). The influence radiated from stones decays as a function of distance: $1/2^{distance}$ in GI; 1/distance in MFG. Programs which rely too heavily on influence do not play well (EX and Go4 no longer use influence although Go4's radiation function is similar to an influence function and EX uses other more measures such as distance to stones of each colour, or connectable points).

The heuristic use of influence includes determining connectivity and armies (GI) and determining territory (MFG). In MFG, the initial value radiated from a group depends on its strength i.e., strong groups radiate more influence than groups which are weak or almost dead. This also means that dead stones radiate negative influence i.e., to the benefit of their opponent. In both MFG and GI, influence does not radiate through stones; in MFG it also does not radiate through enemy links.

4. Discussion

Current Go programs all use considerable knowledge. Based as they are on their designer's own heuristics for successful play, each program can be viewed as an empirical experiment in a (possibly implicit) theory of Go. The instantiation of a theory of Go in a program critically depends on the data structures chosen, as they determine the ease of encoding different types of knowledge, and the computational expense of manipulating such knowledge. Programs evolve as the programmer gains skill both in Go and Computer-Go (e.g., in the past 15 years, as Fotland progressed from 15 kyu to 2 dan, MFG has gained in strength and grown to its current size of ~40k lines of code). A program is only as strong as its weakest component, and the ease of adding new knowledge to a program is an important part of improving its performance.

As yet, no consensus exists in the Computer Go field as to how to structure a Go program, or the priorities that different Go knowledge should be assigned (e.g., Go4 concentrates on connectivity and MFG concentrates on life-and-death). It should be noted that there is also no consensus as to how humans play Go, which is a current topic of Cognitive Science research (see Saito & Yoshikawa, 1997, for a review). Any advancements in this area may feed directly into theories of Go, and possibly lead to improved algorithms in Computer Go programs.

In this paper, we have collated detailed information about current successful Go programs. In doing so, we have analysed Go within the framework of game-tree search, and exposed the problems associated with viewing Computer Go programming from that paradigm. This difficulty is common knowledge within the Computer Go community, but is little known or accepted in the wider AI community. In Go, full-board evaluation requires the determination of the life-and-death status of groups by either life-and-death or tactical search, resulting in evaluation being computationally expensive. The lack of a fast, effective evaluation function is the primary difficulty encountered in Computer Go and is compounded by a large branching factor, and by the real-time requirements of users and computer tournaments (typically 24 seconds per move compared to 180 for chess). Thus, the full-width game-tree search approach common in Computer Chess is unsuccessful in Computer Go.

In addition to outlining the issues inherent in the Go domain our paper surveys how current programs deal with these issues, thereby providing a springboard for prospective Go programmers. Note that the Computer Go field is a commercial one, and the programs themselves (rather than academic papers) are its main products. Other than the references noted, the details reported here have been acquired through personal communication with the programmers themselves (who have generously shared their knowledge), and augmented by information from the moderated computer-go mailing list `<computer-go@anu.edu.au>` and ftp site `<ftp://igs.nuri.net/Go/comp/>`.

The challenge of Computer Go lies in extending the current theories of Go or developing new ones, in particular, with respect to evaluation, designing appropriate data structures and algorithms for the real-time constraints, and solving the knowledge bottleneck. None of the current competitive programs use learning techniques, although there have been several such attempts (e.g., Pell, 1991; Schraudolph, Dayan & Sejnowski, 1994; Donnelly, Corr & Crookes, 1994). It is beyond the scope of this paper to review such programs, but we conjecture that they have not yet succeeded because their programmer's implicit theories of Go lack the requisite understanding of the complexities of the Go domain. It is an open question how to add learning to the current programs (or their implicit theories of Go).

Acknowledgments

We would like to thank Ken Chen, Chen Zhixing, David Fotland, Martin Müller, and Mick Reiss for communicating details of their programs to us and for their helpful feedback on this paper.

References

Allis, V. Searching for solutions in games and artificial intelligence. PhD thesis, University of Limburg, Maastricht, 1994.

Burmeister, J. & Wiles, J. The challenge of Go as a domain for AI research: a comparison between Go and chess. In *Proceedings of the Third Australian and New Zealand Conference on Intelligent Information Systems*, pages 181-186, Perth, November 1995. IEEE Western Australia Section.

Chen, K. Group identification in computer Go. In D. N. L. Levy and D. F. Beal, (eds), *Heuristic Programming in Artificial Intelligence: the First Computer Olympiad*, pages 195-210. Ellis Horwood, Chichester, 1989.

Chen, K. The move decision process of Go Intellect. In David Erbach, editor, *Computer Go*, 14: 9-17, 1990.

Chen, K Attack and defence. In H. J. Van den Herik and L. V. Allis, (ed)s, *Heuristic Programming in Artificial Intelligence 3 - The Third Computer Olympiad*, pages 146-156. Ellis Horwood, Chichester, l992.

Donnelly, P., Corr, P., and Crookes, D. Evolving Go playing strategy in neural networks, 1994. Available on the Internet at `ftp://igs.nuri.net/Go/comp/egpsnn.ps.Z`.

Fotland, D. Knowledge representation in The Many Faces of Go, 1993. Available on the Internet at `ftp://igs.nuri.net/Go/comp/mfg.Z`.

Lichtenstein, D. & Sipser, M. Go is polynomial-space hard. *Journal of the ACM*, 27(2): 393-401, 1980.

Müller, M. Computer Go as a sum of local games: an application of combinatorial game theory. PhD thesis, Swiss Federal Institute of Technology Zürich, 1995.

Pell, B. Exploratory learning in the game of Go. In D. N. L. Levy and D. F. Beal, (eds), *Heuristic Programming in Artificial Intelligence 2 - The Second Computer Olympiad*, volume 2. Ellis Horwood,1991

Robson, J. The complexity of Go. In R. E. A. Mason, (ed), *Proceedings of the IFIP 9th World Computer Congress*, pages 413-417, North Holland, 1983. IFIP, Elsevier Science Publishers.

Ryder, J. Heuristic analysis of large trees as generated in the game of Go. PhD thesis, Department of Computer Science, Standford University, 1971.

Saito, Y. & Yoshikawa, A. Go as a testbed for Cognitive Science studies. In IJCAI Workshop Proceedings *Using Games as an Experimental Testbed for AI Research*, 1997.

Schraudolph, N., Dayan, P. and Sejnowski, T. Temporal difference learning of position evaluation in the game of Go. In J. D. Cowan, G. Tesauro and J. Alspector, (eds), *Advances in Neural Information Processing 6*, pages 817-824. Morgan Kaufmann, San Francisco, 1994.

Zobrist, A. A model of visual organization for the game Go. In Proceedings of the *Spring Joint Computer Conference*, 34: 103-112, 1969.

Zobrist, A. Feature extractions and representation for pattern recognition and the game of Go. PhD thesis, Graduate School of the University of Wisconsin, 1970.