Pursuing abstract goals in the game of Go

A.B. Meijer & H. Koppelaar
Delft University of Technology. Faculty ITS. Section Mediamatics.
Mekelweg 4, P.O.Box 356, 2600 AJ, Delft, The Netherlands.
{a.b.meijer,h.koppelaar}@its.tudelft.nl

August 31, 2001

Abstract

Reasoning and planning at different levels of abstraction is an important skill in the game of Go, for both human and computer players. Over the years, adversarial planning approaches have become increasingly popular for dealing with large search spaces of two player games such as Go. This article describes a different approach to adversarial planning, based on an analysis of human Go playing. Abstract goals are decomposed by searching the space of lower level features. Goal checking is done using an abstraction operator.

1 Introduction

The most popular approach to computer game playing is based on $\alpha - \beta$ search or some other game tree search algorithm and position evaluation functions. These so called *data-driven* approaches have shown excellent results in games with a modest search space and for which position evaluation is not too complex. Chess is an example of a game in which significant successes have been achieved. However, it has been far less fruitful as a paradigm for solving Go. This is due to the large branching factor of the game tree of Go. On average, this is around 235 in Go, compared to around 35 in Chess. Another reason is that position evaluation in Go is much more difficult than in Chess, where the number of pieces on the board is already an indicative measure. In evaluating a Go position, one has to judge a complex balance between locally and globally interacting structures.

An alternative to the data-driven approach is *goal-driven* search, in which an agent formulates goals and searches the space of possible goal expansions. When this idea is used to model two opposing agents, it is called adversarial planning. This method has the important property of greatly reducing the branching factor, but it has the disadvantage that the space of plans has to be modelled and hand coded. We propose a method that is both data- and goal-driven and should overcome this disadvantage, while retaining the advantage of a reduced search space. It is goal-driven in the sense that goals are formulated that are tried to be decomposed, but data-driven in the sense that the decomposition method performs search in the space of abstract moves.

Abstract moves consist of changing the feature vectors, which we use to model the tactical and strategic aspects of the game. We use a hierarchy of boards and associated feature vectors that contain properties of the objects on the board at different levels of granularity. Such a hierarchy is needed to enhance decomposition by search and moreover, it can be used in a model of human reasoning about Go. We wanted to mimic human Go playing, since even amateur Go players are far superior to the best current computer Go programs.

This paper is organized as follows. Section 2 deals with the way humans reason effectively about Go. In section 3 we explain adversarial planning and explain our motivations for an adaptation of this approach. We discuss the board hierarchy and the feature vectors in section 4 and elucidate our choice for a graph representation. Our planning method is subject of section 5.

2 A model of human reasoning on Go

Human reasoning on Go is performed at roughly two levels of abstraction, a strategic and a tactical level. In reasoning on the strategic level one considers the whole board situation and focuses on the global interactions between different structures on the board. Tactical reasoning aims at finding locally good moves. We constructed a simple cognitive model of human reasoning on Go, consisting of the following steps. First (*abstraction*), the situation on the board is judged. Global structures called groups are recognized. Weak points in the groups and opportunities for both sides are investigated. Second (*goal formulation*), based on the relative strength of the groups, the most important areas to play in are determined. Goals are formulated to change the situation in this area. Third (*pursuing goals*), one tries to find locally a move that satisfies the goal, considering just a few good ways of resistance by the opponent. It sometimes happens in professional play that, even though some area is clearly the most important, the player moves attention to a different area because he cannot find a move that is locally good. This exception to the rule indicates that professional play indeed resembles this three step approach.

In order to mimic the effective way of human Go playing, it is also our approach to separate strategic and tactical reasoning. In order to do so, we will use different board representations with an appropriate level of granularity for each type of reasoning. On the tactical board we represent local objects and features, such as strings of adjacent stones and their number of liberties. On the strategic board we only use global objects and features, like groups and their degree of safety.

Besides mimicking human Go playing an advantage of using such a hierarchy of boards lies in the way of adding knowledge to your program. Abstract board representations by definition contain knowledge. This is either explict knowledge that has been used to make an abstraction step, or it is implicit in the form of some search that has been performed in order to deduce an inevitable outcome of some local situation, for example, stones have been shown to be connectable. Using abstract boards prevents that too much knowledge of the program is put in the evaluation function alone, making the program less manageable. Instead

of having to write an evaluation function that performs a whole board analysis using a concrete board representation, it is possible to use the abstract board for strategic evaluation. Such an approach fits in the *divide and conquer* paradigm. Although we propose a goal-driven approach, which in principle would make an evaluation function redundant, an evaluation function will still be used for some purposes. The main purpose is to check whether achieving a particular goal will indeed lead to a satisfactory board situation. This will lead to only very few calls of the evaluation function, compared to $\alpha - \beta$ search.

3 Adversarial planning

One goal-driven approach that has been applied to Go is adversarial planning [3,7]. Adversarial planning is an extension of Hierarchical Task Network (HTN) planning [1] into the domain of two opposing agents. We will first briefly discuss the principles of HTN Planning, loosely following [7], then briefly explaining the extension to adversarial domains.

HTN planning is based on three types of objects: *Goals, Operators* and *Plan Schemas*. Operators are actions which can be performed in the world, such as playing a move on a Go board. Goals are more abstract aims in the world such as "Kill enemy group". Schemas specify the subgoals which must be achieved in order to satisfy the goal. For example, the following schema expresses the fact that G can be achieved by a conjunction of subgoals G_1, G_2 and G_3: $G \Rightarrow G_1 + G_2 + G_3$. The G_i should be at a lower level of abstraction that G. Operators are at the lowest level of abstraction.

Given these three types of object, HTN planning starts with an initial world state and a set of goals which form the initial abstract plan. The plan is refined step by step by expanding the goals within it. Goals are expanded by selecting one of possibly many schemas whose antecedent (the G above) matches the chosen goal, and replacing the instance of G in the current plan by the subgoals (the G_i above). The process continues until all goals have been expanded into sets of operators and when possible conflicts have been resolved. The sequence of operators should, upon execution in the initial world state, lead to the achievement of the planner's goals in the world.

The extension of HTN planning into adversarial domains is non-trivial since plans are no longer sequences of action but trees of contingencies which take into account the opponent's actions. The adversarial planner GoBI [7] models two opposing agents (named Alpha and Beta). To solve a problem in the domain, each agent is given a set of input goals to achieve and has the additional task of preventing the other agent from achieving its goals. A planning step involves selecting an abstract goal and repeatedly expanding it, just like in HTN planning. Once one of the agents (Alpha say) has achieved all of its goals, it knows that it must have satisfied its top level goals and performs the chosen action in a world model. The turn is now passed to Beta, which has his own agenda of goals that are partially expanded by a particular choice between decomposition schemas. Now that Alpha has achieved his goals, Beta tries to force backtracking by changing

one of his prior choices for a decomposition schema, in order to thwart Alpha's plans. It expands newly introduced goals until it finds a move that satisfies them, plays this move in the world model and passes the turn back to Alpha. Alpha is forced to backtrack, because he is now faced with a renewed world model in which some of his goals have become unresolved.

The backtracking activity explores the various interacting plans Alpha and Beta have for the situation and creates a contingency tree of moves. The planning activity comes to searching this tree, whose branching factor is the average number of different decomposition schemas that are at hand for a single plan. In general, this a large reduction compared to $\alpha - \beta$ search.

Adversarial planning has a number of advantages. The biggest is the reduction of the search space. A second advantage is that a lot of Go knowledge is expressed in proverbs, such as "Death lies in the hane". This kind of knowledge is well suited to be encoded in a goal decomposition scheme. A third advantage is that there is less need for global evaluation functions: full board evaluation reduces to checking whether low level goals have been achieved.

One major disadvantage of adversarial planning is the fact that the hierarchical space of plans has to be modelled and hand coded. In order to be able to play a complete game, this comes down to analyzing all aspects of the game, which is a difficult (if not impossible) and very time consuming task. In practice, the plan space will be incomplete. This implies that a move cannot be proved to be the best and as a consequence, some obvious good moves will be overlooked. This danger is much less when a program is data-driven and considers sequences of moves at the basis, possibly by using move generators. Another disadvantage is that there are some types of knowledge which are hard to express in a goal/plan oriented framework, such as patterns. We propose an approach that should overcome these disadvantages. The fundamental difference is that we do not use a decomposition operator, but instead its (more natural) counterpart: the abstraction operator.

4 A hierarchy of boards

Inspired by the way of human reasoning about Go, we adopt a three-layer hierarchy of boards. The boards have *feature vectors* which contain properties of the objects on the board. An *abstraction operator* models the computation of high level board from low level ones. The feature vectors and abstraction operator as we designed them in this section incorporate the most important concepts in Go. However, they are by no means perfect, but that is not the objective at this stage in our research. For understanding the planning algorithm it is enough to know a rough working of the abstraction operator. The modelling of abstract concepts is independent from the planning process and can be improved later on.

Go is a game of gradually surrounding territory and opponent stones. This is reflected by the fact that the *capturing rule* is by far the most important rule of all. It states that a string of stones is captured and removed from the board if all the neighbouring vertices of the string are occupied by enemy stones. From this simple rule, one can deduce shapes that are necessary for stones to be safe from

being captured in the future. In other words, it indirectly defines dead and living groups and territory. We feel that the natural board representation for a game of surrounding is a graph, whose edges represent a neighbourhood relation. Known representations all suffer from a rather naive representation of board positions [2]. The board is commonly represented by a simple one- or two-dimensional array. These representations obviously do not take into account the specific nature of the rules of Go, which refer essentially only to the local neighbourhood structure of the game.

We propose the following three-layer hierarchy of boards, with increasing level of abstraction:

- Full Graph Representation (FGR),
- Tactical Feature Graph (TFG),
- Strategic Feature Graph (SFG).

The abstraction operator will be given the necessary knowledge to compute the TFG and the SFG.

4.1 Full graph representation

The full graph representation (FGR) is a graph with the structure of an $N \times N$ square grid. It is directly motivated by the visual appearance of the classical Go board. The nodes of the FGR represent the intersections of the board and the edges represent the neighbourhood relation. Each node has a label denoting its coordinates on the board. The objects in the FGR are stones, whose only feature is a `colouring`, one of the set $\{black, white\}$.

In our planning methodology, the FGR serves as an exact board representation to play moves on and is input to the abstraction operator, which derives an abstract board from it. However, it also can easily be transformed into a standard array representation and be input to existing move generators. This advantage will be further discussed in section 5.

4.2 Tactical feature graph

The tactical feature graph (TFG) is based on the *common fate graph* of Graepel et alii [2]. They observed that adjacent stones of the same colour, a *string*, will always have a common fate: either all stones remain on the board or all are captured. In any case it is possible to represent them in a single node. This is done by contracting the nodes that the string occupies into one node. This obviously leads to a reduction of complexity in representation while retaining essential structural information. Only the shape of the strings is lost, but this can be found in the FGR.

The objects on the TFG are no longer stones but strings. Common features of a string are the collection of `member_stones`, the number of `liberties` and an `eye_status`. We use a generalization of the number of liberties, namely the

number of `n-th_dame` [6]. This feature contains the number of unoccupied nodes at distance n and is used in the SFG to estimate the strength of groups.

4.3 Strategic feature graph

The basic objects in the strategic feature graph (SFG) are *groups*. A group is standard Go terminology for a number of more or less connected strings and it forms the basis of attack and defense. For the moment, we use the following six basic features of a group: collection of `member_strings`, solidly surrounded `territory`, regions of `influence` (vaguely claimed territory), regions for `future_development`, `weak_points` and `group_strength`. It is the task of the abstraction operator to compute these features. The specific Go knowledge that is required for this task lies outside the scope of this article and is not essential to the understanding of the planning procedure. However, it can be found in all standard Go textbooks, for instance [4]. A useful estimate of `group_strength` is given by the possible omission number [6], the number of moves one can play elsewhere, while being attacked, before one is forced to defend in order to get a living group. The possible omission number depends on the number of `n-th_dame`, a feature of the TFG.

Once it has been computed what strings belong to the same group, they can be represented by a single node. The string nodes and the connecting nodes between them get contracted, just as the stones that belong to one string in the TFG. Likewise, nodes that are all part of the same `territory` or region of `influence` get contracted. The two node contracting steps lead to a further reduction of representational complexity.

5 Pursuing abstract goals

Our planning architecture work in three stages, comparable to human reasoning about Go, as described in section 2. First, the abstract boards are computed. This task will be performed by the abstraction operator (of which the Go-theoretical background is beyond the scope of this article).

The second step is to formulate a goal (or to make a move that was still part of an older plan that has not yet been fully carried out). According to accepted Go theory [4], the choice of goal is largely motivated by the balance of `territory` and `power`. The latter is a combination of `group_strength`, `weak_points` and regions of `influence`. When someone is behind in `territory` but has `strong` groups, his aim could be to deeply invade the opponent's `territory` or to make ambitious extensions to his own `territory`. When a player is ahead with `territory` but low in `power`, he should best defend his `weak_points`. One approach that is suited for this type of reasoning is with a rule-based system. Building such a rule base should be small task compared to building decomposition schemas for the whole game, since it only has to contain relatively few strategic rules. One can extract strategic rules (like the two above) from the well-respected textbook [4], of which most are stated in terms of the SFG features. The important thing is that goal formulation is separated from feature extraction. Thus, the rule base and the

abstraction operator can be improved independently from each other, as long as they keep using the same terms.

The third step is to find a move or sequence of moves that satisfies the chosen goal. In contrast to adversarial planning, this can not be achieved by directly decomposing a goal into lower level ones. That requires a hand coded decomposition schema, which is what we wanted to avoid. We solve this problem with an indirect form of decomposition, which is the main difference between adversarial planning and our method. Indirect decomposition of a goal at a particular level consists of searching the space of the features one level lower, in order to satisfy the goal. During each search step, the abstraction operator computes the features one level higher to perform goal-checking. When the search succeeds, the result will be a goal in the form of values of the features at one level lower. The indirect decomposition will be repeated with these new goal until moves at the lowest level are found. With our three-level hierarchy this means a two-stage searching strategy.

It is, however, not guaranteed that our indirect decomposition method produces goals that can ultimately be refined to moves. It remains to be seen whether desired feature values in the TFG can be indeed caused by a move in the FGR. If this is not the case, it does not mean that the intermediate goal in the TFG is worthless. It can always be used as a good heuristic to guide the search in the FGR towards the goal in the SFG.

The last part of the third step is to take into account the possible actions of the opponent. This is done similar to the adversarial planner GoBI [7], using two opposing agents Alpha and Beta (see section 3). The main difference lies in the way how the backtracking activity is cut off which explores the various interacting plans Alpha and Beta have. In GoBI, a goal can be decomposed as many times as there are decomposition schemas for it. In our method, the indirect decomposition method results in a large number of lower level moves that all fulfil the goal to a certain extent. We restrict the maximum number of indirect decompositions in order to limit the overall branching factor. This number can be used as a control parameter in the planning algorithm: if there is limited time to compute a move, this number will be set low (and vice versa).

An advantage of indirect decomposition is the freedom to use (existing) move generators to improve the search at the level of the FGR. This overcomes the disadvantage of overlooking obvious good moves by (almost unavoidable) incomplete plan knowledge, see section 3. We can use existing pattern-based move generators to produce locally good moves. Using the abstraction operator it is possible to check if this move satisfies the goal. When one uses decomposition schemas, this freedom is cancelled.

We end with a short example of a possible planning episode. Imagine a situation where a `weak` black group without `eyes` is almost surrounded by `strong` white groups, apart from a small gap separating the white groups. In the plan formulation gaining `strength` is chosen as the plan on the SFG level. Now, the planner comes into play. Search in the space of TFG features reveals the following subgoal: an increase of the number of `n-th_dame`. The planner now goes on by searching the space of FGR features (real moves) and considers a move in the gap, which drastically increases the number of `n-th_dame` in the TFG (remember

that n-th dame stands for free nodes at distance n, of which there are many lying behind the gap) and consequently improves the `strength` of the group. Next, counter attacks by white are considered. If they prove not fruitful, the move in the gap is returned by the planner.

6 Conclusions and future work

We modelled human Go playing as a process with three largely independent steps: extracting tactical and strategic features, formulating a goal and finding moves that satisfy the goal. We described a mixed data- and goal-driven planning algorithm for adversaries along these lines. It uses a hierarchy of abstract boards with features, each representing the state of the game at a different level of granularity. The key difference with standard adversarial planning is that we use an abstraction operator instead of its counterpart, a decomposition operator. In our method, decomposition is done indirectly by performing search in the space of feature values on a lower level than the goal. Our method should preserve the main advantage of adversarial planning, a largely reduced branching factor, while retaining the freedom of fitting in data-driven knowledge.

Currently, work is carried out to mathematically formalize and implement the planning algorithm, the abstraction operator and the goal formulation module.

References

[1] K. Erol, D. Nau and J. Hendler. UMCP: A Sound and Complete Planning Procedure for Hierarchical Task-Network Planning. *Proceedings of AIPS-94*, June 1994.

[2] T. Graepel, M. Goutrié, M. Krueger and R. Herbrich. Go, SVM, Go. Berlin University of Technology, 2000. Available at http://www.markus-enzenberger.de/compgo_biblio/compgo_biblio.html.

[3] S. Hu. *Multipurpose Adversary Planning in the Game of Go.* PhD thesis, George Mason University, 1995.

[4] A. Ishida and J. Davies. Attack and Defense. Kiseido Publishing Company, Tokyo, Japan, 1997.

[5] M. Mueller. *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory.* PhD thesis, ETH Zuerich, 1995.

[6] M. Tajima and N. Sanechika. Estimating the Possible Omission Number for Groups in Go by the number of n-th dame. In H.J. van den Herik and H. Iida, editors, *Computers and Games: Proceedings CG'98*, number 1558 in Lecture Notes in Computer Science, pages 265-281. Springer Verlag, Tsukuba, Japan, 1999.

[7] S. Wilmott, J. Richardson, A. Bundy and J. Levine. An adversarial planning approach to Go. In H.J. van den Herik and H. Iida, editors, *Computers and Games: Proceedings CG'98*, number 1558 in Lecture Notes in Computer Sience, pages 93-112. Springer Verlag, Tsukuba, Japan, 1999.