An Adversarial Planning Approach to Go

Steven Willmott[1], Julian Richardson[2], Alan Bundy[2], and John Levine[3]

[1] Laboratoire d'Intelligence Artificielle,
École Polytechnique Fédérale de Lausanne, Lausanne, willmott@lia.di.epfl.ch
[2] Institute of Representation and Reasoning
Division of Informatics
University of Edinburgh, julianr@dai.ed.ac.uk
[3] Artificial Intelligence Applications Institute
Division of Informatics
University of Edinburgh, johnl@aiai.ed.ac.uk

Abstract

Approaches to computer game playing based on (typically $\alpha-\beta$) search of the tree of possible move sequences combined with an evaluation function have been successful for many games, notably Chess. For games with large search spaces and complex positions, such as Go, these approaches are less successful and we are led to seek alternative approaches.
One such alternative is to model the goals of the players, and their strategies for achieving these goals. This approach means searching the space of possible goal expansions, typically much smaller than the space of move sequences.
In this paper we describe how adversarial hierarchical task network planning can provide a framework for goal-directed game playing, and its application to the game of Go.

1 Introduction

Most approaches to computer game playing are based on game tree search and position evaluation functions (data driven approaches). Data driven approaches are appropriate for games with small branching factors, and for which it is possible to accurately assign values to positions which indicate who is winning. While this approach has been very successful for many games including computer Chess, it has been less successful when applied to games with high branching factors and complex positions, such as Go or for games with a high degree of uncertainty such as Bridge.

An alternative to the data driven approach is goal driven search in which a single agent tries to satisfy its goals in the game. Goal driven search has been extensively explored in the AI literature, in particular as Hierarchical Task Network (HTN) planning [23, 8]. When multiple agents need to be modeled and can compete against one another this becomes *adversarial planning*. This paper presents an adversarial planning architecture for performing goal driven reasoning in games and describes its application to the game of Go.

1.1 Goals and Data

Within a specific game, move or action choices often depend upon the state of the game, the phase of the game (e.g. opening, endgame etc.), the future actions of the opponent, the ability of a player to follow up an action appropriately and many other diverse factors. It is these interacting influences on the choice and effect of moves which make games so fascinating for human players and so challenging for machines.

In computer game playing there are two main approaches to making move choices:

- **Data Driven:** At each step, rules, patterns or heuristics are applied to the game state to suggest useful moves. The resulting set of plausible actions is then evaluated using search in the tree of moves. Each move is played out in a world model followed by the possible responses of the opponent. The search continues until the leaves of the tree are reached.[1] These leaf nodes are then evaluated and used to select one of the original plausible actions as the one which leads to the most desirable (by some measure) set of leaf states.

- **Goal Driven:** During play, a goal driven system keeps a number of abstract goals in an agenda. The goals in the agenda represent the things the system would like to achieve in the short, medium and long term. To choose a move, goals are expanded into plans (which are conjunctions of goals at lower levels of abstraction) and eventually into concrete moves. Repeated decompositions form a plan for achieving the goal.

In a data driven search tree, each node represents a possible game position and has one branch for every legal move in that position. By contrast, each node in a goal driven search tree represents a *plan* for achieving the top level goal with some parts still sketchy (abstract goals) and others fixed (concrete actions), and there is one branch at the node for every way the system suggests to further refine the plan.

Which approach (goal driven or data driven) is most advantageous is heavily dependent upon the domain, in particular on the size of the data driven and goal driven search trees. In Bridge, for example, the locations of the cards are not in general known during play, which leads to a large space of possible card plays and therefore a prohibitively large data driven search tree. Frank ([10]) shows that a goal driven approach can very successfully play bridge; a relatively small number of operators is sufficient to describe all the relevant plays.

1.2 The Search Space in the Game of Go

The game of Go is considered by many to be the next great challenge for computational game playing systems. It presents new, significant and different challenges to Chess which has been long been considered the "task par excellence"

[1] Which nodes are the "leaves" can be variously defined by a depth cut off point, quiescence, or further domain dependent heuristics.

for AI (Berliner [2]). Go's search space is both wider and deeper than that of Chess; the size of Go's search space is estimated to be 10^{170} states (cf. Chess $\approx 10^{50}$), games last approximately 300 moves (cf. Chess ≈ 80) and the branching factor at each turn is approximately 235 states (cf. Chess ≈ 35). It is often hard to evaluate the relative strength of Go positions during play. We therefore expect that the brute-force game tree search which has been so effective for Chess will have much greater difficulty with Go.

Due to space limitations, we do not include the rules of Go in this document. A good introduction to the game can be found in, for example, [3].

1.3 Approaches to Computer Go

Although Go has received far less attention than Chess in terms of research, there have been many varied approaches to computer Go. A good summary can be found in [4]. The programs which can play the whole game most successfully, such as Go4++ and Many Faces of Go [9], are primarily data driven but also employ other techniques. Gogol [6] is able to learn patterns for play and various non-symbolic techniques have been used to learn/evolve Go playing controllers (see [26] for more references). Important techniques have also been developed for playing part of the game, in particular focusing on "Life and Death" [27], [28], and using combinatorial game theory to play the endgame [13].

Despite the success of current approaches there is recognition that there is still substantial room for improvement. The advantages mentioned in §6 and earlier work [20, 12] along with the success of this approach in other domains (notably bridge [22]) suggest that a goal driven approach may be useful. It also has much psychological validity, since protocol analysis indicates that Go players consider few candidate moves, and concentrate on their own and on their opponents' purposes [19]. Finally, even in data driven approaches to computer Go, it is still necessary to consider high-level goals, for example in order to decide whether or not a satisfactory result has been achieved in life and death problems (e.g. some strings may be allowed to die but others must live) [28].

1.4 Applications of Adversarial Planning

Early attempts to use goal driven reasoning in adversarial domains include [14] and [5]. The work by Pitrat ([14]) was extended by Wilkins in [24] to produce the successful Paradise system for Chess. More recent work includes [1] (battlefield management) and [30] (command and control). The most recent work is by Smith et. al. on bridge (described in [22] and [21]) which presents a goal driven system for bridge declarer play (Tignum2) good enough to beat the current top commercial computer player.

There have also been several attempts to apply this kind of technique to Go. Early Go Planners due to Sander and Davies [20] and Lehner [12] suffered from the fact that they only work in very open positions and provide high level vague plans. These early systems have difficulties in complex tactical situations, to a large extent due to the difficulty of modeling the interactions between high level goals which must be done for effective HTN planning. One of the crucial differences in the framework presented here is the use of linearisations to produce a total order planner and the use of a world model to track these interactions.

2 An Adversarial Planning Architecture

This section describes an Adversarial Planning Architecture which models goal driven reasoning for adversarial domains.[2] The goal driven approach and use of abstract plans is motivated by work on Hierarchical Task Network (HTN) planning. HTN systems were first used in NOAH [17] and INTERPLAN [23] and have since been extensively studied in the AI planning field. Erol et. al. ([8]) give a complete definition for an HTN scheme and presents UCMP which is a provably sound and complete HTN planner and provides a good template for this type of system.

2.1 Principles of HTN Planning

HTN planning is based on three types of object: *Goals*, *Operators* and *Plan Schemas*. Operators are actions which can be performed in the world (such as flicking a switch, taking a step). Goals are more abstract and express aims in the world such as "Go to the Moon", "Become Prime Minister". Schemas (also called Task Networks), 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 satisfying the conjunction of subgoals G_1, G_2 and G_3:

$$G : - > G_1 + G_2 + G_3$$

The G_i should be at a lower level of abstraction than G, and can generally be satisfied in any order. Operators are at the lowest level of abstraction.

Given these three types, HTN planning starts by taking a starting world state and a set of goals which form the initial abstract plan, which is then refined step by step by expanding the goals within it. Goals are expanded by selecting a schema with the chosen goal as the antecedent (the G) and replacing the instance of G in the current plan by the subgoals (the G_i) listed in the consequent of the schema. As the planning process continues, interactions, incompatibilities and conflicts may arise between combinations of goals. These "interactions" in the plan must be resolved, which can result in backtracking and (in partial order planners) ordering constraints between goals.

The process is complete when all goals have been expanded into sets of operators and all arising interactions have been resolved. The sequence of operators thereby generated should, upon execution in the initial world state, lead to the achievement of the initial goals in the world.

The extension of this idea into adversarial domains is non-trivial since plans are no longer sequences of actions but trees of contingencies which take into

[2] More details can be found in [25].

account the actions of opponents. The interactions in the plan are considerably more complex and serious since the goals of opponents in the world are often conflicting and since the agents are non cooperative. HTN planning for adversarial domains is computationally considerably more complex than HTN planning in standard domains.

2.2 Planning Framework

The adversarial planner presented here models two agents (named Alpha and Beta) which represent two players (*adversaries*) in a game (the framework can be generalised to more than two players). Each agent keeps an open agenda of goals which represents its current plan of action. To solve a problem in a domain, each agent is given an abstract goal (or set of goals) to achieve. The agents then attempt to find sequences of moves which satisfy their goals. Since the goals of the agents are usually contradictory and the agents must take turns in performing actions, their interaction in trying to satisfy their goals can be used to find a plan for the situation.[3]

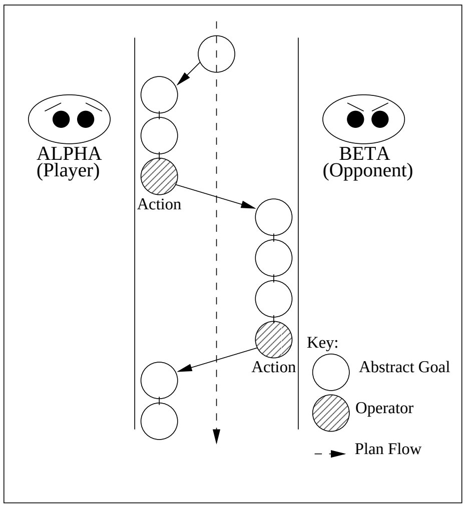

Fig. 1. Planning steps alternating between two agents.

The system allows the two agents to take control of the reasoning apparatus in turns. Once an agent has control it expands some of its abstract goals until it is able to decide upon a concrete action. The chosen action is then performed in a world model[4] before control is passed to the other agent. Figure 1 shows

[3] See below for how this helps choose moves.

[4] A world model is not a standard feature of HTN planners — see §2.3 and [25] for more explanation of its use.

the flow of control during the reasoning process. An agent may need to expand several abstract goals before being able to decide upon an action in the world. During this “active” period it uses its own agenda of goals and has control of the shared reasoning apparatus. Once an action is chosen, control passes to the other agent. Agent Alpha models the player who is *next to move in the game* and agent Beta the opponent, thus the planner is trying to plan for Alpha’s move (Alpha takes control first).

At any one time an agent has a plan which consists of actions already taken (square boxes in figure 2) and goals at various levels of abstraction (circles in figure 2). The actions (squares) are represented in the world model, the abstract goals (circles) are held in the agenda.

A planning step involves selecting an abstract goal (such as X in figure 2) and expanding it. To do this a plan schema is selected for X which expresses how X could be achieved using a conjunction of subgoals at a lower level of abstraction. In figure 2, X is replaced in the plan by the two subgoals X1 and X2. Once expansion has reached the lowest level of abstract goals these lowest level goals need to be shown to be already true or replaced by actions which make them true.

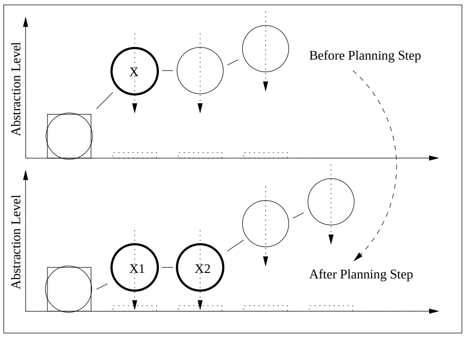

Fig. 2. Plan refinement: abstract goals are expanded to and replaced by sets of subgoals at lower levels of abstraction.

Once one of the agents (Alpha say) has achieved all of its goals (been able to perform actions in the world model which make them true) it knows that it must have satisfied its top level goals (since all its subgoals are strictly descended from these). The other agent is made aware of this and, since in general a good outcome for one agent is a bad outcome for the other, both agents are allowed to force backtracking. Agents are allowed to backtrack to any of their previous goal or expansion choices but only to their *own* decisions. Neither agent may force the other to change plans directly.

The backtracking activity explores the various interacting plans Alpha and Beta have for the situation and creates a plan tree as shown on the left of figure 3. Each choice made by an agent creates a new branch. Underlying the

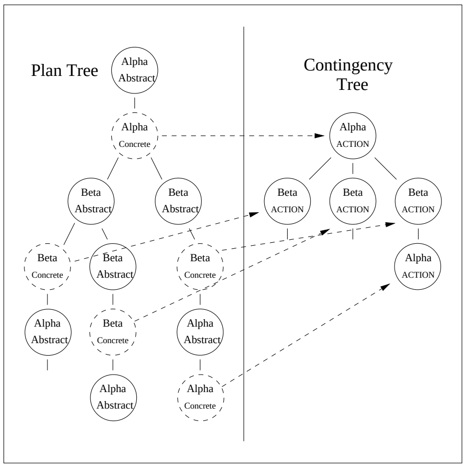

Fig. 3. The plan tree on the left is reduced to the contingency tree on the right by dropping the abstract reasoning nodes.

plan tree is the *contingency tree* which is found by removing all the abstract goal decomposition steps in the plan tree to leave only the operators/actions (shown on the right in figure 3). The final contingency tree acts as a form of proof that the first move is a good step towards achieving Alpha's goals. Hence it supports the choice of the first move in the tree.[5] In general, the final contingency tree contains only a small subset of the move tree which would be generated by considering all the available options at each turn as in a data driven approach. (See §6 below.)

2.3 Discussion

Moves in the contingency tree are *directly descended* from the goals of the two agents, and the tree structure naturally reflects the interactions between the two adversaries. Taking any branch, the moves chosen near the leaf (at the end of a move sequence) are still directly related to the same plan and aim that suggested those moves near the root (the first moves in the tree).

A key difference from standard HTN planning is that goals are expanded in time order using a linearisation. Such a linearisation turns the planner from a *partial order* into a *total order* planner and allows the use of a world model. Figure 2 illustrates how the least abstract goals/actions appear first in the sequence. As soon as goals reach the lowest level of abstraction (become operators) they can be added to the world model. This is a significant difference from standard HTN planners, its importance is discussed further in [25] and [26].

[5] The tree itself can also be used to respond to any of the opponents moves which are represented in it, but re-planning may be required if other moves are made. The question of how long (for how many moves) such a plan remains valid is not addressed here.

3 An Adversarial Go Planner

The planning architecture was instantiated as a Go reasoning system called GoBI both to test the architecture in a challenging domain and to investigate the usefulness of the goal driven approach for Go. GoBI consists of a set of knowledge modules which plug into the planning architecture. The knowledge modules provide the Go domain knowledge, plan schemas and goal types which the reasoner can use to solve problems.

3.1 An Example Go Plan

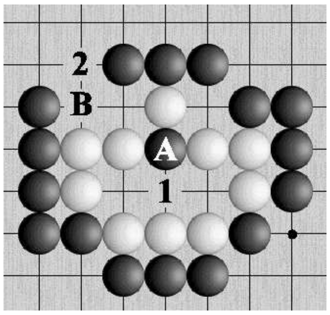

Fig. 4. Example 1: Black to play and kill the white group.

Figure 4 shows a problem from Volume I of "Graded Go Problems for Beginners" (Yoshinori [29]). The aim is for black to move first and kill the white group of stones. The task is specified to GoBI as two abstract goals: the goal *kill-group* for agent Alpha (playing black) and the goal *save-group* for agent Beta (playing white).[6] Agent Alpha takes control, first decomposing the *kill-group* goal using one of the available plan schemas. An abstract plan for killing this group might be a conjunction of the following subgoals:[7]

- *surround-group* — stop the group from running and connecting.
- *squeeze-space* — reduce the space the group has to live.
- *prevent-eye-formation* — block any attempt by the group to make eyes.[8]

One of these subgoals is then expanded further to the next level and so on until at the lowest level in the hierarchy a move such as *play at B* is chosen to satisfy a simple goal such as *prevent-escape-at-2* (figure 4).

[6] Note: the goals need not be directly opposing.

[7] This abstract plan is quite intuitive. It is not obvious how a data driven system would represent the equivalent of such a plan.

[8] An eye in Go is an enclosed space where the opponent may not play - a group with two eyes is unconditionally alive.

Alpha plays this move onto the board in the world model which gives the new world state for Beta to work with. Alpha still has a set of goals at various levels of abstraction remaining in its agenda. These remaining goals represent the plan on how to follow the first move, i.e which other subgoals/actions need to be achieved to make the plan complete. To validate that this first move is good (in this case playing at B would not be), Alpha must eventually show that all these goals can be achieved no matter what Beta does. These remaining goals are kept by Alpha until after Beta's turn.

Beta now begins by expanding its single goal *save-group* in the context of the new board position (after Alpha playing at B in Figure 4). A possible plan for this goal is:

- *make-eye-space*.
- *make-eyes* (try to form two eyes).

After Beta's move (whatever it is) is played into the world model, control is returned to Alpha which then tries to satisfy the rest of its goals. The interleaving of goal expansions by the two agents continues until one is able to satisfy all of its aims (and thus implicitly its main aim). The opposing agent is informed of this and it then backtracks to explore any alternative options it has which might produce a better outcome for itself. In this way a contingency tree is generated which either proves or refutes the validity of the first move (Alpha's).

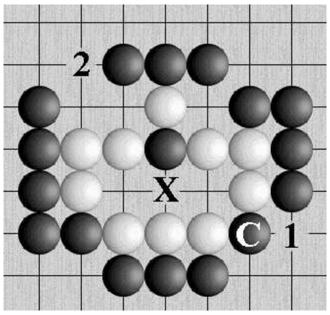

Fig. 5. GoBI plays at X and this kills the group.

For this example (figure 4) GoBI returns the move at X in figure 5, which kills the group. Among the defences tried by Beta are trying to run out at 2 and counter-attacking by playing at 1 (this puts the single black stone C under threat). Since all the moves tried by both agents must be *part of plan of action*, the number of possible moves searched is very small compared to the number of available moves in even this small problem.

3.2 Representing Go Knowledge

The planning architecture and Go knowledge modules which make up GoBI are all written in Common Lisp. Around 1400 lines of code make up the plan knowledge (i.e. schemas and goal types) GOBI has. Writing a full-board Go-playing program is a significant exercise in knowledge engineering, so to enable us to add enough depth in knowledge to do useful testing in a short time, GOBI's knowledge is focused on the area of killing and saving groups.[9] The knowledge base is made up of 45 goals at five different levels of abstraction. The average number of possible plans per goal is approximately two (thus the knowledge has a relatively low branching factor). The two highest level goals available in GOBI are *kill-group* and *save-group*, which emphasises the focus on life and death problems.

The following example goal taken from GOBI's knowledge base illustrates the structure of the knowledge the program holds (the *save-group* goal was also mentioned in the previous example). Note that the plan knowledge is not complete (for example making eyes is not the only way of following up a counter attack); more work is needed to extend the knowledge base.

```
GOAL: save-group,
LEVEL = 5,

 Plan 1 - Find Eyes:
  *make-eye-space,
  *make-eyes.

 Plan 2 - Escape Group:
  *running-sequence,
  *secure-escape.

 Plan 3 - Counter Attack String:
  *locate-vulnerable-string,
  *kill-string,
  *make-eyes.
```

In turn the *make-eye-space* goal from `Plan1` has two alternative plans:

```
GOAL: make-eye-space,
LEVEL = 4,

 Plan 1 - Ambitious Extend:
  *large-extending move,
  *consolidate-space.

 Plan 2 - Creep extension:
  *creeping-extending-move,   //A single
  *consolidate-space.         //step extension.
```

The structure of the knowledge shown here is intuitive for Go and is very different from the kind of heuristic information added into data driven approaches. The knowledge represented can be seen as an AND-OR tree with the AND component

[9] Note however that this does not mean GOBI is limited to enclosed problems (see §4.2).

represented in the conjunction within the plan schema and the OR component in the choice between alternative plans. Goals within plans are not evaluated at every step, for simple goals truth is established using a simple test or if an appropriate move can be made. For higher level goals truth is inferred from the fact that the goals descended from them were achieved.

3.3 The Planning Process

Currently GoBI has no goals and plans which persist *between* game moves and top level goals are given to the planner for each position to solve. In order to extend GoBI to whole-board play, one of the most important tasks is to define goals and abstract plans above the current highest level. The planner may keep these goals open (and still unexpanded) from one turn to another and indeed for long periods of the game. Earlier work [20, 12] has already shown the usefulness of planning at the higher strategic level, GoBI aims to be useful and applicable at all levels.

As described in §2 during the reasoning process a planning step involves two steps: 1) Choosing a current goal to replace in the plan and 2) Choosing an appropriate plan to replace it with. In GoBI the choice of which goal to expand next is unordered, however after choosing a goal from the agenda GoBI will try to work on it down through several levels of expansion. Repeatedly choosing one of it's descendents for expansion, this leads to some parts of the plan being expanded quickly (so they can be tracked in the world model) while others remain abstract for longer. Once a goal has been chosen, one of its associated set of plans must be selected for use. The plan schemas in GoBI are tried in a fixed order designed to try the most promising goal decompositions first. Plan preconditions can also be used to screen out plans not thought to be suitable for the current situation. If one plan leads to a dead-end, backtracking allows GoBI to try the others available for the situation.

The plans stored in the Go modules are not pre-programmed solutions and are expanded *in the context of the current game state*. The development of a plan is influenced by schema preconditions and by the choices in the world for making the lower level goals true. The failure of goals early in the plan quickly forces choice of alternative subplans for making these goals true.

The expansion of goals into plans eventually leads to goals at the lowest level of abstraction. These need to be checked for satisfaction and used to choose moves. Some example lowest level goals are: *fill-a-liberty*, *play-a-hane-move* (near *here*), *play-a-connecting-move* (between *string1* and *string2*), *play-a-placement-move*, *play-a-blocking-move* (near *here*), *play-an-extending-move* (near *here*).

The cost of move choice can be divided into three components:

1. Abstract Planning steps: these are very cheap since they consist purely of two cycles of matching items in a list (choose a goal, choose one of its plans). They can be more expensive if complex pre-conditions are used to select between plans.

2. Checking the satisfaction of low level goals: Since the goals are very focused (are these two strings connected? could black run out here?) this is also not too expensive. Checking is made easier by the fact that it can be supported by various representations of the game state — strings, groups, influence etc.

3. Using low level goals to generate moves: This is the most expensive part of the process although again the goals by this stage are very focused and limited to a small area (the *here* above). In GoBI selection is done using simple rules which define the type of move which could satisfy a goal.[10] An example set of rules is that a connecting move must:
   (a) be a liberty of (next to) *string1* and
   (b) be a liberty of (next to) *string2*.

Although the workload is divided between these three parts it is clear that this move choice method is in general more expensive for choosing a single more to try in the move tree than most data driven methods (see §6.2 for some estimates). The potential gain of using this method is in reducing the number of candidate moves in the move tree which needs to be tried. There are two important tradeoffs in the planning process:

- If goal selection and expansion mechanisms get more complex (i.e. through the extensive use of complex preconditions) their time cost will increase, however it should also mean a better use of the knowledge available.

- The simpler and more specific the lowest level goals are, the easier it is to establish their truth and choose actions with them but the more planning is needed to decide upon them.

3.4 Mixing Goal and Data Driven Approaches

Although the work here focuses on goal driven approaches to Go, it is clear that human players mix both types of reasoning. Patterns are thought to play a large part in human Go play. Thus, there is a need to consider how the two approaches can be mixed successfully.

One of the advantages of using a world model in the planner is that the changing situation of the state during planning is reflected in the world model. The changing world state may highlight interesting opportunities (or obvious problems) which arise as a result of the plan actions but were not expected effects. The architecture described above was extended to include plan *critics* which have access to the world model and watch for important situations. The critics are able to insert goals into the agendas of the agents for the planner to consider *alongside* the current track of reasoning. Thus the planner is made aware of opportunities, for example, during planning and can react to them.

[10] Note this could just as well have been done with local pattern matching. The planning framework poses no restriction on how this relationship between abstract goals and concrete actions is established (in general this is domain dependent).

Such opportunistic reasoning seems a plausible mechanism in terms of human play. Plans suggest moves which might lead to unexpected opportunities which in turn might lead to new plans etc.

Two critics were added to GoBI which detect groups under immediate threat of capture and insert (optional) group-saving/group-attacking goals into the agents' agendas.

4 Testing and Evaluating GoBI

Yoshinori's four-volume series [29] provides an excellent source of problems for testing Go programs. Since setting up each test problem was time-consuming, we chose (test set I) a representative sample (85 problems, approximately one third) of the problems from volume I of [29]. Problems were chosen for the test set essentially randomly, with a bias towards harder problems and excluding the simple capturing problems at the beginning of the book (which were considered too easy). A second set of tests using problems from volume II [29] and [7] was also conducted. All the tests were limited to 30 seconds of runtime on a SUN UltraSparc.

4.1 Test Results

The system successfully solved 74%[11] of the examples in test set I, which is a significant achievement given its limited knowledge and the complexity of some of the problems. GoBI has also solved considerably harder problems from volume II [29] and [7]. These tests were not comprehensive however, and used a range of hand picked examples, so the most useful indicator of performance remains the performance on problems from volume I [29].

In 98% of the correct solutions, GoBI considered most of the significant defences or responses of the opponent in its plan. This statistic is encouraging since not only were the answers correct — so was the reasoning behind them. Most of the failures were due to incomplete plan knowledge, several problems relied on groups connecting and escaping to live for example which is something GoBI currently has no plans for (GoBI has no plans for cutting groups). Another weak area was in problems which required a lot of forcing moves (e.g. ladders). GoBI has no special way of handling these and so is not able to take advantage of the problem simplification they provide (they are planned for in the normal way).

Curiously strengthening GoBI's defensive knowledge led to an improvement in attacking plans and vice versa, reflecting the fact that the better opponent model is more likely to find refutations for poor attacking plans.

4.2 Discussion

Further useful tests would involve comparing solutions with those of current programs such as GoTools and Many Faces of Go, however we currently

[11] 67% when critics were disabled.

have access to neither of these. A second difficulty is that in fact comparisons would need to be made as to the number and relative cost of steps taken (search or reasoning) since our current system is a prototype and the other two systems are commercial with several years of development.

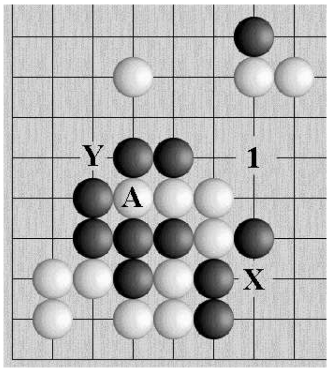

Fig. 6. Example 3: GOBI solves this more open position, killing the white string marked A by playing a net move at 1. Adding white stones at X or Y for example (or various other places) would cause GOBI to realise that the string can no longer be killed directly.

Even though most of GOBI's knowledge is about killing and saving groups it is more general than specialist life-and-death programs, knowledge for other parts of the game can be freely added and GOBI is also able to handle more open positions (which for example GOTOOLS has trouble with [28] such as the one shown in figure 6). In fact one would expect GOBI to have more of an advantage in open (though still tactical) positions where the number of possible move sequences increases rapidly. The plan knowledge in the system would then focus on only the relevant parts of the move tree.

5 Relationship to Other HTN Systems and Adversarial Planners in Go

Since what was probably the first use of the notion of goal directed search by Reitman and Wilcox [15] in 1979 there have been several attempts at applying planning approaches to Go.

Work by Ricaud [16] has similarities to that presented here although it focuses more on the use of abstraction. The GOBELIN system uses an abstract representation of the Go game state to produce plans which are then validated in the ground representation. GOBI is more flexible than this in that it shifts between several levels of abstraction (five with current knowledge) dynamically during planning rather than having a generation/validation cycle. GOBELIN probably also suffers in a similar way to [20] and [12] in that it has difficulty in representing complex interactions in the abstract representation. This difficulty may explain why GOBELIN makes the strongest plans during the opening where play is very open and there are few complex tactical situations. In his PhD thesis Hu [11] again focuses mainly on the strategic level of Go play, addressing issues of combining individual strategic and tactical goals into goals with multiple purposes.

The use of a world model to track the interactions between goals and making linearisation commitments during planning seems essential to any progress in domains with high levels of interactions between player goals. The utility of a world model has also been noted in other domains — notably Bridge [22, 21] — and makes GOBI much more applicable to tight tactical situations which can arise in Go (which was the major stumbling block in previous work [20, 12]). GOBI also uses an intuitive planning structure with the planning engine clearly separated from the knowledge.

The architecture is similar to the Wilkins' PARADISE Chess system [24] in the way that it uses plans to guide move choice. The use of the world model allows a similar access to the traversal of the search tree used in PARADISE. A significant difference, however, is that GOBI's plan schemas are used to both generate and justify plans of action, whereas PARADISE uses a complex mix of static evaluation, production rules, and limited move tree search.

6 Advantages of the Goal Driven Approach

The goal driven approach which is presented here has some clear advantages for Go and other similar games. Together with some of the previous work on the usefulness of planning at the strategic level of Go, GOBI shows that this approach can be used for reasoning at all levels, moving transparently between them and providing a unifying framework for Go play.

6.1 Representation and Communication of Domain Knowledge

Go knowledge in books and folklore is often expressed in a form appropriate for encoding as decompositions of abstract goals into other goals at different levels of abstraction. As reported in [18], there is a rich vocabulary which Go players can use to express their reasons for making a move. There are many popular proverbs, which convey strategies at various levels of abstraction, for example "death lies in the hane" is more tactical, whereas "don't push along the fifth line" is quite strategic. It may be easier to add this kind of knowledge to a goal driven system than to a data driven system which requires the delicate balancing of heuristics.

By following the trace of the goal decompositions one can see *why* the Go player is trying to do something — its aims and plans. Access to aims and plans is not only helpful for adding knowledge and debugging, but could be useful in the context of a Go tutoring aid. Some of the current commercial Go systems (MANY FACES OF GO for example) have teaching mechanisms but it is not clear if these are based directly on the reasoning process of the computer player.

6.2 Search Properties

Using a goal driven approach leads to a very different search from a data driven approach, some key advantages are:

- There is no longer a need for global evaluation functions: this reduces to checking if goals can be satisfied. Evaluation functions may still be used in a limited way to carry out judgements which cannot be made on a move-by-move level, for example using an influence function to judge whether or not a group can successfully run, or using a fast $\alpha-\beta$ search to determine the life or death of enclosed groups. When an evaluation function is employed in determining whether or not a goal has been satisfied, the goal can be used to provide a focus for the evaluation function to a limited area of the board. The use of goals to focus evaluation makes sense when thinking about how humans play go — often making instinctive judgements (evaluations) of a local situation but rarely considering (evaluating) the whole board.

- Quiescence is defined automatically, thus avoiding problems such as the horizon effect in search. Since each agent has an agenda of goals, a situation can be defined as "open" (unsettled) until all goals are achieved.

- What seem like heuristically bad moves (e.g. sacrifices) are not discriminated against because the only consideration is their value to the plan.

Whilst these are important advantages a further strong point often associated with goal driven approaches is their resilience in domains with large search spaces and branching factors. Move choices are driven from above and not by the options available at each choice point. The plan knowledge therefore focuses which of the choices are directly *relevant to the current plan.* The problems GoBi was tested on gave a reduction in the number of move tree nodes visited of 10 - 100 times compared to standard $\alpha-\beta$ search. The current top Go programs are able to use a great deal of pattern and heuristic knowledge to improve on α–β, however these gains are still significant.

The number of nodes visited in the move tree does not tell the whole story since GoBi requires more reasoning to choose which nodes to visit. This reasoning consists of the three components identified in §3.3 (expansion of a goal to a further abstract plan, checking if a low level goal is satisfied, choosing a move given a low level goal). For the examples in test set I (see §4) there were on average 4 planning steps per move chosen (including the steps which choose moves). Initial analysis suggests that given GoBi's current knowledge, each move tried in the game tree involves an average overhead of 2.5μ in reasoning, where μ is the average cost of modifying the internal data structures to make a move (including checking for Ko's, removing any stones killed, checking legality etc). Thus the cost of choosing a move in GoBi is about 3.5 times as much as in α–β (since α–β's main cost is adding stones to the board - i.e. μ). The computational advantages of GoBi over α–β are still clear-cut since the number of move choices (tried in the move tree) is significantly less than for $\alpha-\beta$. Note also that any system which begins to use heuristics for move choice also begins to increase the cost of each individual move choice. A more direct comparison with the systems able to apply large amounts of heuristic information to improve upon $\alpha-\beta$ would be useful but we do not have access to the relevant systems. The figures above at least illustrate how the plan knowledge in GOBI is able to cut down the search space by reasoning from above.

7 Disadvantages of Goal Driven Search

Obviously the goal driven approach is not always the best choice and has its own difficulties:

- The goal driven approach requires significant effort to encode strategies as goal decompositions. By contrast, in the data driven approach, good play can be achieved using even relatively simple evaluation functions as long as the game tree can be searched deeply enough.

- There are some types of knowledge which are hard to express in a goal/plan oriented framework, such as knowledge which is not reliant on understanding the motivation behind a move (patterns for example). It seems clear that good Go play requires both pattern (data driven) and abstract plan (goal driven) knowledge which is what leads us to try and integrate the two approaches (see §3.4).

- For games with low branching factors, shallow search trees or where near exhaustive search is possible data driven approaches have a strong advantage. It is only when searching most of the move tree is infeasible and large amounts of knowledge are needed to prune the tree that goal driven approaches can gain the upper hand.[12]

As with all knowledge based approaches, the search space is determined by the knowledge in the system, so large amounts of knowledge can dramatically impact performance. The key to making knowledge based approaches work is good organisation of that knowledge. Having many overlapping plans for example would cause serious performance problems, since each would be evaluated in turn.[13] In general there is little problem in adding "breadth" of knowledge (e.g. plans for different parts of the game) since knowledge is often obviously not applicable or leads to failure quickly. Adding more detailed plans ("depth"

[12] This point was illustrated by PARADISE [24] in the early eighties which despite being able to solve Chess problems requiring search up to 20 ply deep (far beyond other Chess programs of the time), still saw its knowledge based approach outstripped by the ever increasing efficiency of fast search techniques.

[13] Some moves may be chosen twice for different reasons (in different plans) which is perfectly acceptable, however if many plans overlap long sequences of move choices could be the same.

in knowledge) needs to be done carefully however since this can lead to overlaps and redundant search. The levels of abstraction used in GOBI (and in many AI planners) are key in avoiding this since knowledge can be appropriately structured and similar plans can be grouped together and expressed in a compact form.

The main problem with GOBI itself is still lack of knowledge: much more is needed before GOBI could be used as a tool to play the complete game of Go. Unfortunately adding knowledge takes a long time since it must be hand coded, most of the top programs have had knowledge added over periods of years. In this respect this approach (and many current approaches) are at a disadvantage to learning or non-symbolic approaches which can use automatic training to improve.

8 Conclusions

In this paper we have presented an adversarial planning architecture capable of reasoning about games, and an application of this architecture to Go. The planning architecture and Go reasoner reported here represent an advance on previous work for goal driven planning in Go. The system

- has a clear separation of domain knowledge from the abstract planning architecture and a clear model of the opponent in the game,
- can reason at multiple levels of abstraction simultaneously,
- can address complex tactical situations as well as high level strategic problems,
- can provide support for the integration of data driven and goal driven approaches.

We presented the advantages that a goal driven approach could have for Go and discussed the importance of mixing data and goal driven aspects when working in complex domains such as Go. GOBI as a prototype is certainly no match for current systems which play the full game of Go (and indeed cannot yet play the full game), however it does represent a step towards understanding how goal driven approaches could be applied to Go, even at a tactical level. GOBI needs more knowledge adding to be applicable in other areas of the game. A further important step is to incorporate higher level, persistent goals which allow GOBI to play the whole game of Go. We also aim to further develop structures for mixing data and goal driven approaches.

Go has several strong domain features which make goal driven approaches applicable (very large search spaces, clear layers of abstraction in domain descriptions — stones, strings, groups etc, and a wealth of knowledge similar in structure to abstract plans) GOBI represents a further step towards turning this theoretical possibility into a reality. The work described in this paper again shows that Go is an excellent test bed for AI research. There has been very little work on adversarial planning in recent years — the challenge of Go really motivated this work.

9 Acknowledgements

The first author is very grateful for financial support from EPSRC under grant number 96417312. The second and third authors are supported by EPSRC grant GR/L/11724. The fourth author is partly funded by DARPA/AFRL contract F30602-95-1-0022. Special thanks also go to Martin Müller for suggestions regarding test data and to Neil Firth for numerous instructive games of Go. We are grateful to the anonymous referees whose comments have resulted in numerous improvements to this paper.

References

1. C. Applegate, C. Elsaesser, and D. Sanborn. An Architecture for Adversarial Planning. *IEEE Transactions on Systems, Man and Cybernetics*, 20(1):186–294, 1990.
2. H. J. Berliner. A Chronology of Computer Chess and its Literature . *Artificial Intelligence*, 10:201–214, 1978.
3. R. Bozulich. *Second Book of Go*. The Ishi Press Inc, 1987.
4. J. Burmeister and J. Wiles. An Introduction to the Computer Go field and Associated Internet Resources. Technical report, The University of Queensland, January 1997. Available online at: http://www/psy.uq.edu.au/~jay/go/go_page.html.
5. J. G. Carbonell. Counterplanning: A Strategy Based Model of Adversarial Planning in Real World Situations. *Artificial Intelligence*, 16(1):295–329, 1981.
6. T. Cazenave. *Système d'Apprentisage par Auto-Observation. Application au Jeu de Go*. PhD thesis, L'Université Paris 6, 1996.
7. J. Davies. *Life and Death*. The Ishi Press Inc, 1978.
8. K. Erol, D. Nau, and J. Hendler. UMCP: A Sound and Complete Planning Procedure for Hierarchical Task-Network Planning. *AIPS-94*, June 1994.
9. D. Fotland. Knowledge Representation in The Many Faces of Go. Technical report, American Go Association, 1993. Available online at: ftp://bsdserver.ucsf.edu/Go/comp/mfg.Z .
10. I. Frank. *Search and Planning under Incomplete Information : a Study using Bridge Card Play*. PhD thesis, Department of Artificial Intelligence - University of Edinburgh., 1996.
11. S. Hu. *Multipurpose Adversary Planning in The Game of Go*. PhD thesis, George Mason University, 1995.
12. P. Lehner. Strategic Planning in Go. In M. A. Bramer, editor, *Computer Game Playing: Theory and Practice*, pages 167–176. Ellis Horwood, 1983.
13. M. Müller. *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory*. PhD thesis, Swiss Federal Institute of Technology, Zurich, 1995.
14. J. Pitrat. A Chess Program Which Uses Plans. *Artificial Intelligence*, 8(1):275–321, 1977.

15. W. Reitman and B. Wilcox. The Structure and Performance of the Intrim 2.0 Program. In *International Joint Conference on Artificial Intelligence*, pages 711–719, 1979.
16. P. Ricaud. A Model of Strategy for the Game of Go Using Abstraction Mechanisms. *International Joint Conference on Artificial Intelligence*, pages 678–683, 1997.
17. E. D. Sacerdoti. *A Structure for Plans and Behaviour.* Elsevier, 1977.
18. Y. Saito and A. Yoshikawa. Do Go Players Think in Words? — Interim Report of the Analysis of Go Player's Protocols. In Hitoshi Matsubara, editor, *Proceedings of the Second Game Programming Workshop.* Computer Shogi Association, 1995.
19. Yasuki Saito and Atsushi Yoshikawa. An Analysis of Strong Go-players' Protocols. In Hitoshi Matsubara, editor, *Proceedings of the Third Game Programming Workshop.* Computer Shogi Association, 1996.
20. P. T. Sander and D. J. M. Davies. A Strategic Approach to the Game of Go. In M. A. Bramer, editor, *Computer Game Playing: Theory and Practice*, pages 152–166. Ellis Horwood, 1983.
21. S. J. J. Smith, D. S. Nau, and T. A. Throop. A Planning Approach to Declarer Play in Contract Bridge. *Computational Intelligence*, 12(1), 1996.
22. S. J. J. Smith, D. S. Nau, and T. A. Throop. Total-Order Multi-Agent Task-Network Planning for Contract Bridge. *Proceedings AAAI-96*, pages 108–113, 1996.
23. A. Tate. Generating Project Networks. 1977.
24. D. E. Wilkins. Using Patterns and Plans in Chess. *Artificial Intelligence*, 14(1):165–203, 1980.
25. S. Willmott. Adversarial Planning and the Game of Go. Master's thesis, Department of Artificial Intelligence, University of Edinburgh, September 1997.
26. S. Willmott, J. D. C. Richardson, A. Bundy, and J. M. Levine. An Adversarial Planning Approach to Go (long version). Technical report, Department of Artificial Intelligence, University of Edinburgh, 1998. Forthcoming, Serial number TBA.
27. T. Wolf. The Program GoTools and its Computer-generated Tsume Go Database. In *Proceedings of the First Game Programming Workshop*, pages 84–96. Computer Shogi Association, 1994.
28. T. Wolf. About Problems in Generalizing a Tsumego Program to Open Positions. In Hitoshi Matsubara, editor, *Proceedings of the Third Game Programming Workshop.* Computer Shogi Association, 1996.
29. K. Yoshinori. *Graded Go Problems for Beginners (Volumes I - IV).* The Ishi Press Inc, 1985.
30. P. R. Young and P. Lehner. Applications of a Theory of Automated Adversarial Planning to Command and Control. *IEEE Transactions on Systems, Man and Cybernetics*, 16(6):186–294, 1990.