Monte Carlo Go

Bernd Brügmann
Max-Planck-Institute of Physics
Föhringer Ring 6
80805 München, Germany
bruegman@iws170.mppmu.mpg.de

October 9, 1993

Abstract[1]

We present an algorithm for the board game go which attempts to find the best move by simulated annealing. Remarkably, without including any go knowledge beyond the rules, our implementation leads to a playing strength of about 25 kyu on a 9x9 board.

Introduction

The board game go is one of the grand challenges of game related artificial intelligence. While go is a finite, perfect information game, its game tree is prohibitively large for tree searches. The situation in go is perhaps best compared with that in chess, which has been called "the drosophila of artificial intelligence", meaning that a multitude of key problems of artificial intelligence appear in chess and can be studied there. Currently, the best chess playing programs rely heavily on large scale tree searches and use a comparatively small amount of chess knowledge. In go it is the other way round, i.e. the quality of the go knowledge is the determining factor. Due to the structure of go, strong go programs emphasize much more the human way of thinking about go rather than brute force methods for which computers are ideal, and from this point of view one could argue that go is a much more interesting problem than chess. Many important advances have been made in go regarding the application of rule based knowledge representations, pattern recognition and machine learning. For a recent, lucid review of computer go see [**?**]. Several relevant papers on computer go are collected in [**?**].

In this article we want to look at go from a completely different angle: *How would nature play go*? This is not the starting point of a metaphysical discussion. This is an attempt to learn from physics about a method of optimization which nature applies very successfully and "naturally". We will consider simulated annealing [**?**, **?**, **?**, **?**], which under surprisingly general circumstances is able to find approximations to the extrema of a mathematical function. Our topic will be the question of how to formulate the task of finding a good move as a problem in extremization which is tailored to the strengths of simulated annealing.

Perhaps it is worthwhile to describe the main idea right away in very simple terms to indicate the direction in which we are heading. There are essentially three ingredients in computer programs for playing games like go: (i) look-ahead where different sequences of moves are tried, (ii) evaluation where the value of a position is computed without explicitly performing moves, and (iii) an overall scheme for how (i) and (ii) are combined. Here is our proposal:

(i) moves are performed randomly with probabilities assigned by the method of simulated annealing,

(ii) the value of a position in which the game is over is defined by counting, and

(iii) to find the best move in a given position play the game to the very end as suggested by (i) and then evalu-

[1] This article was written with the readership of Computer Go magazine in mind. Therefore I tried to emphasize a clear and complete development of concepts rather than formalism. I would like to thank the referees of the AAAI Symposium on 'Games: Playing, Planning, and Learning' (Fall 1993) for many helpful comments.

ate as in (ii); play many such random games, and the best move will be the one which does best on average.

Let me quickly point out the obvious. The above idea is certainly simple, and I believe that any good idea can be stated simply. However, not every simple idea is a good one, and at first sight the above one may sound ridiculous. The main portion of this article is dedicated to convince the reader that at least this is not necessarily a bad idea. Furthermore, one may suspect that when considered in detail, (i), (ii), and (iii) are not has harmless as they seem. Let me just state that the actual implementation in the go program described below is still quite simple, but one can hope that the more sophisticated the design, the better the results.

The paper is organized as follows. First, we introduce some background of simulated annealing. Then we discuss how it can be applied to the problem at hand, in particular we discuss why our approach could work in principle. In the next section a computer implementation, Gobble, is presented together with some examples for what kind of strengths and weaknesses Gobble displays in games on the 9x9 board. We conclude with a general discussion.

Simulated annealing, a Monte Carlo method for combinatorial optimization

Monte Carlo methods in physics

At the heart of most topics in theoretical physics is a variational principle. Given the phase space of a system, i.e. the space of possible configuration and momentum variables, the dynamics of the system is contained in the definition of the action. The action is a functional on phase space, and the equations of motion are obtained from it by the condition that the path in phase space which the system follows corresponds to an extremum of the action. All the fundamental classical theories like mechanics, thermodynamics, electromagnetism and general relativity are conveniently defined in terms of the action. In fact, given the action one can often construct the corresponding quantum theory straight-forwardly with so called path-integrals. All this we only point out to emphasize how fundamental extremization procedures are in physics.

Monte Carlo methods have their origin in an anology with statistical physics. A statistical system consists typically of a large number of particles, and we are interested in its configuration, say the position and velocity of the particles, and in how it evolves in time. Key aspect of the evolution is that some quantity like the action or the energy is minimized. Let us give a concrete example. Consider a metal near its transition from a liquid to a solid state. A configuration is described by giving the position and velocity of each atom in the metal. The atoms do not move around freely but experience forces due to the combined electric potential created by all atoms. At high enough temperatures, the atoms move around quite randomly, but as the metal is cooled, the atoms move less, and when the metal becomes solid, the atoms are essentially fixed in place by their mutual electromagnetic forces. This process is called annealing. The energetically most favourable configuration for the solid state is that of a regular, crystalline structure.

The surprising fact is that in experiments nature is able to come extremely close to this minimum in energy, i.e. billions of atoms form a perfectly regular lattice — if enough time is allowed for the cooling process. Imagine the atoms moving around in the electric potential like balls on a surface with many dents and bumps. At the minimal energy every atom/ball has come to rest at a place where the potential has a local minimum. If the atoms are slowed down too quickly, however, they probably have not found the smallest local minima possible, i.e. the metal solidifies in a less regular, amorphous state in which the total potential energy is larger than in a crystal.

In Monte Carlo simulations one simulates the evolution of a statistical system on a computer [?]. One does not attempt to solve the terribly complicated equations of motion for each particle exactly, but follows a probabilistic evolution that stays close to but not exactly at the minimum of the action. First of all, one chooses a suitable description of all possible system configurations. Second, one needs a set of elementary moves which are ergodic, that is, changes in the configurations which enable one to cover the space of all configurations step by step. Third, for a given configuration moves are suggested completely randomly. Finally, the physically correct evolution is approximated by accepting a suggested random move with a probability that depends on whether the move increases or decreases the action.

So let us give an example for what these probabilities should be for a system in thermal equilibrium. In this case the temperature is the same everywhere and the energy is constant. Notice that even if the total energy or the temperature is fixed, individual particles may behave differently. In many cases the probability $p(E)$ that a single particle has energy E when the system is in thermal equilibrium at temperature T is given by the Boltzmann probability distribution,

$$p(E) \sim \exp(-E/kT), \tag{1}$$

where k is Boltzmann's constant. For such systems the probabilistic evolution can be given by the following rules (Metropolis et al. 1953 [?]). If a proposed move lowers the energy, it is accepted. If a proposed move raises the energy by $\Delta E > 0$, it is accepted with probability $p = \exp(-\Delta E/kT)$ (i.e. $1 > p > 0$). Note that every move, even very 'bad' ones, have non-vanishing probability.

Simulated annealing is special in that 'cooling' is introduced [?]. The temperature is lowered towards 0, and in that limit moves that raise the energy are almost never accepted by the Metropolis algorithm. Again, if the system is cooled very slowly so that it is effectively always in thermal equilibrium, the system is also close to its minimal energy. If cooling happens too quickly, many particles are caught after moves away from the minimum and the total energy is not minimal. What we have to add to the prescription for Monte Carlo simulations of systems in thermal equilibrium is an "annealing schedule" which defines how fast the temperature is lowered.

Let us end our discussion of physical systems at this point. Monte Carlo simulations are widely and successfully used in the physics of statistical systems. They are that successful, in fact, that non-statistical theories like QCD (quantum chromo dynamics), the theory of elementary particles, has been turned into a statistical theory to make it tractable. At least one other idea of physics, the idea to use potentials to measure influence, has been successfully applied in computer go [?, ?].

Simulated annealing for mathematical problems

We now want to focus on the features of simulated annealing which make it a successful method of extremization for many mathematical problems [?]. Consider an arbitrary function of several variables. If the function is simple enough and depends only on a small number of variables, there are very efficient exact numerical methods to find its local minima. They all are based on essentially the same principle. Pick initial values for each variable and compute the function. Change the variables slightly in such a way that the function value decreases. Repetition allows one to come arbitrarily close to a local minimum. Different algorithms have been invented to optimize the progress at each iteration for different classes of functions.

Whenever one of these methods is applicable, e.g. find the minimum of $f(x) = x^2$, it usually works much better than simulated annealing, but there are several types of problems where there is virtually no alternative to simulated annealing. As it turns out, these exact algorithms are 'greedy' by design, that is given a starting point they greedily walk downhill (following the steepest descent) until they have found a local minimum. But if one is looking for a global minimum among many local ones, and if one does not know before-hand where to look for it, one will never find it. Simulated annealing does much better in finding at least an approximate solution to a global minimum since it allows uphill moves. Another area were exact numerical methods fail is when the function depends on a large number of variables. Finally, simulated annealing works equally well for non-differentiable (even discontinuous) functions while exact methods often require the existence of the derivative of the function.

Perhaps the reader feels that already far too much as been said about the physical and mathematical background of simulated annealing (see [?] for the mathematical theory behind simulated annealing). After all, consider the simple prescription for simulated annealing to find the minimum of a function of many variables that can be distilled from the above. Given is a space of configurations described by some variables and a function which assigns a number to each configuration. To approximate the global minimum we perform small random moves in configuration space. If the function decreases, we accept the move. If the function increases, we accept the move with a probability which decreases exponentially with the increase in the function and also decreases exponentially with the inverse of the temperature. The temperature is decreased according to our annealing schedule which should leave enough time for proper thermalization. The reason I gave so much background is that even though the procedure of simulated annealing may look trivial, it should not be underestimated.

We conclude this section with two examples from combinatorial optimization, which is in many ways similar to our goal, to analyze game trees. Simulated annealing has, as far as practical applications are concerned, solved the famous 'traveling salesman problem' [**?**, **?**]. The problem is to find the shortest path that connects N cities. The same type of problem arises when one wants to optimize the layout of integrated circuits, or in a phone system with several possibilities to locate the switches. These problems are examples for combinatorial optimization since the configuration space on which a function (length of path) is to be minimized is a discrete, factorially large space. In the traveling salesman problem, there are $N! = N * (N-1) * \ldots * 1$ possibilities to make an ordered list of N cities. The computation time to find the solution by an exhaustive search increases exponentially with N. Simulated annealing achieves a good approximation to the optimal solution in computing time which grows with N only as a small power of N. The procedure starts with an arbitrary path. Two types of local changes are performed. One move is to reverse the order of several cities which are next to each other on the path, and the other move is to remove several neighboring cities from the list and move them to a different position in the list.

For a second example see [**?**], where the problem is to find the arrangements of 25 playing cards in a 5x5 tableau such that the value of the rows, columns, and diagonals interpreted as hands for poker is maximized. Again, simulated annealing is successful and much faster and simpler than other methods.

Simulated annealing for tree searches

The problem

After having gained some intuition about simulated annealing, let us now try to apply our knowledge to tree searches [**?**]. Given a game position, we are looking for the best move. Tree searches are similar to combinatorial optimization problems in that an extremely large number of alternatives has to be considered for an exhaustive search.

Let us first focus on aspects of tree searches which lead to the failure of some natural suggestions for the implementation of simulated annealing before we argue why the suggestion given in the introduction could work in principle. Faced with a game tree which is far to large for complete exploration, some sort of pruning becomes essential. There are different algorithmical techniques which allow one to ignore parts of the game tree, for example, α-β pruning. In fact, the evaluation function which is invoked at the deepest level of the tree to find the value of a position is also some sort of pruning. The outcome of the remainder of the tree is estimated by applying game heuristics. For example in go, certain patterns are good because they have been proven to be so in previous games, and to increase one's influence on the board is advantageous because experience shows that influence helps in the long run (down the tree).

A first proposal could therefore be to apply simulated annealing to pruning. At each branching, evaluate the move possibilities from go knowledge but pursue moves only with a certain probability. This is actually an old idea, and I do not believe that the exponential probabilities from simulated annealing would give anything useful. The main problem is that the game tree of go is still far too large even if only two moves are considered at each level. Go knowledge remains essential.

After completion of this work the author found [**?**], where Andrei Grigoriev reports that simulated annealing can be successfully applied to tree searches in Gomoku and Renju, and in fact suggests that the method could also be useful in go. His approach is based on exactly the idea that I just described, i.e. to grow game trees stochastically. His annealing schedule consists of lowering the temperature with the depth of the tree. When I was thinking about how to map the tree search problem into a problem that is well-suited for simulated annealing, I conjectured that the game tree in go is too large for such an approach. I wouldn't mind at all to be proven wrong in this point.

Getting back to combinatorial optimization where simulated annealing succeeded in optimizing long lists of objects, to find the best move really means to find the best move considering optimal play of both players, i.e. we want to optimize a sequence of moves. Maybe we can consider *entire games* as the object of optimization? This would make the evaluation trivial, since at the end of a game counting gives a precise value.

Game trees are, however, profoundly different from the traveling salesman problem because there are two competing parties playing the game (trees are not paths). This fact makes it impossible to find local moves in the configuration space of the type used in the traveling salesman problem. We are really dealing with the optimization of two sequences of moves, one for Black, one for White. The problem is that any local change in the order of moves that Black makes has a *non-local* effect on the game tree. For any move that Black chooses at a given point in the game over another, the optimal play for both players will in general have changed completely for the remainder of the game. Think of the value of a move for Black as a function of the order in which Black intends to play his moves. For a numbered list of such orderings, the value for ordering $1, 2, \ldots$ is likely to look like a sequence of random numbers because of the different, optimal responses by White. It seems that simulated annealing cannot be applied in this case since there are no local moves and under changes in the list of moves the value function behaves so discontinuously.

One possible solution

The question is, can anything be learned from playing random games? The answer is yes for the following reason. There will be some moves which are good *no matter when they are played*, if a player actually gets to play them. For example, if a certain move of Black makes a group of Black stones live, but if White prevents that move, Black's group dies, the average result of the games differs by the corresponding value. To what extent this heuristics is valid depends of course on the game that is considered.

A general rule taught to novices in go is to "always look for the biggest move on the board". One separates the situation on the go board into local move exchanges and assigns to the move initiating play in a region the estimated benefit explicitly in the number of points gained or lost. The suggestion is to play the move with the best value first. Of course, this can only be a rule of thumb because often the moves are not completely independent of each other. Furthermore, it is important which player will have the initiative after a sequence of moves (sente), i.e. a smaller move which does not loose sente can be played first. For our goal it is of interest that go nevertheless allows such localization to a high degree, much more so than it is for example the case in chess.

Under these circumstances, playing random games to the end is not a wasted effort since good moves may be detected. To identify good moves whenever they happen in a game tree is also a standard tool of pruning. The α-β algorithm has to consider the minimal number of moves if at each branching the best move is tried first. The so-called killer heuristic suggests to try those moves first which are currently the best moves anywhere in the tree, and significant improvements can be achieved [**?**].

For these reasons we adopt the following strategy for playing random games. Each player decides beforehand in what order he wants to play his moves (taking into account that pieces may be played several times onto the same field after captures). A game is played to the end such that if a move in the predetermined list is not possible, the next one is used. Each move is assigned the average value of all the games in which it was played. This value is initially set to zero and then updated after every game. After every update the moves are ordered according to their average values. The strategy is therefore that moves which are most successful on average are deemed most important to be played first. After a large number of games both Black and White will have settled on a sequence of moves which are most successful with regard to the other. The best move for the one to move first is the first one in his list.

As described so far, we are considering a greedy algorithm which will tend to remain at local extrema. The modification for simulated annealing is to introduce a finite probability for a move to be played out of order which depends on the exponential of the value difference divided by a parameter playing the role of temperature. As before, when the temperature becomes zero, we have frozen all deviations from the greedy strategy.

The algorithm given in the introduction provides the framework for several variations of this idea. The above construction could be called the zeroth order approach in the following sense. Obviously, there are some moves which are only good as answers to particular other moves. To incorporate such interdependencies one can store not just the average value of each move but the average value after a particular other move was played first. This could be called the first order approach, and there clearly is a generalization to n-th order. In the limit of large n the complete game information has been stored and the method becomes equivalent to an exhaustive tree search. This means that in the limit the method works perfectly (if inefficiently).

The principle limitation is that the more information we want to use from the games that are played, the more games have to be played to make this information meaningful. A general fact about the statistics of the data is that the error in the average value Δv is typically proportional to one over the square root of the number of games n,

$$\Delta v \sim \frac{1}{\sqrt{n}}. \tag{2}$$

(Other numerical methods work usually much better.) This implies that to determine the value of a move to within 1 point when the value in the random games fluctuates by 100 we have to play on the order of 10000 games! For the actual tests presented below a few hundred games turned out to be feasible. Therefore, higher order considerations have to be postponed.

What our algorithm is expected to accomplish is to play *perfect* games against a specified, *dumb* opponent. The games can be made perfect by collecting enough statistics (cooling slowly enough). How dumb the opponent is depends on how much information can be extracted from the games. Playing a large number of games against a strong, perhaps human opponent could possibly let the algorithm play a better game against any opponent. The problem is, of course, that the only opponent available is the algorithm itself.

This concludes our motivation for the algorithm introduced in the introduction. The crucial point of our proposal for applying simulated annealing to tree searches in go is to replace the strict move order (as familiar from combinatorial optimization) by the concept of timeliness or priority of each possible move. Hopefully, this representation of a tree search leads to changes of configuration that simulated annealing can manage. The other novel aspect (as far as computer go is concerned) is to try to utilize the strengths of simulated annealing by playing complete games as a substitute for go heuristics. Enough of theoretical musings, does it work in practice?

Gobble: a program plays Monte Carlo go

In this section we present results obtained with the computer program Gobble (version 1.0) that plays go on the 9x9 board using simulated annealing to find “the best” move. The point is that nothing more than simulated annealing is implemented on top of the bare go playing rules to allow us to study the method.

We will not describe the program in detail, since this would obscure rather than help the general discussion. Anyone with some experience in programming will agree that an implementation of the algorithm should be straightforward. One technical issue we have to mention is how we define the end of a game (recall that we want to play games routinely to the very end). On first sight this is at least intuitively clear to human players, but while the rules for playing moves in go are elegant and simple, rules for when the game ends and how the result is to be counted are surprisingly complex. The latter two issues are related, and there are in fact different (Japanese, Chinese, ...) rules which in rare cases like a multilple ko or special seki give slightly different results.

We adopt for the moment the following prescription. The computer only passes if either no legal move is available or all legal moves reduce the eye space of one of its groups from two to one. The (over-) simplification lies in the definition of an eye, which is defined to be a field whose direct neighbors are all of the same color and whose diagonal neighbors contain no more than 1 stone of the opposite color (0 for border and corner fields). The reader may convince himself that this definition of an eye is correct for most situations, excluding sekis. When both sides have to pass, the game is over and the result is determined via Chinese counting. These are standard rules except that the computer player does not realize that stones in a seki are alive. Sekis can be dealt with once passing is allowed as an option, but we leave it at that for simplicity and speed.

One could argue that the rule to never fill in one of two eyes represents go knowledge. Let us point out that there is another instance where simple, logical rules for playing go are needed, and an equivalent rule is defined. In Berlekamp's mathematical go [?] a group with two true eyes becomes "immortalized", i.e. even when its liberties are filled in it is not removed from the board. Furthermore, the go knowledge that is implicitly included by this rule is worth very little in practice. On the one hand, the rule is necessary for technical reasons, and even though (if passing was implemented) the program could figure out the rule by itself, that would be very inefficient. On the other hand, just knowing not to kill its own groups once they have two eyes does not help a program at all if it otherwise plays random moves. So the two eye rule is necessary for the program to be able to play acceptable go at all, but it is by far not sufficient to lift the playing strength of a program playing random moves (worse than 50 kyu) to 30 kyu.

Let us also mention a few points about the probabilistic aspects of the program and the annealing schedule. Determination of a good annealing schedule is a matter of experimentation [?], here is what we found useful. First we order the moves strictly by value, which corresponds to a probability of 1 that a move with better value is played first. Then we sweep over the list once from best to worst move and switch the order of two neighboring moves with probability p_{swap}. The probability $p(n)$ that a move is shifted $n \geq 1$ steps down the list is

$$p(n) \quad = \quad (p_{swap})^n = \exp(-n/T), \qquad (3)$$

$$T \quad = \quad -1/\ln p_{swap} \geq 0. \qquad (4)$$

The annealing schedule is to lower p_{swap} to 0 linearly with the number of games and set $p_{swap} = 0$ (i.e. $T = 0$) for a few games at the end to settle in the nearest local extremum. Notice that this is not the Metropolis algorithm (which is not the only choice possible anyway). We found that for the few games we play it didn't seem to matter whether p_{swap} depends on the value difference or not. In addition a few percent of all moves are performed completely randomly to avoid infinite (or long) loops. Further experimentation in conjunction with a measure of how well the annealing process works is needed.

Gobble was developed on a 286/16 PC, which translates into very slow and powerless by todays workstation standards. We restrict our attention to the 9x9 board. One important aspect of statistical methods like simulated annealing is that some minimal amount of data is needed to find the signal in the noise. Let us denote by strategy A the algorithm of order 0 described in the previous section. Strategy A requires several hundred games to be played before the data is reasonably reliable which takes on the order of 1 minute on the PC and a few seconds on a IBM/RISC workstation. (Typical physics applications in Monte Carlo are allowed to run for days or weeks, but we have game play with humans in mind.)

A simple example

Here is our first example! Figure 1 shows a timeless beautiful position: the empty board, Black to move. After playing 10000 random games, Gobble assigned to each field a number, ranging from 0.9 to 5.3 in this case. These values are the average game results for all games in which Black was the first to move onto a field. For example, if Black was the first to take the center field, whether it is in move 5 or 50, all such games resulted in an average point count at the end of the game of 5.2 points. In particular, 5.2 is not the average value for games with Black's first move at the center.

The main feature is that moves near the center give better results than moves onto the border and corners. This result is non-trivial, indicating that strategy A is able to find one of go's simple rules of thumb. The absolute worth of the first move amounts to about 5 points, which is lower than it should be, but remember, these 5 points are just the advantage that Gobble knows to derive from the first move. During the computation it is nice to watch how the average value of all games increases steadily as Gobble becomes smarter and tries better moves.

This one example already gives an impression of the statistical errors involved. The board is symmetric, but equivalent moves do not get exactly the same values (consider the corners, for example). An estimate for the error could be $\Delta v = \pm 0.6$. The values near the center are more evenly distributed because good moves are more often tried at the beginning of games which leads to more consistent results. Bad moves are played near the end of the game when they tend to be irrelevant and less correlated with the outcome of the game.

Test results for games on the 9x9 board

To determine the strength of Gobble we had it play several games on the 9x9 board against Many Faces of Go, the excellent go program by David Fotland. Many Faces of Go is the North American computer go champion. In 1986 [?], Fotland writes, "I would love to have other computer programs to compete with. ... It is hard for me to tell how good it really is." To have Many Faces of Go available (public domain, xgo.s800, rev. 7.34) has been very helpful since it provides a strong, unvarying opponent. Fotland gives this version of Many Faces of Go a 16 kyu rating [?]. The latest version of Many Faces of Go (rev 8.3 running on a HP Snake workstation at level 15) has earned a respectable 12 kyu rating in hundreds of games against human opponents on the Internet Go Server.

Table 1 shows the results of twenty games with Gobble playing Black at different handicaps and Many Faces of Go playing White at level 10 (medium strength). For strategy A, Gobble played 500–1000 games to find its moves. Strategy B is an extension of strategy A to the first halfmove. In the left column for strategy B, 400 games were played and then the 4 best moves performed and evaluated with another 400 games each. In the rightmost column, games 1 to 3 were played with 500 games per evaluation and 3 trial moves, and for games 4 and 5 the number of games was doubled. Handicap stones are placed on the star points.

One should keep in mind that such a small sample can only give a rough overview. The sample is representative in the sense that except for a few test games used to adjust the parameters all games the author had time to play are shown. The scores vary a lot because the players are weak by human standards, and the board-size is small. Nevertheless, there is clear and perhaps unexpected evidence that *Gobble plays respectable novice go.* Gobble's strength for strategy B seems to be about 25 kyu. The exact number is unimportant, what matters is that Gobble plays better than a human beginner who typically is 30–35 kyu after a few games.

Strategy A requires three handicap stones to have a chance against Many Faces of Go, but although this is a huge advantage, Gobble displays a basic capability to stay alive and also to kill its opponent. Strategy B brings an improvement of about one handicap stone. Gobble's strategy, stated in general terms, is to build a strong center, while Many Faces of Go tries to increase its influence along the borders. Gobble only wins a game if it succeeds in cutting off and killing one of its opponents groups. Otherwise, Many Faces of Go wins because of its superior territorial skills.

Analysis of a game between Many Faces of Go and Gobble

It is instructive to analyze one of the games in detail to get a feeling for Gobble's non-human, probalistic way of "thinking". Computer go is usually not pretty, even to the eyes of the author who is a 13 kyu, but there are some interesting observations to be made. Remember that Gobble lacks even the simplest go heuristics, and any signs of go understanding produced by our algorithm are therefore of interest.

Figures 2 and 3 show the game record of one of the games played with strategy B and a two stone handicap. Black wins by 9 points.

Move 2. Figure 4 shows the board after White moved onto 2. To give an impression of how Gobble evaluates the position, the six highest and lowest values according to strategy A are shown. Border moves are considered bad by Gobble while moves somewhere in the center or near White 2 are preferred. Incidentally, a move onto the field now occupied by White, i.e. after White's stone has been captured, is worth about 40 points.

Let us again emphasize that there are noticable random fluctuations in the result, although 2000 random games were played. Figure 5 shows exactly the same position for a second evaluation. Notice that the average value has shifted by 2 points and only one move made it into the top six twice, and for some reason the idea to attack White's stone does not play a role. The lesson is that there may be many moves which cannot be differentiated by Gobble. However, among the top six there always seems to be one or two good moves on an objective scale, which is remarkable in itself. The challenge is to find them, which is one motivation for strategy B. As with human go players, many will recognize that a move should be played "somewhere in the center", fewer are able to pinpoint the best move.

Move 4. White has knowledge about fuseki.

Move 5. Definitely much too tight. We tried to reproduce this move with strategy A, but failed. 400 games per move in strategy B may allow to much randomness. Also see moves 9 and 11.

Move 6, 7, 8. Nice shape.

Move 9, 11. These moves look good to Gobble because on average a connected group with many liberties is more likely to survive than separate stones.

Move 12. Defends territory.

Move 13. If connected pieces are strong, a cut produces a weakness.

Move 14–24. White defends its territory while Black is cluttering the center.

Move 25. Having one eye under all circumstances is another feature which is valuable on average, and this move is sente.

Move 26. By now White has a big advantage and it is still open whether Black is capable of making a second eye.

Move 27, 28. Black tries to make territory for the first time, White takes it away at the other end of the board.

Move 29, 30. A futile invasion.

Move 31. Black probably considers White's stones in the lower right corner as half dead, simply because with a probabilistic algorithm there is always a chance that a player does not notice an atari. This chance may be slim, but if the gain involved is large enough, such a move will be played.

Move 32. Uncharacteristically poor move for White, the loosing move.

Move 33. Black is able to take advantage of White's mistake.

Let us pause here for a moment and consider the values strategy A would assign to this position. Figure 6 shows that Black 33 clearly stands out as the best move for Black with 20.2 points (gained by starting from this position). The worst move is worth 12.2 points. There is a large number of moves worth about 15 points. The difference of 5 points between these moves and the best move can only mean that Black wrongly considers part of White's stones in the lower right corner as dead.

Figure 7 shows the situation before White 32. That the best move is the one worth 5.7 points (for White, the lower the value the better) indicates that Gobble is consistent in that he makes the same mistake as in the evaluation for Black 33. Still, this move is much better than White 32 which scores a 10.5. The move values in the upper left corner suggest that Gobble does not understand the position that Many Faces of Go has build. We return to the game.

Move 47 (figure 3). This move should of course be played at 51. The bonus from an oversight by White must have outweight the territorial loss in the upper right corner. The single most important reason that Gobble looses in the endgame are such nonsense threats, while defending the borders is consistently underrated.

Move 49, 51. More meaningless ataris.

Move 57, 59, 61, 65. Meaningless.

Move 63, 69, 71, 73. At times when no competing flawed goals exist, Gobble's statistic is good enough to detect even small gains.

The remaining moves are not shown since they do not affect the outcome of the game.

Let us summarize what we have learned from this example. Gobble displays its own brand of typical computer go: surprising (relative) strengths in some areas and glaring weaknesses in others. Examples are life-death and territorial evaluations, respectively. Based on these observations there certainly are many natural suggestions for improvement of Gobble's strategy, some of which are mentioned in the following discussion.

Discussion and Conclusion

First we pointed out that there is a powerful optimization scheme in nature, known on the computer as simulated annealing, which has not been applied in computer go. We proceeded to adapt simulated annealing to the problem of finding the best move in the game tree of go. The adaptation is based on the observation that there is a certain move reorder invariance in go. There are two key elements to our proposal: 1) We suggest that the timeliness of a move is a good objective function for simulated annealing; 2) Playing games to the very end can substitute for go heuristics. Finally, we presented first test results of a computer implementation of our algorithm that plays go clearly above the human beginner's level of 30 kyu.

Since the idea seems to work in practice, we would like to point out several interesting avenues for further exploration. First let us comment about the relevance of the algorithm for two features of computer go, tree searches and evaluation functions. A lot of research has been done on α-β pruning and related algorithms. One should perform a quantitative study of how simulated annealing performs for the different orders of implementation (one move, two move, ... sequences), and find out how well the method can do in practice (see Chs. 1–3 of [**?**]). Recall that for the traveling salesman problem the computational effort was reduced tremendously. Here we face the additional problem that our objective function for tree searches is an approximation.

As far as evaluation functions are concerned, an age old dream of game related artificial intelligence is to tell the machine the rules, and nothing else, and have it play a perfect game. Although Gobble plays very poorly by human standards, whatever "understanding" of go Gobble has was not fed in by humans in the form of go heuristics (except perhaps for a small benefit from the two eye rule). The fact that points are best counted when the game is over apparently is important enough to matter even through the haze of random games and poor statistics. It is open whether this phenomenon can be utilized, for example, on larger boards.

So far we have studied simulated annealing only in its pure form to identify its characteristica. We definitely do not want to suggest that this is the best way to program go. Strong go programs like Many Faces of Go succeed because they apply a collection of ideas like pattern matching, tactical analysis and influence potentials. The most difficult problem is to combine all methods into one well-rounded computer go player. Just because Gobble shows some aptitude in tree searches and evaluation does not mean that this is the optimal strategy under all circumstances. Whenever exact methods are available, say local or small scale tactical searches (e.g. for ladders), they are much superior to statistical methods.

For example, we noticed that Gobble is particularly weak when it comes to defending its territory near borders. The correct moves are elementary patterns most good programs know. Indeed, one promising way to improve Gobble could be the addition of a pattern library. This could be implemented on the top level of the move tree for alternatives which are hard to distinguish. The same applies to all standard techniques. A pattern library might have an advantage over other methods if pattern matching is sufficiently fast to allow its inclusion *at all stages* of the random games. The stronger the computer player is that plays the random games, the more representative the final results are.

On the technical side, let us mention that simulated annealing is well-suited for special computer hardware. Computers are good at simple repetitive tasks, which is in contrast to the requirements of many applications in artificial intelligence where the main problem is programming and manipulating complicated structures, e.g. in expert systems for go. Monte Carlo simulations are a prime example for number crunching. Simulated annealing is easily implemented on parallel computers [**?**], which are the most powerful and affordable computing resource today. Another approach to increase the raw computing speed in Gobble combined with a pattern library is dedicated hardware. In fact, there is an ongoing project to design a fast VLSI chip which incorporates precisely the two functions which are most time consuming in such an algorithm, move generation and pattern matching [**?**].

To summarize, simulated annealing seems to be a promising idea about how to approach computer go from a new direction. Simulated annealing is certainly not a magic wand which resolves all the profound problems related to computer go. But we believe it is quite possible that simulated annealing could become an integral part of the go programmer's repertoire.

References

[1] D. Erbach, Computers and Go, in The Go Player's Almanac, ed. R. Bozulich (The Ishi Press, 1992) 205–17

[2] D. Levy (ed.), Computer Games II (Springer, 1988)

[3] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, Numerical Recipes in C (Cambridge University Press, 1988)

[4] R. Otten and L. van Ginneken, The simulated annealing algorithm (Kluwer Academic Publ., 1989)

[5] S. Kirkpatrick, C. Gelatt, and M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–80; S. Kirkpatrick, Optimization by simulated annealing: quantitative studies, J. Stat. Phys. 34 (1984) 975–86

[6] M.P. McLaughlin, Simulated annealing: This algorithm may be one of the best solutions to the problem of combinatorial optimization, Dr. Dobb's Journal (Sept. 1989) 26–37, 88–91

[7] K. Binder, D. Heermann, Monte Carlo simulation in statistical physics: an introduction (Springer, 1992)

[8] To determine the influence of stones on a board, a potential function can be computed very much like the electric potential due to charged conductors. This is a good anology, since all connected parts of a perfect conductor are always at the same potential, very much like in go where each stone of a chain of stones has the same number of liberties assigned to it. Physics may be able to teach as more about how such potentials are computed.

[9] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, Equations of state calculations by fast computing machines, J. Chem. Phys. 22 (1953) 1087-92

[10] P.W. Frey, The alpha-beta algorithm: Incremental uptdating, well behaved evaluation functions, and non-speculative forward pruning, in Computer Game Playing: Theory and Practice, ed. M.A. Bramer (Ellis Horwood Ltd., 1983) 285–289

[11] The author is not aware of any previous attempts to apply simulated annealing to game trees as they are present in go, but see [**?**]. There is an isolated comment in [**?**] about random number generators suitable for "Monte Carlo exploration of binary trees". Any references are welcome!

[12] A. Grigoriev, Artificial intelligence or stochastic relaxation: simulated annealing challenge, in Heuristic programming in artificial intelligence 2, ed. D. Levy and D. Beal (Ellis Horwood Ltd., 1991) 210 – 16

[13] R. High, Mathematical Go, in The Go Player's Almanac, ed. R. Bozulich (The Ishi Press, 1992) 218–24

[14] D. Fotland, The program G2, Computer Go 1 (1986) 10–6

[15] D. Fotland, private communication

[16] R. Azencott (ed.), Simulated annealing: parallelization techniques (Wiley, 1992)

[17] Rob H. Tu, Developing a VLSI go game processor, a Berkeley class project (newsreader post 3/93)