Searching for Compound Goals Using Relevancy Zones in the Game of Go

Jan Ramon and Tom Croonenborghs

Department of Computer Science,
Katholieke Universiteit Leuven, Leuven, Belgium
{Jan.Ramon, Tom.Croonenborghs}@cs.kuleuven.ac.be

Abstract. In complex games with a high branching factor, global alpha-beta search is computationally infeasible. One way to overcome this problem is by using selective goal-directed search algorithms. These goal-directed searches can use relevancy zones to determine which part of the board influences the goal. In this paper, we propose a general method that uses these relevancy zones for searching for compound goals. A compound goal is constructed from less complex atomic goals, using the standard connectives. In contrast to other approaches that treat goals separately in the search phase, compound goal search obtains exact results.

1 Introduction

In complex games with a high branching factor, such as the Asian game of Go, global alpha-beta search is computationally infeasible. Therefore, there has recently been some research towards local searches.

Local searches can be obtained by splitting the global search into several independent local searches. This approach is successfully adopted by Müller using decomposition search in the endgame [5]. Unfortunately, in the middle game it is rare to find isolated regions which are not influenced by other regions. Therefore, other notions of locality have been proposed which are more suitable for the middle game.

One way to preserve locality is by using selective goal-directed search algorithms [1,3,6]. This approach has been successfully adopted to several sub-games of Go, such as capturing a string of stones, connecting two strings of stones, and making life for a group. These goal-directed searches can use relevancy zones to determine which part of the board influences the goal. However, searches for such elementary goals cannot be used for solving more complex problems or for evaluating game positions. For instance, in Figure 1, there are two white stones with only one liberty. If Black is to move, he[1] can capture one of them, but then the other stone can escape. The question then arises how he should play in order to capture both stones. As we will see, Black can capture both stones by playing a ladder break on the crossing of both ladders. Cazenave and Helmstetter

[1] In this article, when 'he' and 'she' are both possible, 'he' is used.

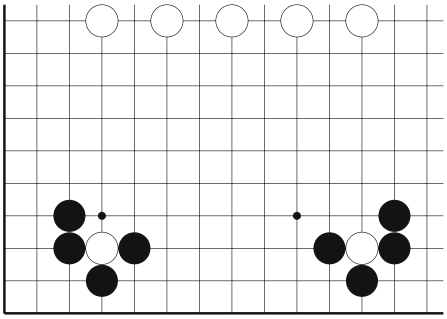


Fig. 1. A double ladder

[4] made a start on combining several goals into one search by considering the proof of the connection of two goals A and C using the proofs of the connections between A and B and between B and C.

In this paper, we propose a general method that searches with compound goals. A compound goal is a more complex goal, that is built from simpler atomic goals. Searching using compound goals is especially interesting when the atomic goals used are more or less independent, but not completely. In contrast to other approaches that treat local goals separately in the search phase (such as planning approaches [7]), compound goal search obtains exact results. The technique presented here is a generalization of [4] on transitive connections, as we treat compound goals which are arbitrary combinations (using conjunction, disjunction, and negation) of arbitrary atomic goals (not only connections but also, e.g., capturing and living).

The remainder of this paper is organized as follows. Section 2 reviews selective goal-directed search algorithms and introduces the terminology used in this paper. Next, in Section 3 we explain how we construct compound goals, using atomic subgoals. In Section 4 our experiments and analyses can be found. Finally, in Section 5 we present our conclusions and ideas for further work.

2 Searching for Atomic Goals

In this section the basics of selective goal-directed search algorithms are reviewed and the used terminology is introduced.

In games such as Go, locality is important, as the board is large and at the same time intermediate goals can be formulated using information of only a small local area. Special algorithms have been developed recently. The important idea in these algorithms is that one can be very selective on the candidate moves by only considering those that could potentially prevent a negative result that follows when no move (a nullmove) is played. To human Go players too, it is common knowledge that "my opponent's point is my point".

A goal is formulated for a particular player. He is called the attacker or MAX. The opponent is called the defender or MIN (he should try to prevent MAX from reaching the goal). For a particular goal G we will denote MAX by MAX(G) and MIN by MIN(G). Some common examples of goals in increasing order of complexity include the `capture` goal (MAX should capture a particular block of stones), the `connection` goal (MAX should connect two of his blocks), the `life` goal (MAX should make life for a given set of blocks) and the `invade` goal (MAX should make a living group somewhere in a given area).

The order of a move is an important concept. Generally speaking it indicates how directly the move can realize the goal. A MAX move of order n indicates that the goal can be reached, if it is followed by n successive moves of MAX[2]. By using this notion of the order of a move, the search can be kept local by limiting the order of the moves to be searched. A MAX move of order 0 reaches the goal directly. A move of a higher order n is a move such that if MIN passes, MAX can play a move of order at most $n - 1$. For the `capture` goal, ladders are moves of order 1 and a geta (net) has order at least 2.

Generally speaking, a relevancy zone is a set of intersections that supports a proof. When a search engine needs to prove a tactical goal, it has to remember all the reasons that are responsible for the result of that search. This collection of intersections is called the relevancy zone or trace of that search.

To illustrate the use of relevancy zones, the double ladder example from the previous section is repeated. Figure 2 shows the same position as Figure 1, but this time the relevancy zones are included. A triangle indicates an intersection that belongs to the relevancy zone, obtained by the local search which goal is to capture the left white stone. The squares indicate the intersections from the relevancy zone of the local search for the right white stone. The intersection of both relevancy zones are the points marked by a circle.

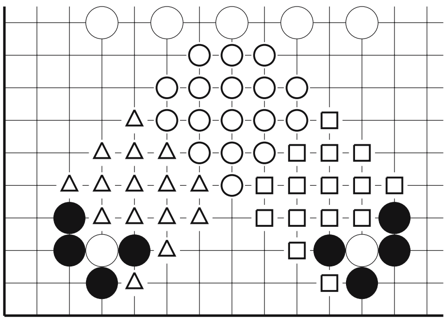


Fig. 2. A double ladder, including the relevancy zones

Relevancy zones have several uses. We mention two of them. First, as a proof remains valid as long as the intersections of its relevancy zone remain unchanged, it is natural for a playing program to store the relevancy zone of a proof together with the result, so it knows when it has to recompute the status of the goal. Second, if it is proved that for a certain order no move exists that reached the

[2] Definitions differ somewhat among the different articles in literature.

goal, one can use the relevancy zone of that proof as the set of candidate moves in a new search for moves of a higher order.

Different relevancy zones can be obtained by proving the same tactical goal, e.g., when proving that a block cannot be captured in one move, one only needs to add two liberties of that block (arbitrarily) to the relevancy zone. Ideally and not considering memory constraints, one could use a tree-like representation for a relevancy zone that entails the minimal relevancy zone, obtained by browsing through the tree. Currently, only one of the proofs is used as the relevancy zone for that search. This still obtains correct results, but it can be less efficient. (It can be necessary to recalculate the search, just because the "wrong" liberty is added.)

We distinguish between two subsets of a relevancy zone: the positive relevancy zone (denoted `Rzone`$^+$), containing all points such that when they remain unchanged, the result for the associated search cannot become better than the previously computed result for MAX and the negative relevancy zone (denoted `Rzone`$^-$) of all points such that when they remain unchanged the obtained result can not get worse for MAX. To illustrate this distinction, consider the position in Figure 3, where the goal is to capture the white group, so the black player is MAX and White is MIN. The outcome for the search is WON for the player to move. The markings indicate the relevancy zone, the circles indicate the `Rzone`$^+$-relevancy zone, which contain the intersections that can make the result better for Black (MAX) and the triangles indicate the `Rzone`$^-$-relevancy zone, intersections that, when changed, can make the result worse for Black (better for White). The two rectangles indicate the only intersections that belong to both `Rzone`$^+$ and `Rzone`$^-$.

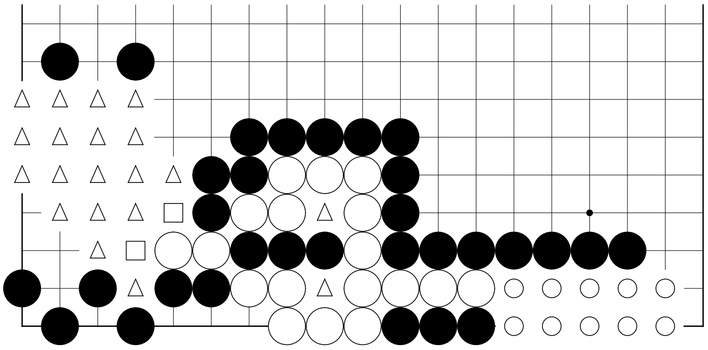


Fig. 3. Illustration of `Rzone`$^+$ and `Rzone`$^-$

One of the selective local search algorithms described in literature is lambda search [6]. Lambda trees are limited by the order of the moves (in lambda search, a λ^i attacking move threatens to do a λ^{i-1} move if the defender passes).

Abstract Proof Search [1] is based on similar ideas. However, the search can be both bounded by the depth and the order of the moves. Also, candidate moves cannot only be generated from the relevancy zones of lower order or lower depth searches, but also from goal and game specific hard-coded knowledge. The advantage of the latter is that often smaller sets of candidate moves can be used. Unfortunately, coding the knowledge turns out to be a quite cumbersome task which can only be done with some confidence for the lower order moves. For higher order moves, later versions of abstract proof search fall back on relevancy zones.

Several generalizations of abstract proof search have been proposed. Among these there is Iterative Widening and General Threats Search. Iterative widening [2] works in an analogous way as iterative deepening in that it performs an iteration of searches of increasing complexity by trying the most plausible candidate moves in the earlier searches and gradually considering more candidates resulting in 'wider' trees. Generalized Threats Search [3] is a generalization of the above algorithms.

In this paper we will use an implementation of abstract proof search (with the different generalizations) called TAIL, which was originally created by T. Cazenave and further extended in collaboration. The search algorithm is implemented in a generic goal-independent way. For each goal, a number of methods have to be specified; the most important ones are those listed in Table 1.

Table 1. Methods to specify for each goal

Method	Description
`do_move`(*move*,*color*)	perform a move and necessary updates to the goal status
`undo_move`()	retract a move
`eval`(*colortomove*,$\texttt{Rzone}^+$,$\texttt{Rzone}^-$)	evaluates the current position w.r.t. the goal and the player who moves first
`max_moves`(*order*,$\texttt{Rzone}^+$,$\texttt{Rzone}^-$)	returns the candidate moves for MAX up to order *order*
`min_moves`(*order*,$\texttt{Rzone}^+$,$\texttt{Rzone}^-$)	returns the candidate moves for MIN up to order *order*

We denote the application of a method M of some goal G in some position P with $G.M[P](args)$, where $args$ are the arguments. We will denote the relevancy zones of that call, i.e., the set of all intersections which may not change for the result of the method call to remain valid, with $\texttt{Rzone}^+\big(G.M[P](args)\big)$ and $\texttt{Rzone}^-\big(G.M[P](args)\big)$. One important method is the evaluation function. It either returns WON, KO, LOST, or UNKNOWN. The first three values are terminal, which means that the search can be stopped in nodes where these values are obtained. In this paper we do not make a distinction between the different kinds of ko. There can be several heuristic levels in UNKNOWN, according to the belief of the evaluation function in the possibility to reach the goal. We will denote these with $\textsc{Unknown}_i$ ($i \in \mathbb{R}$) where higher values of i are more desirable for MAX.

3 Compound Goals

In the previous section we have discussed existing work on searching atomic goals. In this section we explain how one can search for compound goals, mainly using the same search algorithms.

We start with defining a language that allows one to build more complex goals from atomic ones. Therefore one can use the three standard connectives: conjunction, disjunction, and negation. These complex goals are constructed as new goals, by defining the functions listed in Table 1, so that any search algorithm (for instance those described in the previous section) can be used to compute the outcome. We start with the simplest operator: the negation.

3.1 Negation

The negation of a goal is the inverse goal, where a player tries to prevent his opponent from achieving the basic goal. We define it more accurately below.

Definition 1. *Let G be a goal. Then $not(G)$ is a goal, called the negation of G, such that* $\textsc{max}(not(G)) = \textsc{min}(G)$ *and* $\textsc{min}(not(G)) = \textsc{max}(G)$*. Moreover, for every position P,* $G.\texttt{eval}[P](c) = \textsc{Won} \Leftrightarrow not(G).\texttt{eval}[P](c) = \textsc{Lost}$, $G.\texttt{eval}[P](c) = \textsc{Lost} \Leftrightarrow not(G).\texttt{eval}[P](c) = \textsc{Won}$ *and* $G.\texttt{eval}[P](c) = \textsc{Ko} \Leftrightarrow not(G).\texttt{eval}[P](c) = \textsc{Ko}$

The implementation of the necessary methods to be able to search for $not(G)$ given the methods for the basic goal G is in theory straightforward. Table 2 gives a summary. Note that because the roles of MAX and MIN are switched in the inverse goal, the roles of $\texttt{Rzone}^+$ and $\texttt{Rzone}^-$ also need to be reversed.

Table 2. Methods for the negation of a goal

Method	Implementation
not(G).max_moves(*order*, $\texttt{Rzone}^+$, $\texttt{Rzone}^-$)	G.min_moves(*order*, $\texttt{Rzone}^-$, $\texttt{Rzone}^+$)
not(G).min_moves(*order*, $\texttt{Rzone}^+$, $\texttt{Rzone}^-$)	G.max_moves(*order*, $\texttt{Rzone}^-$, $\texttt{Rzone}^+$)

However, there are a number of additional requirements on the basic goal in order for this schema to work. In particular, the evaluation function should be sufficiently accurate in returning the value LOST. Indeed, for the existing approaches, evaluation functions are often implemented asymmetric. It is important to return WON in positions where the goal is reached, but if UNKNOWN is returned instead of LOST in positions where the basic goal cannot be reached anymore, the search will take longer. However, it will still succeed in the same iterative deepening iteration. If the evaluation function is able to detect the LOST status early enough and the basic goal can be disproved by the original search algorithm then the negation of the goal can be proved efficiently. However, for a number of goals detecting the LOST status is more difficult than detecting the WON status. For instance, for the `capture` goal, the status is WON if the block to capture is removed from the board. Many playing programs assume that a block cannot be (easily) captured when its number of liberties is above some threshold, but even a block with tens of liberties can be dead. Hence, detecting that a block cannot be captured may require to recognize life.

3.2 Conjunction

A conjunction of two goals is the goal in which MAX is trying to achieve both goals. Below it is defined more accurately.

Definition 2. *Let A and B be goals such that* $\textsc{max}(A) = \textsc{max}(B)$*. Then $A \wedge B$ is a goal, called the conjunction of A and B, such that* $\textsc{max}(A \wedge B) = \textsc{max}(A)$ *and* $\textsc{min}(A \wedge B) = \textsc{min}(A) = \textsc{min}(B)$*. Moreover, for every position P,*

– *if* $(A \wedge B).\texttt{eval}[P](c) = \textsc{Won}$*, then* MAX *can win both A and B if c moves first;*
– *if* $(A \wedge B).\texttt{eval}[P](c) = \textsc{Ko}$*, then* MAX *can win both A and B by winning a ko, if c moves first;*
– *if* $(A \wedge B).\texttt{eval}[P](c) = \textsc{Lost}$*, then if c moves first, it is not possible for* MAX *to win both A and B, even not by winning a ko.*

We first propose a simple, static implementation of a compound goal construction for the conjunction goal that does not need search for the atomic goals independently (in 3.2.1). In a second step we will then discuss a dynamic implementation for $A \wedge B$ that performs local search for the goals A and B individually (in 3.2.2). Finally, we discuss the correctness of the conjunction goal search (in 3.2.3).

3.2.1 Static Functions

Below we discuss two types of static functions, viz. a static evaluation function and static candidate move generation.

A static evaluation function

We will denote the static evaluation function of the conjunction goal with $(A \wedge B).\texttt{eval}_\texttt{S}$. For the case MIN moves first, we define it by

$$(A \wedge B).\texttt{eval}_\texttt{S}[P](\textsc{min}) = A.\texttt{eval}[P](\textsc{min}) \wedge B.\texttt{eval}[P](\textsc{min})$$

where Table 3 gives the conjunction of two position evaluations.

The values in this table are only valid in the case that the two evaluations are independent. With MIN to move, this means that the actual static evaluation differs from the values of the "raw" evaluation function when the evaluation functions for A and B detect WON or KO before this status has actually been realized on the board. The remaining "aji" (latent possibilities of

MIN in the remaining proof of the realizability of the evaluation) of one goal could prevent the winning by MAX of the other goal in the case the relevancy zones ($\texttt{Rzone}^-$)of $A.\texttt{eval}[P](\text{MIN})$ and $B.\texttt{eval}[P](\text{MIN})$ overlap. In that case $(A \wedge B).\texttt{eval}_\texttt{S}[P](\text{MIN})$ should return a (high) UNKNOWN value to urge deeper search.

Furthermore, it is noteworthy that the mathematical intuition of conjunctions (taking the minimum of two values) is not applicable to the case of ko. Indeed, if both A and B evaluate to KO, then usually $A \wedge B$ evaluates to LOST as MAX can not win two ko fights at the same time, and $A \wedge B$ evaluates only to KO if both ko fights are in fact the same ko fight and MAX can win (lose) both A and B by winning (losing) this one ko. Even the conjunction of $A.\texttt{eval}[P](\text{MIN}) = \text{KO}$ and $B.\texttt{eval}[P](\text{MIN}) = \text{WON}$ can turn into $(A \wedge B).\texttt{eval}_\texttt{S}[P](\text{MIN}) = \text{LOST}$ instead of the expected KO in the (rare) case that B could only be won in double ko, providing an infinite source of ko threats for MIN to win the goal A.

We define the static evaluation function for the case where MAX moves first, by

$$(A \wedge B).\texttt{eval}_\texttt{S}[P](\text{MAX}) = \qquad (1)$$
$$\big(A.\texttt{eval}[P](\text{MIN}) \wedge B.\texttt{eval}[P](\text{MAX})\big) \vee \big(A.\texttt{eval}[P](\text{MAX}) \wedge B.\texttt{eval}[P](\text{MIN})\big)$$

where the disjunction $E_1 \vee E_2$ of two evaluations E_1 and E_2 can be found in Table 3.

Table 3. Conjunction and disjunction of two independent evaluations

E_1	E_2	$E_1 \wedge E_2$	$E_1 \vee E_2$
WON	WON	WON	WON
WON	KO	KO (or LOST)	WON
WON	LOST	LOST	WON
WON	UNKNOWN$_j$	UNKNOWN$_j$	WON
KO	WON	KO (or LOST)	WON
KO	KO	LOST (or KO)	WON (or KO)
KO	LOST	LOST	KO (or WON)
KO	UNKNOWN$_j$	UNKNOWN$_{j-c}$	UNKNOWN$_{j+c}$
LOST	WON	LOST	WON
LOST	KO	LOST	KO (or WON)
LOST	LOST	LOST	LOST
LOST	UNKNOWN$_j$	LOST	UNKNOWN$_j$
UNKNOWN$_i$	WON	UNKNOWN$_i$	WON
UNKNOWN$_i$	KO	UNKNOWN$_{i-c}$	UNKNOWN$_{i+c}$
UNKNOWN$_i$	LOST	LOST$_i$	LOST
UNKNOWN$_i$	UNKNOWN$_j$	UNKNOWN$_{\min(i,j)}$	UNKNOWN$_{\max(i,j)}$

The actual static evaluation function also differs here from the “raw” case when one of the conjunctions evaluates to LOST and the relevancy zones of the local searches overlap. In this case UNKNOWN is returned, so that the “global” search for the compound goal continues.

Static candidate move generation

When the atomic goals are not searched individually, the union of the candidate moves generated by the atomic goals have to be taken as the candidate moves for the compound goal. So,

$$(A \wedge B).\texttt{max\_moves}[P] = A.\texttt{max\_moves}[P] \cup B.\texttt{max\_moves}[P]$$
$$(A \wedge B).\texttt{min\_moves}[P] = A.\texttt{min\_moves}[P] \cup B.\texttt{min\_moves}[P]$$

3.2.2 Dynamic Functions

Below we discuss tow types of dynamic functions, viz. a dynamic evaluation function and dynamic candidate move generation.

A dynamic evaluation function

The evaluation function for the conjunction goal can be much more accurate by using local search to the components of this compound goal. This allows to prune much faster.

For every node P in the search for $A \wedge B$, the first call to a method of the goal $A \wedge B$ performs internally a search for the goals A and B with c moving first. If $A.\texttt{eval}[P](c)$ and $B.\texttt{eval}[P](c)$ respectively return UNKNOWN, for both $c =$ MAX and $c =$ MIN, this does not mean that potentially four independent searches have to be performed since to determine the set of candidate moves for MIN, the standard algorithm already does a search with MAX moving first (in order to obtain the relevancy zone). We will denote the results of these searches with $A.\texttt{search}[P](colortomove)$ and $B.\texttt{search}[P](colortomove)$ and the corresponding relevancy zones with $\texttt{Rzone}^{+}\big(A.\texttt{search}[P](colortomove)\big)$, $\texttt{Rzone}^{-}\big(A.\texttt{search}[P](colortomove)\big)$, $\texttt{Rzone}^{+}\big(B.\texttt{search}[P](colortomove)\big)$, and $\texttt{Rzone}^{-}\big(B.\texttt{search}[P](colortomove)\big)$. These local searches will either return a more informative value (WON, KO, or LOST) or will still return UNKNOWN. The latter means that a deeper local search is needed. If searching more deeply for the local goal is too expensive, so will the search for $A \wedge B$ be. This would mean that it is better to return UNKNOWN from the node P in the search for $A \wedge B$ rather than expanding it further. Of course, if $A \wedge B$ is searched using iterative deepening, in deeper iterations, more time can be allocated to the local searches.

Once both local searches are finished the function discussed in the previous paragraph is used to combine both results. As pointed out, if the relevancy zones of the two local searches overlap, the compound goal evaluates to UNKNOWN and deeper search is needed, by continuing the global search algorithm with the compound (conjunction) goal.

Using the information of the local searches, the appropriate intersections can be added to $\texttt{Rzone}^{+}\big(A \wedge B.\texttt{search}[P](colortomove)\big)$ and $\texttt{Rzone}^{-}\big(A \wedge B.\texttt{search}[P](colortomove)\big)$, so that one is able to construct hierarchies of goals.

Dynamic candidate move generation

If MAX moves need to be generated, this means that the evaluation function for MAX moving first (Equation 1) leads to UNKNOWN. In the game of Go, passing is allowed and hence $\texttt{eval}[\mathrm{P}](\textsc{max})$ will always be higher or equal to $\texttt{eval}[\mathrm{P}](\textsc{min})$, so one can concentrate on the $\texttt{eval}[\mathrm{P}](\textsc{min})$ evaluation. Further, because the individual goals are searched, one can find the proofs for the outcomes of these individual goals in the different relevancy zones.

Based on the results of $\mathrm{A}.\texttt{eval}[\mathrm{P}](\textsc{min})$ and $\mathrm{B}.\texttt{eval}[\mathrm{P}](\textsc{min})$, the set of candidate MAX moves can be obtained. If both these atomic goals evaluate to LOST, this means that MAX has to play a move that changes the result of both sub searches and therefore has to play a move from the intersection of the $\texttt{Rzone}^+$ of each atomic goal. If one of the above searches evaluates to WON, the MAX moves are those played on the intersections from the $\texttt{Rzone}^+$ of the other goal.

Moreover, candidate moves for MAX that do not allow MAX to win either goal A or B individually will be pruned away immediately by the local searches for goals A and B. In this way, one can hope that only a very small set of candidate moves for MAX will remain.

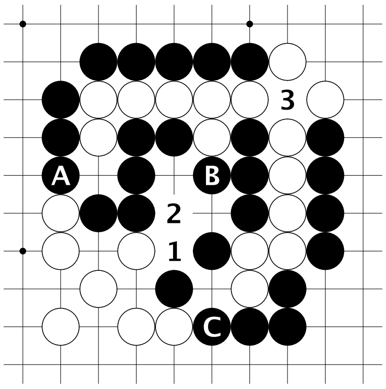
In a MIN node, whatever MIN does, MAX can win both goals independently. This is a consequence of the fact that after the local search for the goals independently, the node has not been pruned. So, MIN has to try a move in

$$\texttt{Rzone}^-\big(A.\texttt{search}[P](\textsc{min})\big) \cup \texttt{Rzone}^-\big(B.\texttt{search}[P](\textsc{min})\big)$$

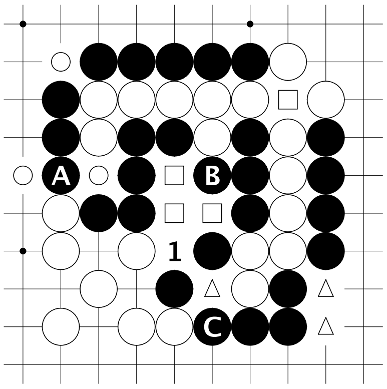
that makes miai of preventing MAX's win in A and B. In many cases, a move in $\texttt{Rzone}^-\big(A.\texttt{search}[P](\textsc{min})\big) \cap \texttt{Rzone}^-\big(B.\texttt{search}[P](\textsc{min})\big)$ will work for MIN, so these moves can be tried first.

While in [4] for the case of transitive connections, it is argued that some candidates for MIN are not in the union of the relevancy zones as described above. But as will be shown in the next paragraph, the fact that this union of relevancy zones is sufficient follows from the definitions of relevancy zones and conjunction goal.

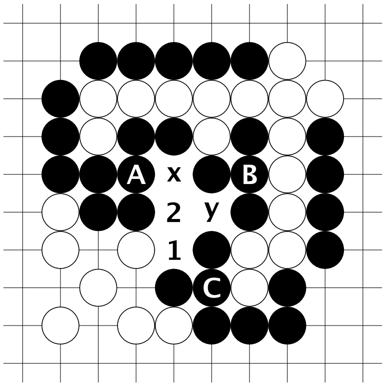
We illustrate this with the same example as in [4]. In the position in Figure 4a, we consider the connection of A and B and the connection of B and C. Figure 4b contains the $\texttt{Rzone}^-$-zones, the circles denote $\texttt{Rzone}^-\big(A - B.\texttt{search}[P](\textsc{min})\big)$, the triangles $\texttt{Rzone}^-\big(B - C.\texttt{search}[P](\textsc{min})\big)$, and the rectangles the intersection of the two. White 1 is a working move for MIN, which does not belong to the relevancy zone of the proof of either connection and subsequently is not considered as a candidate move. If White plays 1, Black cannot protect against 2 and 3 simultaneously. But this does not necessary mean that white 1 should indeed be a candidate move. Point 1 will appear in the $\texttt{Rzone}^-(A \wedge B)$ though, when the search for the compound goal (transitive connection) is continued, 2 is tried and Black defends at 1 against this by capturing the stone. As [4] already points out, 3 is also a working move on itself belonging to the intersection of the relevancy zones of both connections. Moreover, 3 is the only (and real) move that prohibits the transitive connection. So locally, 1 is a sente move against the connection, but not a threat that should be included in the set of candidate moves for MIN. Figure 4c shows a slightly modified example, where the move at 3 in the original example is solved for Black and consequently white 1 is no longer a working move.



a. original example



b. example with Rzone$^-$



c. modified example

Fig. 4. A counterexample for generating candidate moves for MIN

While it seems that a great deal of search should be performed to determine the status of a conjunction of goals, several optimizations can be applied. First, if a player moves outside the relevancy zone of one of both goals, the status of that goal does not change, and local search for that goal is not necessary in the next search node. Second, one can try to simplify the search by first trying those moves that minimize the intersection of the relevancy zones of the subgoals.

3.2.3 Correctness of the Conjunction Goal Search

Local searches are abstract proof searches. It only returns a terminal result (LOST,WON) if this is the correct result. When a result involving ko is returned, it is possible that a deeper search still obtains a 'better' result (WON instead of KOWON, or LOST instead of KOLOST).

To prove the correctness of the proposed compound search algorithm, we must show that all candidate moves are correctly generated and the evaluation function is correct. When the evaluation function returns a value which is not in the Unknown range, it is correct that this result can be obtained (this follows from the definition and the properties of the local abstract proof searches). Still, it could be that a 'better' (non-ko instead of ko) result can be obtained if local searches are searched deeper and change from non-ko to ko.

We show that all necessary candidate moves are generated. With this we mean that there is no move which is not considered as candidate move and could yield a better result for either player. In the above subsection on dynamic candidate move generation using the relevancy zones of the local searches, we already showed that in the case none of the local searches evaluates to Unknown, the set of candidate moves is correct. In the case one of the local searches evaluates to Unknown, a correct and complete set of candidate moves cannot be derived. Still, if the global search gives a solution with a principal path of nodes where no subgoals evaluated to Unknown, the result is correct. In the other case, one can get a more certain answer by deepening the local searches in the principal path that returned Unknown, or by iterating the global search and adding in nodes with subgoals returning Unknown, the appropriate relevancy zones of the search starting in this node in the previous global iteration, to the set of candidate moves.

3.3 Disjunction

A disjunction of two goals is a goal where max tries to realize at least one of both goals. Defining a disjunction of goals is possible using the classical laws of logic.

Definition 3. *Let A and B be goals such that* $\text{max}(A) = \text{max}(B)$*. Then $A \vee B$ is a goal, called the disjunction of A and B, such that*

$$A \vee B = not\big(not(A) \wedge not(B)\big)$$

This shows in fact that the properties of a disjunction of goals will be dual to the properties of a conjunction of goals. Of course, in practice a direct implementation without the negations turns out to be faster.

4 Experiments

In this section we will present experiments in order to evaluate our compound goal search and compare it with existing methods.

In order to make our implementation to solve compound goals we had first to construct basic goals producing good relevancy zones, also for goals that cannot be solved in a few moves. This provided an interesting opportunity to evaluate the use of relevancy zones as compared to hard coded knowledge. The use of relevancy zones does not mean that no hard coded knowledge is used. Some minimal goal-dependent knowledge is used. In the `capture` goal, for instance, the fact is used that one needs to play on a liberty to capture a block in 1 move. However, we use no move selection based on knowledge of the games or heuristics for move ordering.

4.1 Atomic, Relevancy-Based Games

In several experiments, it seems to turn out that search using hard coded knowledge is often faster, while search using relevancy zones solves more problems. Table 4 gives an impression of the results for different thresholds for the maximum nodes allowed for three datasets. These datasets were provided by T. Cazenave and contain problems from games amongst computer programs together with some artificial problems. For two of them the goal is to capture an opponent string or defend a friendly string from capturing. These datasets contain 144 and 75 problems. In the third dataset of 90 problems, the goal is to connect two friendly strings or disconnect two opponent strings.

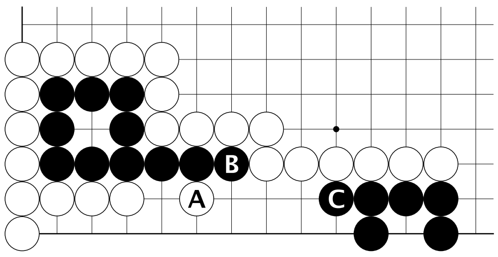
Table 4. Comparing search using (mainly) relevancy zones to search using hard-coded knowledge. For different thresholds and for 'maximum nodes to traverse', the timings of both methods and the number of correctly solved problems is given.

goal	test set	nodes/problem	hard-coded		relevancy zones	
			time(s)	solved	time(s)	solved
capture	1	1000	0.57	57%	0.64	51%
		20000	1.18	57%	2.96	57%
		200000	1.85	58%	11.41	82%
	2	1000	0.27	33%	0.25	32%
		20000	0.50	35%	1.47	44%
		200000	0.77	35%	5.72	50%
connect	1	1000	0.56	58%	0.66	68%
		20000	1.24	60%	0.77	68%
		200000	3.98	68%	0.78	68%

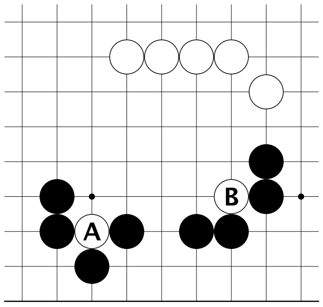
4.2 Compound Goals

Now we move to compound goals. However, there seems to be no large dataset on compound goals available. We will therefore limit our discussion to a smaller set of hand-constructed and real-game problems. Most problems from games are transitive connection problems from [4]. In future work, we will collect a larger dataset of problems from games requiring compound goal techniques.

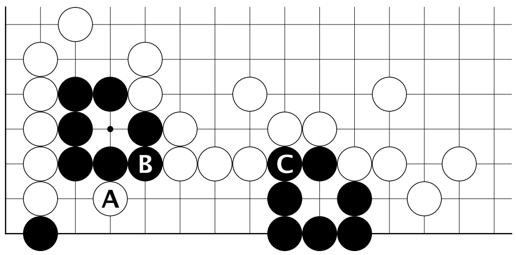
Table 5 lists a number of compound goals problems. We consider basically two methods to solve compound goals. The first one is the method described up to now. The second (naive) one is a method without search local to one of the subgoals. Candidate moves are just the union of the candidate moves proposed by the separate goals. This is about the best guess one can do without using relevancy zones and using no local search. As can be seen, for some problems



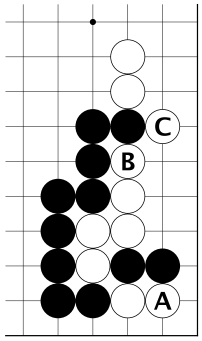
a. `capture`(A)∨ `connect`(B,C)



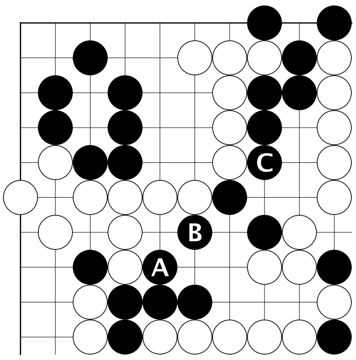
b. `capture`(A)∧ `capture`(B)



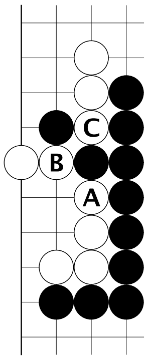
c. `capture`(A)∨ `connect`(B,C)



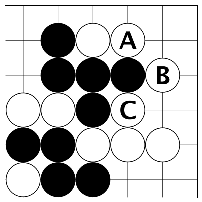
d. *not*(`connect`(A,B)∧ `connect`(B,C))



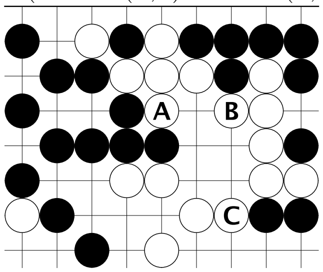
e. *not*(`connect`(A,B)∧ `connect`(B,C))



f. *not*(`connect`(A,B)∧ `connect`(B,C))



g. *not*(`connect`(A,B)∧ `connect`(B,C))



h. `connect`(A,B)∧ `connect`(B,C)

Fig. 5. Compound goal problems

the naive method works and is much faster. However, on several problems the naive method fails, i.e., it does not find a correct solution (a.o. due to missing candidate moves). Also, the naive method is not guaranteed to produce correct results. Of course, instead of this naive method one could try to take a safe approximation of the correct set of candidate moves, but this is very difficult because the complexity rapidly blows up as unnecessary candidates are added.

Table 5. Compound goals problems. Timings are given in seconds.

	goal	dynamic	static
1 (Figure 1)	`capture`(D3)∧`capture`(M3) (Won)	0.14	timeout
2 (Figure 5a	`capture`(A)∨ `connect`(B,C) (Won)	0.39	timeout
3 (Figure 5b	`capture`(A)∧ `capture`(B) (Won)	0.10	timeout
4 (Figure 5c	`capture`(A)∨ `connect`(B,C) (Won)	0.39	timeout
5 (Figure 5d	*not*(`connect`(A,B)∧ `connect`(B,C)) (Won)	1.12	0.01
6 (Figure 5e	*not*(`connect`(A,B)∧ `connect`(B,C)) (Won)	1.12	0.01
7 (Figure 5f	*not*(`connect`(A,B)∧ `connect`(B,C)) (Won)	1.62	0.01
8 (Figure 5g	*not*(`connect`(A,B)∧ `connect`(B,C)) (Lost)	0.15	wrong
9 (Figure 5h	`connect`(A,B)∧ `connect`(B,C) (Won)	0.55	0.03

5 Conclusions and Further Work

We proposed a general method for searching for compound goals. While existing techniques can guess good moves quite fast for such goals, our compound goal search technique is more accurate. Also, it is more general than existing techniques. Hence it allows one to determine accurately the status of a larger area on the board than existing methods. We considered atomic goals and three connectives to combine them: negations, disjunction, and conjunction. The binary connectives depend on correct computation of relevancy zones of their subgoals.

There are several directions of further work. First, a more accurate treatment of ko could make the search algorithm much more valuable in real games. When combining goals, it can be important to know what kind of ko is the local result of each of the subgoals.

Second, we have limited our discussion to the main logical operators (negation, conjunction, and disjunction). However, other operators may be useful. For instance, consider the problem of making a maximal number of points in some area (e.g., during an invasion). Then, the subgoal would return a real value instead of the logical values Lost, Won, and Ko considered here. Combining such goals would be much more similar to decomposition search.

Finally, a more memory-intensive approach could avoid much recomputation. In our current approach, a global transposition table is used. However, it would be interesting to consider local transposition tables for every subgoal, which store also relevancy zones. This would probably save much computation, while if care is taken would still fit easily into current memory capacity.

Acknowledgements

Jan Ramon is a post-doctoral fellow of the Fund for Scientific Research (FWO) of Flanders. Tom Croonenborghs is supported by the Flemish Institute for the Promotion of Science and Technological Research in Industry (IWT). We thank Tristan Cazenave and his colleagues for the collaboration on the code used for this paper ("tail").

References

1. Tristan Cazenave. Abstract proof search. In T. Marsland and I. Frank, editors, *Proceedings of the Second International Conference on Computers and Games*, volume 2063 of *Lecture Notes in Computer Science*, pages 39–54, Hamamatsu, Japan, 2000. Springer-Verlag.
2. Tristan Cazenave. Iterative widening. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, volume 1, pages 523–528, 2001.
3. Tristan Cazenave. A generalized threats search algorithm. In Schaeffer J, M. Müller, and Y. Björnsson, editors, *Proceedings of the Third International Conference on Computers and Games*, volume 2883 of *Lecture Notes in Computer Science*, pages 75–87, Edmonton, Alberta, Canada, 2002. Springer-Verlag.
4. Tristan Cazenave and Bernard Helmstetter. Search for transitive connections. *Information Sciences*, 175:284–295, 2005.
5. Martin Müller. Decomposition search: A combinatorial games approach to game tree search, with applications to solving go endgames. In *Proceedings of the International Joined Conference on Artificial Intelligence*, volume 1, pages 578–583, 1999.
6. Thomas Thomsen. Lambda search in game trees with applications to Go. In T. Marsland and I. Frank, editors, *Proceedings of the Second International Conference on Computers and Games*, volume 2063 of *Lecture Notes in Computer Science*, pages 57–80, Hamamatsu, Japan, 2000. Springer-Verlag.
7. Steven Willmott, Alan Bundy, John Levine, and Julian Richardson. An adversarial planning approach to Go. In H.J. van den Herik and H. Iida, editors, *Proceedings of the First International Conference on Computers and Games*, volume 1558 of *Lecture Notes in Computer Science*, pages 93–112, Tsukuba, Japan, 1998. Springer-Verlag.