Positions in the Game of Go as Complex Systems

Thomas Wolf
Brock University
St. Catharines, Ontario, Canada
WWW: http://www.brocku.ca/mathematics/people/wolf/
Email: twolf@brocku.ca

***Abstract*—The paper gathers evidence showing different dimensions of the game of go: the continuous and discrete nature of the game and different types of relations between state variables happening on ultra local, local, regional, and global scales. Based on these observations a new continuous local model for describing a board position is introduced. This includes the identification of the basic variables describing a board position and the formulation and solution of a dynamical system for their computation. To be usable as a static evaluation function for a game playing program at least group-wide (regional) aspects will have to be incorporated.**

I. Introduction

When someone invests much time into a project, into using a computer with exclusively one operating system, into programming using only one programming language, or into playing one specific game then the person is naturally biased, and it is almost impossible to convince him or her that another operating system is 'better' or that a different game is 'harder' or 'more interesting' (whatever that is supposed to mean). In this contribution the author tries to do something exactly like that: to justify the claim that the game of go is different from other board games, and that it shows features of complex systems.

We will show that go can be described along different and fairly independent dimensions; that go is a rich challenge that can neither be 'solved' by a mathematical formula, nor be played nearly perfectly by a single elegant computational algorithm. The purpose of this paper is not to establish superiority (in any sense) of one game over another, but to analyse why computers do not play go as well as, for example, Chess. Statements will not be proven but justified by examples.

In section II it is argued that go has inherently continuous and discrete characteristics which suggest an intertwined continuous-discrete problem solving technique like the one introduced later in section VI.

In section III a different dimension of go is discussed. It is shown that there are (at least) four spatial types of relations between points, stones and chains on the board.

The richness of go also becomes apparent from the many different solution strategies which have been tried so far. Section V describes a few. A new approach to computer go is outlined in section VI.

More go-specific content is to be found in the appendix. Specialist go terminology set in *italic* in the text is explained there.

II. Continuity versus Discreteness

Although the game of go is strictly speaking fully discrete it nevertheless has partially continuous aspects which we will refer to below simply as 'continuous' or 'continuity' although they are of course only approximately so.

A. Continuous Aspects of Go

With two players alternating in making a move the game definitely has discrete aspects. But go also shows continuous features, especially when the influence of *chains* matters, and when the number of *liberties* is not crucial.

The first support for the claim of the partially continuous nature of go comes from the fact that the number of legal board positions on a 19×19 board which according to [1], [2] is about $0.011957528698 \times 3^{361} \approx 2.081681994 \times 10^{170}$, and by far exceeds the number of possible different scores ($722 = 2 \times 19^2$ for integer *komi* as well as for half-integer komi). Therefore many different positions must have the same score even if one is 50 or 100 moves into the game. Although this statement theoretically still allows for the case that just one move gives the optimal result and all other remaining legal moves give a lower final score, this is extremely unlikely in calm positions earlier in the game. (Of course if one side plays a large threat then there is often just one possible reply.)

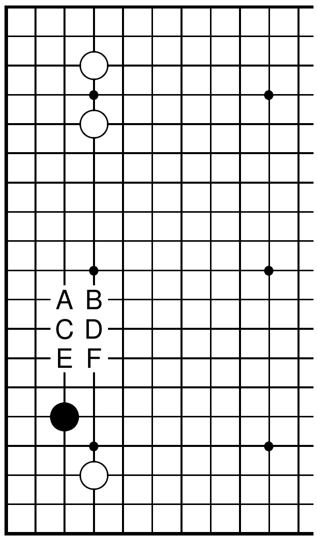

Diagram 1. ◯ to move with comparable options A..F

An example supporting the claim of partial continuity is shown in Diagram 1 which is taken from [3]. It shows points A-F as possible places for ◯ to move next. Although they will lead to different games, their game-theoretic values are very close if not equal.

The second example shows how go is a game of trade and not simply one of finding a "killer"/magical move which inexplicably does the job. The possibility to trade approximately equal values (of possibly different qualities, like area against safety) demonstrates the continuous side of go too. Naturally, the possibility for exchanges of continuously-valued entities is higher at earlier stages of the game, when the board is still mostly empty, as in the above example. Diagram 2 shows an example from the early middle game: the position after move 49 in a professional game between Yun Seong-hyeon (●) and An Yeong-kil (○) played on 3rd Jan 2001. The task for ○ is to strengthen his weak group marked ⊗.

The need for ○ to act, and offer or enforce a trade, comes from the fact that a group can survive permanently only if it surrounds at least two separated empty points on the board, i.e. there is a threshold for the minimum size and extension of a group of stones to be stable. But in Diagram 2 the stones ⊗ currently do not have that needed control on empty points. Neighboring assets that ○ still has at his disposal for trading are some potential on the lower edge and some minor potential on the ● dominated right edge. An idea for an exchange could be to offer ● total control of the right edge, and increased influence on the bottom edge, in exchange for more influence and strength of the ⊗ stones in the center. In the appendix we discuss in more detail how a ○ attack on the stone at A, and a sacrifice on B can achieve that. Although two moves (A and B) are shown in Diagram 2, often many variations are possible, but sometimes only one move is able to initiate a (pseudo steady) shift of potential.

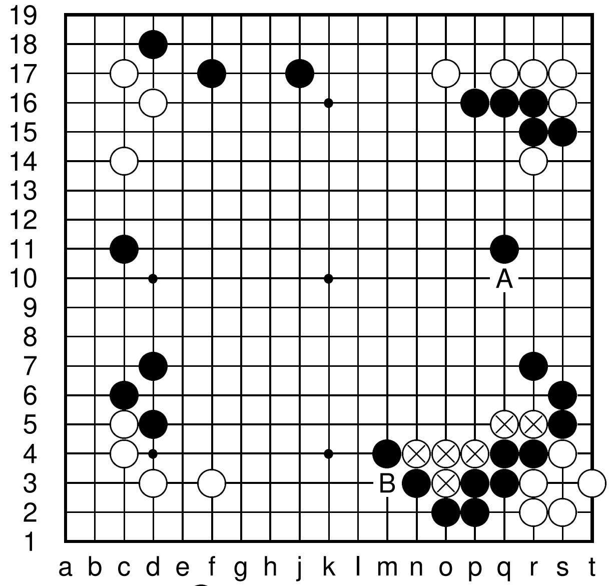

Diagram 2. ○ to move.

B. Discrete Aspects of Go

That board games live in a discrete domain of trees of alternating moves is obvious. A different type of a stronger discreteness / branching is shown in Diagram 3. This is the position after move 48 (⊗), in a game between two high level amateur players. For comments on this position please see the appendix.

● moving next in Diagram 3 can attack at A or B, and either get the corner, or give the corner to ○ but weaken the ◬ stones with strategic consequences. Although the two alternative moves on A and B are spatially close, the resulting trade has a lasting effect on a more extended area and is non-reversible. More details are given in the appendix.

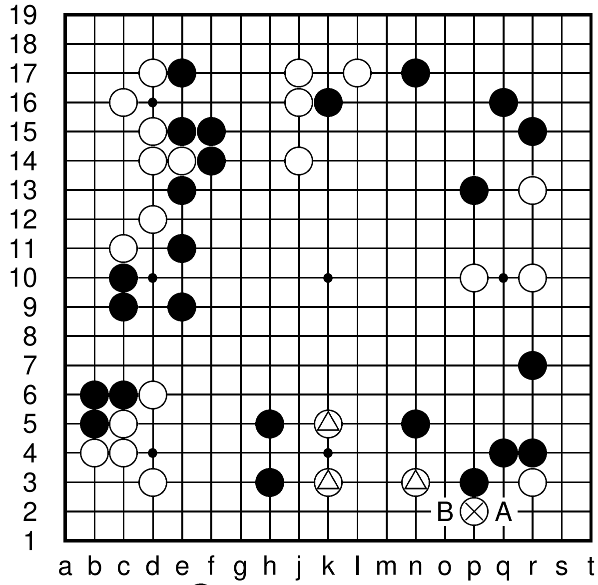

Diagram 3. ● to move.

C. Unbounded Amplification

Continuous processes with discrete outcomes, like all real life games (hockey, soccer, rolling dice, roulette,..) or real life decision processes, are of interest as they show on occasion an arbitrarily large amplification of minor differences in continuous input data to result in a discrete outcome. These are often dramatic events, for example, when a football just touches a goalkeeper, then hits the post, and scores or does not score. Such branchings are not restricted to games. The outcome of these processes is a discrete results which often has much impact, and can not be reversed. Therefore the decision process has, if necessary, to be flexible enough to provide large resources for trying to predict the outcome.

In professional go it is often the case that a game is won only by a small margin, sometimes by the smallest possible margin of 1/2 point (half integer outcomes resulting from half integer *komi*). Thus, when a decision such as that in Diagram 3 has to be made, it is crucial that it be a well-informed choice. In this case the value of the lower right corner can be determined exactly, but the point-value of the weak, but extended, ○ group on the lower edge is tied to the rest of the board and can only be estimated. For computers correct estimation still implies large simulations.

In a game like Chess one would do a quiescence search, i.e. one would search locally deeper in the search tree, as long as the situation is not quiet – e.g. as long as stones can still be captured in the next move, etc ..., whereas in go this is not feasible. In go, searches are either always done to the end of the game (Monte Carlo search), or are done mostly too flat (mini-max search) with many local threats being able to push the real issue of a fight out of the horizon of visibility for the search (even if one might be able to design counter measures for clear cut cases).

III. Relations on different Scales

When designing a solution process, and determining the data structures that are to be used, one wants to minimize the number of variables and one wants to minimize the number of dependencies between these variables to increase effective speed, i.e. to compute deeper and achieve a higher playing strength.

To minimize the number of dependencies one needs algorithms to be as local as possible, for example to investigate only relevant followup moves which are in some sense 'local' to the previous move.

In the next short sub-sections we give evidence how relations between variables that describe the situation in a go game can be of different nature and involve information that is available locally or only globally. We start with the most simple and preferred type of relations which we call local and extend to ultra local, regional and global as need arises.

A. Local Relations

The most obvious interactions in go are purely local - the interactions between neighbouring *chains*, and their shared liberties.

The capture rule in go states: *A chain of stones (of one colour) is captured (and by that all its stones are taken off the board) when the opponent occupies all adjacent points* (see appendix). Even if not captured, chains are considered dead at the end of the game if the opponent can show that a capture can be enforced. Thus, all that matters for this essential rule of go is the immediate neighbourhood of chains.

For a local model based on this observation the elementary objects (called *units* in the remainder of this paper) would consist of all chains and all *points* (empty intersections on the board). All that would be stored for a unit would be

- its name (for a point its coordinates and for a chain its index),
- a list of neighbouring units (points and chain indices),
- some data describing the state of this unit, for example, a strength value for a chain (e.g. the probability of survival) and for a point the influence value (e.g. the probability of becoming owned by ○ or ● at the end of the game), but
- not the shape of the chain.

Although chains can become as large as the board, we still call this model *local* because it is only the immediate neighbourhood on which the state of each unit depends. To give an example, *semeai* fights (races to capture), for example between ◎ and ◉ on the right edge of Diagram 4, could be solved by such models, as could any other local fights which end when one chain is captured, for example, when chains get so big that their survival is essential.

Another type of tactical question that is in the range of local models is how far one side can invade enemy territory with the next move, so questions that can be answered based on influence.

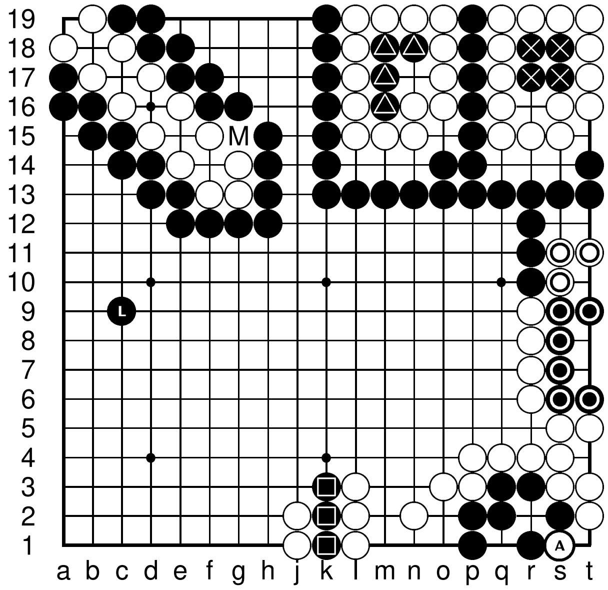

Diagram 4. Several local positions

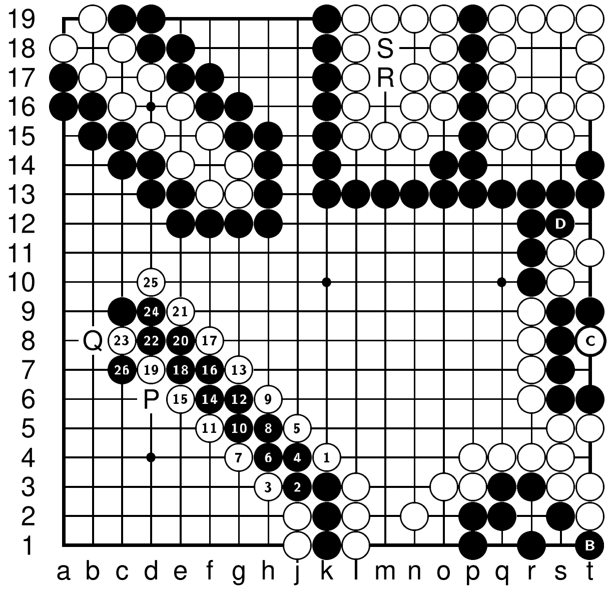

Diagram 5. The same positions at a later stage

B. Ultra local Relations

If a chain is captured then on the arising empty space many points (i.e. units) appear and start to interact – suddenly it matters what the shape of the captured chain was. For example, when the throw-in chains ▲ and ⊗ at the top of Diagram 4 are captured (as at the top of Diagram 5) then the shape of the ⊗ chain implies that the capturing chain will be dead whereas the chain capturing ▲ will be alive. (○ can always play at R or S and live.) When chains can not be represented by a few numbers as in the local model but if their shape matters then we call this *ultra-locality*.

The question arises: Why should one consider locality if, to be precise in computations, eventually one would have to consider ultra-local data and relations anyway? The answer is: The neighbourhood of a chain does not hold enough information and the shape/interior of a chain becomes interesting only if the chain gets captured and if in addition the chain had more than 3 stones and also not too many stones because the capturing chains would otherwise live independently of the shape of the captured chain.

Thus, working by default in a local model, and going ultra-local only when necessary (e.g. when evaluating eyes), reduces complexity compared with working ultra-locally throughout.

C. Group (regional) Relations

Apart from ultra-locality, could a sufficiently sophisticated local model be rich enough to derive all go knowledge and play almost perfectly, given enough computing power?

How about positions where a move has a long-distance effect? For example, the sequence of moves starting with ① in Diagram 5 could finally catch the chain ◙ in Diagram 4 if the stone 🅛 in Diagram 4 were not present. Although it would be difficult for a local model to realize the crucial role of 🅛 this is not a good counter example for locality in go. For example, in physics waves can propagate long distances and fields can be far ranging and still be described purely by local relations (differential equations instead of integral equations).

An example for a concept in go that can *not* be described locally in the above sense is the concept of *independent life* which is defined recursively: *A chain is alive if it participates in at least two living eyes and an eye is alive if it is surrounded only by living chains.*

Thus, life is not necessarily the property of a single chain but of a whole set of chains which are all alive, or all dead. An (artificial) example is shown in the upper left corner of Diagram 4. The life of ○ depends solely on who plays next at M. If this is ● as in Diagram 5 then all ○ are dead.

It is important to realize, that the property of life of a whole group of chains has to be verified at once, and can not be decided in an iterative local process based on a local dynamical model, starting with some initial values for the state variables and then trying to recognize exact life through an iteration process. The crucial point is the recursive nature of the definition of life.

The range of this non-locality would be the range of all neighbouring chains whose life status is yet undecided, and which are dependent on each other somehow. The typical size of such undecided areas would typically be much less than the whole board, so one could call relations between the units of such a group of chains regional or group-wide, but not global.

D. Global Relations

Are there further causal relations between parts of the board which are not local, or group-wide? The answer is yes. The stronger a player is, the more wide-ranging are the effects that (s)he consider. But, even for weaker players some positions are of a truly global nature, like *ko* situations. When a ko threat is played, areas which are settled may become unsettled if a ko threat is not answered, and thus one side plays two times in a row in that area. This is illustrated in the following example.

For ● to live in the lower right corner of Diagram 4 at first Ⓐ has to be captured as in Diagram 5 by 🅑 and then ● has to put a stone onto the position of the former Ⓐ. ○ instead wants to re-capture 🅑 but is not allowed to do this instantly due to the *ko-rule* (to avoid loops, see appendix). Before continuing with the example it should be noted that the semeai on the right side is settled. Even if ○ would move first, he would still be behind one move and would get captured. But with an on-going ko White plays Ⓒ in Diagram 5 and challenges ● either

- to answer with 🅓 and to stay ahead in the semeai but allow ○ to capture 🅑, or
- to link 🅑 to the other black stones in the corner and live there but to loose the semeai race because ○ will move the 2nd time in a row in the semeai and thus be ahead.

IV. Summary on Continuity and Locality

The consequence of the above considerations is that there can not be go programs that are elegant/compact and efficient at the same time. To be efficient the programs have to take advantage of relations being local when that is the case but that requires much go specific domain knowledge. For example, pure Monte Carlo type programs are compact but not as efficient as programs could be because they know almost nothing about local or group-wide relations. The procedures described in section VI are efficient and compact, but they have to be extended by life-and-death knowledge, by more knowledge about local stability dependencies, and by global search. Thus to become a strong full-game playing program, such a program would definitely no longer remain compact.

The essence is that go positions are complex hierarchical systems without a mathematical, or compact algorithmic, solution.

The example in Diagram 2 supports this conclusion. When thinking on a large scale, the possibility of planning to exchange areas, allows one to cut down the number of moves that need to be examined in detail. This simplifies the problem for humans. But, in go, what is relatively easy for humans (planning, being creative to move the solution process partially into a more abstract domain) is hard for computers.

The situation is different in other games. For example, the game of Chess is essentially purely global because queens, rooks and bishops can move over the whole board in one move and pawns can be promoted, and turn into any piece. Thus, the different parts of the Chess board are much more causally linked. Imagine how difficult a Chess game on a 19x19 board would be, with about 30 pieces being able to move over the whole large board in just one move! Also, Chess is completely discrete in its nature. To be clear, the issue of complexity should not be mistaken for difficulty. Both games – Chess and go – are too difficult to be solved by computers and are therefore in some sense 'equally difficult' but go shows properties of complex systems which most other well-known games do not have.

V. Different solution paradigms

In the following subsections we look at the pros and cons of different approaches to computer go.

A. Offline versus Online Computations

One way to characterize each method is its ratio of computer time spent before the game to that spent during the game – offline versus online. Computational tasks are

1) suitable for online computations, i.e. computations during the game, if the computational tasks
   - are relatively quick to perform, and
   - hardly ever occur in that form twice, i.e. the problems are too numerous to store, and are
2) suitable for offline computations if they
   - are non-computable, or very hard to compute, and
   - occur relatively frequently, i.e. the problems are not too numerous to store.

The essence of this statement is that any method which concentrates on only one side of this duality is seriously handicapped. For example, it is impossible to pre-evaluate all positions which can come up after 50 moves and it is impossible to compose all pattern of size, say, 16×16 which may very well be the extension in one direction of a weak group that fights for life. It also is not possible to reduce the essence of a position with, say, 50 stones to manageable amounts of ideas, patterns, and concepts ... that can be pre-computed and stored. A position can change its nature completely by displacing a single stone by one point, with the consequence that a ladder now does not work,... . As a consequence online computation can only partly be compensated by offline learning. One could argue that professional human players can not do large computations online (i.e. not do a tree-search with 1000s of nodes per second), i.e. that the offline/online ratio is very large for them. Although this is true, current AI is very far away from implementing a human professional go player. With the rise of Monte Carlo programs the trend was rather in the other direction of a low ratio offline/online.

B. Knowledge based programs

As remarked above in section III-A, relations between chains are often purely local. Naturally, it would be an inappropriate waste of search effort to solve a local problem (semeai, local invasion) by a wide search and thus exponentially less effective than through a narrow search. But pruning search requires knowledge.

Knowledge based programs were the more successful ones in the first three decades of computer go. Their advantage is to solve specialized problems (opening moves, local life and death fights, creating patterns of effective shapes, ...) at a strong amateur Dan (master) level. But they have disadvantages:

- For knowledge to be of increasing quality not only does the effort of acquiring and maintaining the knowledge increase, the domain of applicability also shrinks. In other words, situations on the board need to be specific to have efficient and exact algorithms and procedures describing them accurately. Thus the cost-benefit balance gets worse the higher the quality of the knowledge is.
- Information coming from different knowledge bases, solving different sub-problems, and describing different issues, has to be merged to reach a single decision: "what is the best next move". The quality of merged knowledge is typically at a much lower level than the specialized high level knowledge itself. Thus, knowledge-driven programs are good if the whole problem reduces to the solution of a tactical sub-problem for which a specialized module exists, but not if dual purposes have to be pursued simultaneously at a high level.
- The more involved the software for the knowledge base gets (several data bases, either partially hand crafted (pattern), or automatically generated (opening, intermediate results of local tree-searches), pre-generated eye-database,..), 100's of tactical modules written over decades by one person or a team - the harder it gets for someone new to penetrate such a package, and to continue developing it.
- Knowledge-based programs do not scale – i.e. they can not easily, or even at all, convert computing power into strength.

C. Learning through statistics from professional games

By collecting statistics on patterns occurring in professional games, and then nesting these patterns, and organizing them in a database, it is possible to achieve relatively high (in the 50% range) prediction rates for moves in professional games (see [4] and other papers by Stern et al). The problem with such approaches is that the programs have no understanding of the situation so they occasionally make moves which, in a crucial situation, are totally wrong and thus ruin a game.

D. Monte-Carlo programs

Based on an early concept for Monte-Carlo (MC) simulations applied to go [5], this approach started to dominate computer go in recent years when combined with the UCT algorithm [6], [7], which is a tree search method based on Upper Confidence Bounds (UCB). This type of approach (see e.g. [8]) produced new programs – [9], [10], [11] – that are stronger than the previous best programs, by an equivalent of 5-6 handicap stones. For the first time it was possible to beat high dan professional players (at least in the first game) when starting with 7 handicap stones ([12]).

This strength increase was made possible by abandoning initial knowledge completely, avoiding the derivation of special domain knowledge, and thus saving the effort of merging knowledge from different sources. Instead they performed a progressively selective tree-search, based on success rates of moves learned during the search. The UCT formula provides a compromise between exploring new moves and replaying successful moves and exploring their consequences. Although starting out only with random move sequences, and performing thousands to millions of simulation games, just to find the next move, this method can convert, at least to some extent, computing power into playing strength, and thus is able, for example, to utilize large parallel computer clusters. The following are the principal problems of this approach:

- If professional games are played to the end then the result is typically in the 5-point range. With a game taking on average 250 moves this means that one side was on average only $\frac{1}{50}$ of a point per move better than the other side. As a go player it is hard to imagine $\frac{1}{50}$ of a point! It would be very expensive to derive such tiny relative advantages of one move against another move purely from a statistical approach. Instead, using knowledge and logic it is often possible to make a statement about the relative value of moves without doing any computations by just recognizing minor differences between otherwise equivalent moves. One could argue that one could program that knowledge and add such routines to MC tree search but then one is back to all the problems of knowledge based programs, especially losing scalability gradually, as more knowledge is added.
- The larger the board is, and the earlier it is in the game, the longer are the simulation games performed in the tree-search phase, and the less accurate are the results.
- Even if MC based programs should improve considerably in coming decades, it will still be highly unsatisfactory if, just to beat young kids, we require large computer clusters, each needing their own power station for the energy to run and cool them.

VI. Experiments with a local Model

Following the outline of section III-A a computer program was written that takes as state variables the Black/White influence value at each (empty) point and the probability for survival of each chain. All values are represented by real floating point numbers in the interval $0 \ldots 1$. A dynamical system of relations is formulated by expressing each variable in terms of these variables from neighbouring units. Then values are initialized and the system is solved numerically.

In using the probability values b, w for one point to be occupied by ●, ○ and the probability values $\bar{b}, \bar{w}$ that at the end of the game at least one neighbouring point is occupied by ● , ○ we compute b, w from the simple system $b + w = 1, \quad b\bar{w} = w\bar{b}$ (after expressing $\bar{b}, \bar{w}$ in terms of b, w of neighbouring units). The second formula is of course only a simple ansatz but at least it is correct for extremal values 0 and 1 of $\bar{b}$ and $\bar{w}$.

For chains we define the computed value of strength as 1 minus the probability to be captured, i.e. all neighbouring points to be occupied by the opponent using the values associated with the neighbouring units. If one drops one liberty of the chain in this computation (the liberty that is least accessible by the opponent and thus to implement the fact that a capturing move is always legal (apart from ko)) then the resulting algorithm is able to recognize life based on two 1-point eyes. But already if 2-point eyes are involved the algorithm will not be able to recognize unconditional life.

An incremental version of this algorithm has been implemented such that initial values of 0.5 for points and 1.0 for chains are given only once at the very beginning when the first position is evaluated (in a game the empty board or the board just with handicap stones).

To check the correctness of numerical computations the resulting polynomial system for the unknowns (black/white influences at points and strength values of chains) was solved analytically with all its complex solutions for small positions. In general, there was only one real solution with values in the interval $0 \ldots 1$. The only observed case where the solution of the dynamical system did depend on the initial numerical values was the case when two chains of opposite color were attached to each other and both had only one liberty. This situation is obviously very unstable, whoever moves next captures the other's chain. For such 'hot' situations a static analysis is of only little value.

The positive aspects of this algorithm are its simplicity, robustness, speed, global nature, and the lack of any artificial (and thus strictly speaking wrong) parameters. The algorithm's weakness in not recognizing static life or death is shared by all local iterative algorithms, as summarized in section IV and should therefore not be held against it. A more complete description of results is beyond the scope of this paper, and will be published elsewhere.

We can sum the probability-of-ownership for all points, and then, for each chain, calculate the survival probability, weighted by size. These can be added to get a simple estimated score. By performing all legal moves, and selecting the one with the highest score, the module can even play games. Because the program in its pure form does not know about life and death, it has hardly any chance in normal games. A more instructive test is to perform a ranking of all legal moves in the 10.4 million positions of the 50,000 professional games in the GoGoD collection [13] and to record where in this ranking the next move in the games appears. The large number of games allows one to calculate statistics for each move number. Figure 1 shows such a statistic for move number 100 (a typically difficult phase for the influence function) in all games. A point on the graph with coordinates x (horizontally) and y (vertically) means that in $y\%$ of the games in the position after move 100 the next move in the game is ranked in the range $x \ldots (x + 1)\%$ of all legal moves in that position.

Practically all related diagrams in other computer go publications show only the 'optimistic' right end of the diagram, not the 'pessimistic' left end. However, the characteristics of playing strength is not only to have a high rising graph near the 99% mark, but more importantly to have virtually no games where the professional move scores low. This shows that the local module that we studied still has a long way to go to become a strong static evaluation function.

VII. Summary

In this paper reasons are given why board positions in the game of go show properties of complex systems. A new model,

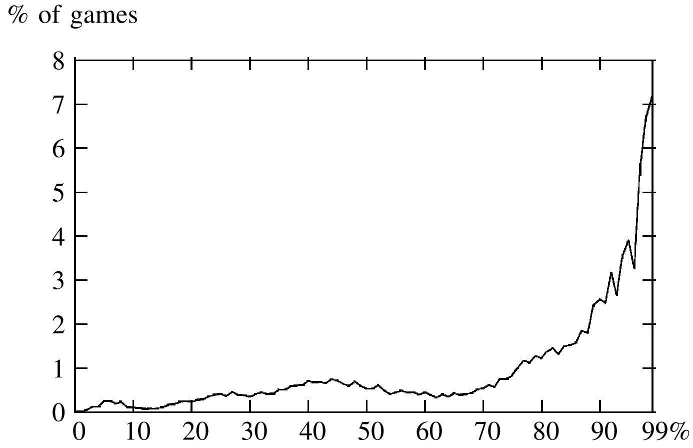

Fig. 1. A statistic of the ranking of the next professional move after move 100 in 50,000 professional games

and an algorithm, for computing a static influence function is described and its strengths and weaknesses are discussed.

Appendix on Go Terminology

To make this paper readable to non go players a few go specific terms are explained in this appendix. Some formulations are borrowed from Sensei's Library (`http://senseis.xmp.net/`) where more details including an introduction to go can be found.

A ***go board*** consists of a square grid of 19×19 lines (exceptionally $9 \times 9, 13 \times 13$), The two players have an unlimited number of, respectively, white and black stones and alternate in putting one of their stones onto an unoccupied intersection of two lines. An intersection of lines on the go board will be called ***point*** if it is empty and ***stone*** if it is occupied. Stones of one color attached to each other will be called a ***chain***. A point (that is empty by the above convention, and) that is a neighbour to a chain, is called a ***liberty*** of that chain. The ***capture rule*** of go requires that each chain has at least one liberty and if the last liberty becomes occupied by an opponent's stone, then the chain is captured and taken off the board. The captured stones are prisoners and count one point each after their automatic capture at the end of the game.

The aim of the game is to surround territory and to capture stones (1 captured stone is worth as much as one point of territory). The advantage to ● of moving first is usually compensated by ○ getting a number of ***komi*** (compensation points) which are in the range 5-7 and can be integer or half-integer valued (to avoid a draw – called ***jigo*** in Go).

We introduce the term ***unit*** to stand for either a chain or a point (i.e. an empty intersection of lines on the board).

The word field will be used for a numerical value being attached to each unit like a strength field attaching a numerical strength to each point and each chain on the board. To avoid confusion with points we will call this ***numerical field*** and otherwise avoid the word field.

An ***eye*** denotes points and stones surrounded by chains of one color.

In go the rule which forbids infinite loops of repeating sequences is called the ***ko rule*** and the situation to which it applies is called ***ko***.

A player who has ***sente*** can decide where to play next. Not to have sente is often to be constrained into a direct answer to your opponent's previous play. The converse of this state of affairs is called ***gote***.

Seki means mutual, or shared, life, and involves at least one chain of each colour. In its simple form, it is a sort of standoff where two live groups share liberties which neither of them can fill without being captured.

Joseki are generally agreed-upon sequences of play, mainly in empty corners, resulting in what is considered a fair outcome for both players.

Semeai describes a race of two neighbouring chains of opposite color trying to capture each other.

Moyo is a larger area, potentially owned by one side.

There are many playing levels in go. They are grouped into ***kyu*** grades (the majority of amateur players starting from about 35^{th} kyu to 1^{st} kyu), ***dan*** grades (strong amateur players from 1^{st} dan to 7^{th} dan) and professional dan grades (from 1^{st} to 9^{th} professional dan). The difference between two amateur levels corresponds to the number of stones the weaker player is allowed to have already on the board at the start of the game, to have a 50% winning chance.

Appendix to Section II-A

This appendix refers to Diagram 2, and looks in more detail into the options for ○ 's next move.

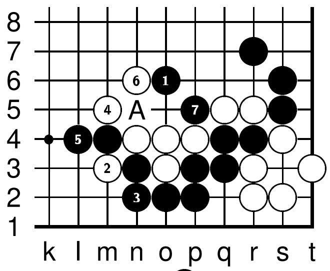

Diagram 6. ● to move after ○ plays elsewhere.

If ○ should play an ordinary move elsewhere (not shown in Dia. 6), then ●'s threat is an even harsher attack on the white group. ○ cannot even play the good shape connection ⑥ because then ● could cut at ❼ and threaten A, which will capture 4 stones. So ○ would have to play ⑥ at A, making very bad shape for his running group.

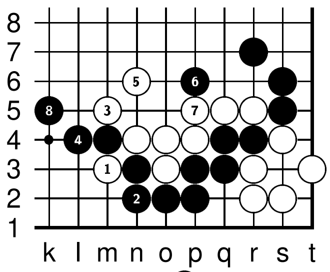

Diagram 7. ○ to move.

Since ○'s center group, which is an important cutting group, is in danger of being attacked severely, normally ○ would defend it immediately, and directly, with a sequence such as in Dia. 7. However, the group would nevertheless remain heavy and under attack, while ● could only profit from attacking.

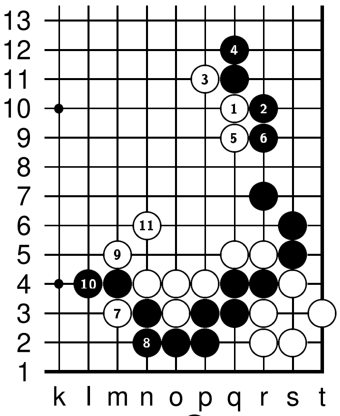

Diagram 8. ○ to move.
A ○ counter attack

① is the game's move. It is a flexible counter-attack that wants to either a) help his weak group by leaning on the single black stone with a sequence like 1 to 6, and installing a foothold towards which the group can later move after starting with 7 to 11, or b) (as it happened in the real game) to get an exchange starting from the game moves ❷ on p5, ③ on r11, ❹ on r12, where ○ sacrifices his center group (at least part of it temporarily) and gets as compensation the reduction of ● 's right side *moyo*.

Conclusion (for a go player):

When answering the opponent's local threat (here: to attack ○ 's center group more severely), if it is not favourable for a player, then he might counter by playing a threat of his own elsewhere (attack on ● 's single stone) if that also indirectly threatens to minimize the opponent's threat.

Appendix to Section II-B

This appendix refers to Diagram 3 and looks in more detail at the options for ● moving next.

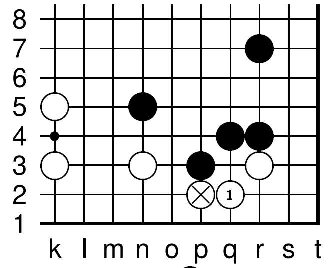

Diagram 9. ○ to move.

The last move – ⊗ – threatened to play ① in Diagram 9 next. This would take away all the lower right corner territory of ●; take away much of his eye-space; give ○ territory in the corner, and provide secure eye-space for the lower middle white group. Therefore ① would be very big. ⊗ is a so-called probing move, i.e., a move that the opponent ● must answer immediately but where he has more than one possible answer to choose from.

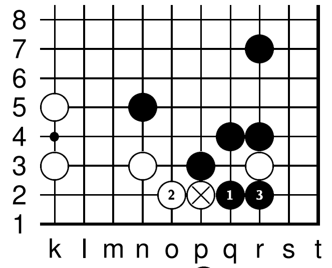

Diagram 10. ● to move.

Diagram 10 shows the game sequence. ❶ is one of the two possible black answers to ⊗. The sequence ❶ to ❸ is a middle game joseki. ● secures his corner while ○ reduces the black corner territory in *sente*, and provides his lower middle group with enough eye-space for two eyes. During the rest of the game, it is inconceivable that ● would get another chance to sacrifice the lower right corner, to attack the white group driving it into the center, or to kill the white group (unless ○ should later decide to sacrifice it).

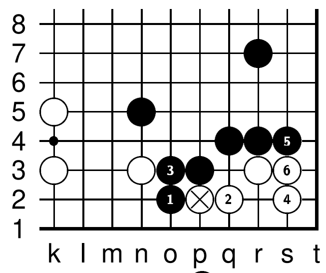

Diagram 11. ● to move.

❶ in Diagram 11 is the other possible reply to ⊗. The sequence 1 to 6 is another middle game joseki. ○ makes small life in the lower right corner in *gote*. In exchange, ● gets a strong cutting group in sente, and will be able to use it to attack ○ 's lower middle group, or even launch a double or multiple attack also against ○ 's right side or upper middle groups (see Diagram 3). During the remaining part of the game, ● cannot get back his lower right corner territory and take away ○ 's territory there (unless ○ should later decide to sacrifice it).

Conclusion:

At move 49, ● is faced with an irreversible decision between just two reasonable moves. ❶ in Diagram 10 chooses the corner territory and drops later easy attacks on the life of ○'s lower middle group. ❶ in Diagram 11 sacrifices the corner territory to get an easy attack on the life of ○'s lower middle group.

Acknowledgements.

The author thanks Robert Jasiek and Sam Owre for consultations and Harry Fearnley for comments on the manuscript. The work was funded by a DARPA seedlings grant.

References

[1] J. Tromp, "Number of legal go positions," preprint http://homepages.cwi.nl/~tromp/go/legal.html.

[2] J. Tromp and G. Farnebäck, "Combinatorics of go," 2007, preprint http://homepages.cwi.nl/~tromp/go/gostate.ps.

[3] O. Hideo, *Opening Theory Made Easy*. The Ishi Press, 1992, p 19.

[4] D. Stern, R. Herbrich, and T. Graepel, "Learning to solve game trees," in *ICML '07: Proceedings of the 24th International Conference on Machine Learning*. ACM Press, 2007, pp. 839–846.

[5] B. Brügmann, "Monte carlo go," 1993, preprint ftp://ftp.cgl.ucsf.edu/pub/pett/go/ladder/mcgo.ps.

[6] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2/3, pp. 235–256, 2002.

[7] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Machine Learning: ECML 2006, Lecture Notes in Artificial Intelligence 4212*, M. S. J. Fuernkranz, T. Scheffer, Ed. Springer, 2006, p. 282293.

[8] B. Bouzy and B. Helmstetter, "Monte-Carlo Go developments," in *Advances in Computer Games. Many Games, Many Challenges. Proceedings of the ICGA / IFIP SG16 10th Advances in Computer Games Conference*, J. van den Herik, H. Iida, and E. Heinz, Eds. Kluwer Academic Publishers, 2004, pp. 159 – 174.

[9] S. Gelly, "Mogo home page," http://www.lri.fr/~gelly/MoGo.htm.

[10] D. Fotland, "Many faces of go home page," http://www.smart-games.com/manyfaces.html.

[11] R. Coulom, "Crazystone home page," http://remi.coulom.free.fr/CrazyStone/.

[12] The Netherlands Organisation for Scientific Research, "French software and Dutch national Supercomputer Huygens establish a new world record in Go," http://nwo.nl/nwohome.nsf/pages/NWOA_7PLLJY_Eng.

[13] T. Mark Hall and J. Fairbairn, "GoGoD database & encyclopaedia," professional games of go on disk, http://www.gogod.co.uk/.