TOWARDS MULTI-OBJECTIVE GAME THEORY – WITH APPLICATION TO GO

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

KEYWORDS Combinatorial Games, Multiple Objectives, Computational Intelligence, Dependence of Games, Threat, Game of Go, Ko

ABSTRACT

We define a multi-goal as a conjunction and/or disjunction of ordinal-scaled objectives. We give exact formulas to compute the conjunction and disjunction of independent combinatorial games associated with the objectives. Dependence of games is formalized. We also propose a definition for the (con/dis)junction of effectively dependent games. In all the above formulas, we can work with uncertain and unresolved (ko) outcomes. With these formulas, the status of a multi-goal can be computed with considerable less effort compared to current search approaches. Algorithms to compute multiple solutions elegantly and to extract the threats to a won game from its search tree are outlined, implemented and applied.

INTRODUCTION

Multiple objectives (or shorter: multi-goals) are commonly used by human players of strategic games like Chess and Go. For instance, in Chess, one could aim at simultaneously attacking a horse and a rook. In Go, one can rescue an endangered group either by connecting it to another living group or by making life on its own. Achieving a multi-goal does not necessarily imply to win the game, the other part of the trick is to choose the right set of goals to strive for. The use of multi-goals helps the player to obtain overview and structure in the game.

Computers are not humans and do not have to follow the same (multi-goal) approach as humans. Deep Blue beat human Chess world champion Kasparov in 1997 with the single goal of getting a better position than the opponent, using a brute force approach where heuristics gave an estimate of the state of the game. Other good Chess programs use a similar approach.

Go is a game far more complex than Chess. Its average branching factor is around 240, compared to around 40 for Chess. Full board evaluation is expensive to compute and cumbersome to design and implement, whereas for Chess exist fast and reasonably good heuristics. Go needs a different approach (Wilmott e.a., 1999; Bouzy and Cazenave, 2001).

In their respected computer Go survey, Bouzy and Cazenave (2001) pointed out that a full board Go evaluation function needs tactical searches to determine properties like safety and connectivity. Several Go programs use multiple goals in some way. However, there is little theoretical work on this subject or publications about it. Bouzy and Cazenave conclude that "the problem of performing tree search on conjunctions and disjunctions of goals remains to be solved". Also, "An interesting idea would be to formalize (...) the interaction between several games".

This paper addresses the problem of multi-goals, i.e. conjunctions and disjunctions of goals. First, we give an introduction to the game of Go and Combinatorial Game Theory. Then we discuss independent multi-goals, followed by a treat of dependent games. We continue with algorithms for finding multiple solutions and threats, show some experimental results and end with concluding remarks.

THE GAME OF GO

Go is an ancient game, originated in China about 4000 years ago. It is a two-player, deterministic, complete-information, partizan, zero-sum game. Go is played on a 19×19 grid (although other sizes are also used), which is initially empty. The two players have to embark territory by alternately placing a stone on a grid, gradually building strongholds and eventually walls that completely surround one's territory. The goal of the game is to end with more territory than your opponent.

The rules of Go are very simple in principle (but in finesse, different rule sets like the Chinese, Japanese or mathematical Go rules vary quite a bit). The *capturing rule* is the most important, stating that a string of stones gets captured if all of its neighbouring intersections (called *liberties*) are occupied by enemy stones. For example, white can capture the four black stones

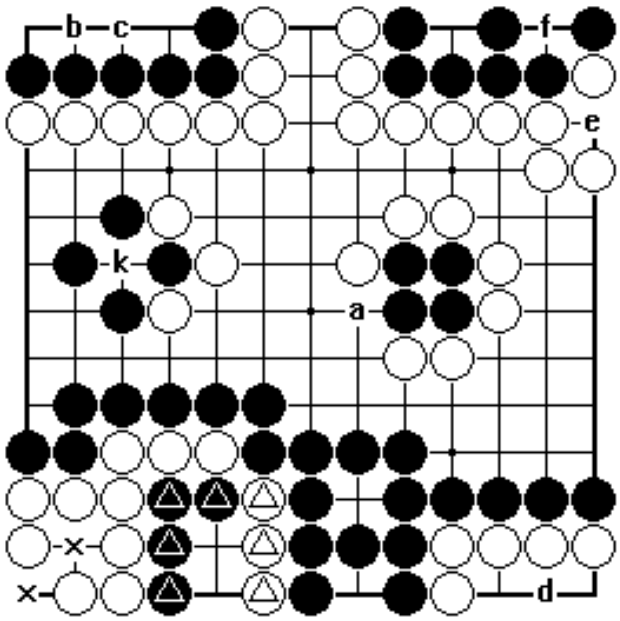

Figure 1: Examples on the Go rules and game states

in the middle left of figure 1 by playing *a*. It also implies that the two *x*'s are suicide, which implies in turn that the white group that surrounds them cannot be captured (Go terminology: the group lives). The white group can only be captured if white would cooperate foolishly and plays at one of the *x*'s himself. Black is then allowed to play the "temporary suicide" of the other *x*, because this would capture the entire white group and the suicide is resolved. The black group in the upper left should also live, even if white moves first. If white *b*, then black *c* (and vice versa) and black has two eyes.

The *ko rule* is also important. The simplest formulation is that immediate recapture is prohibited. However, the ko rule varies a lot over different rule sets. A useful and general formulation is that a move is forbidden if it repeats the board position. An example of a ko is the situation around *k*. White can capture a black stone by playing *k*, and if black were to recapture immediately by playing to the right of *k* the board situation would be repeated. Without a ko-rule this could go on and on until one of the players falls asleep.

COMBINATORIAL GAME THEORY

This section introduces (the representation of) Combinatorial Game Theory. It is a mathematical theory for two-player games and numbers, developed by J.H. Conway (1976) and adapted to many games by Berlekamp, Conway and Guy (1982). However, it can also be used for subgoals like "capture string" or "kill group". A combinatorial game can be associated with every goal.

Definition 1 *A combinatorial game* $G = \{F|O\}$ *is composed of two sets* F *and* O *of combinatorial games. Every combinatorial game is constructed this way.*

The left part of G, F, can be seen as the set of board positions that player *Friend* can reach with one legal move. Right part O can be looked at as the options for player *Opponent*. Essentially, F can have two possible values, W (a win for Friend) or L (a loss for Friend, so a win for Opponent). If Friend has a legal move which ensures a won game, then the value of F is W. For a moment we assume that we have enough computing power to compute the values, otherwise we would have to introduce another symbol to represent uncertain outcomes. This gives four possible states for a combinatorial game: W|W, W|L, L|L and L|W (we will use both WW and W|W as abbreviations for {W|W}). The left half of a game value is the maximum result that Friend can obtain, the right half is Opponent's best result.

W|W denotes a game that is won by Friend, irrespective of who moves first (both player can at best move the game to W = a win for Friend). This means that Friend does not have to spend a move to win the game. A Go example is the (life status of the) white group in the bottom left of figure 1, which lives unconditionally. Another example is the black group in the upper left, it lives even if white moves first (white *b*, black *c* and vice versa).

W|L is an unsettled game, it is won by the player who moves first. An example is the life status of the white group in the bottom right corner. If white moves first he can play at *d*, resulting in a living shape (two eyes). If black plays *d*, the resulting shape is dead (one eye only).

L|L is a lost game for Friend, so a sure win for Opponent, even if Friend moves first. It is the opposite of W|W. Killing the black group in the upper left corner is a lost game for white.

L|W is a somewhat strange equilibrium situation where the player who moves first will lose the game. In Go, this situation is known as seki. The triangled stones form a seki: either player who wants to capture the opponent string of triangled stones and plays at one of the two shared intersections will immediately be captured himself. Not playing in this game is best for both players.

In essence, a game can have only two values, W or L. However, it is often intractable to compute the precise game value. Cazenave (1996) therefore extended Conway's theory to uncertain outcomes. He introduced a symbol U to denote an uncertain game value. Cazenave showed that U can be used as a control parameter along the risky-safe axis, since one is free how to evaluate of U. Evaluating U as if it were L models a very conservative strategy, evaluating U as if it were W models a highly risky strategy. More neutral strategies are equally well possible.

In Go, even more symbols are useful because of the ko rule. The outcome of a game can be ko, like the life and death situation of the black group in the upper right of figure 1. In order to make f his second eye, black needs to capture the white stone with e and prevent recapture. The outcome of this ko will eventually depend on the existence of threatening moves somewhere else on the board, since after a threat is answered by the opponent the ko-recapture is no longer prohibited. In the end, a ko fight will be either W or L, depending on the number of threats of each side. However, it would be wise to call the outcome "ko" and not to try and resolve the ko immediately (i.e. during a search procedure), by searching lines of play starting with a ko-threat. This would mix up local and global issues, causing all kinds of complications, result in higher branching factors and might not even be necessary (e.g. example 8).

There are different types of ko: ko with Opponent to find a threat first, ko with Friend to find a threat first, indirect ko's like multi-stage ko and multi-step ko and even more exotic types. For clarity, we will only introduce symbols for the first two, most common, types. We denote them KO↑ and KO↓, respectively. KO↑ means that Friend can move the game to W (if the ko rule allows it). Opponent can move KO↑ to KO↓, upon which Friend will have to find a useful ko-threat first, before he can move the game back to KO↑. If Friend has no useful ko-threat, Opponent can move the game to L with a second play. Summarizing,

$$KO\uparrow = \{W|KO\downarrow\}, \quad KO\downarrow = \{KO\uparrow |L\},$$

if the ko-rule allows the required move.

Game notations like L|L, U|L, W|L and W|KO↓ correspond to what Go players call the *status* of a game. It is a summary of the outcomes of a game with either of the players moving first. However, Combinatorial Game Theory is more general. The left and right parts of a game are games themselves, each having a left and right part (remember that the definition of a combinatorial game is recursive). These latter left and right parts denote the results which can be achieved if a player moves twice in a row in the same game (e.g. if the opponent passes or plays in a different game).

We now have new games like for example W|L||L|L. This is a game which Opponent can move to the right side of the double vertical bar, i.e. L|L or lost for Friend. Friend can only move the game to W|L, unsettled, upon which Opponent can answer and move WL to L in the usual way. If Friend got a chance to play a second move in a row, he can move this game to W. In other words, this game is won for Opponent, but Friend has a threat to snatch victory away if he gets two moves in a row. In Go, such moves are useful ko-threats.

We can also have games like KO↑|L||L|L|||L|L||L|L, where it would take Friend three moves in a row to turn this lost game into KO↑ (and a fourth to win the ko). And so on.

When there is no ambiguity in how to read games, we will drop some vertical bars and even some outcome symbols. For example, W|L||L|L is also denoted WL|LL and sometimes simplified to WL|L. KO↑|L||L|L|||L|L||L|L becomes KO↑LLL|LLLL and can be simplified to KO↑LLL|L.

Some games have outcomes representing a score, for instance the territory game in Go. The symbols of such games correspond to natural numbers and valuate on a nominal scale, whereas W, L, U, and ko outcomes are symbols valuating on an ordinal scale. These games can be added up, for instance to find the total 'amount' of territory (amount is written between inverted commas, as in general a sum of games remains a game and not a number). It makes little sense, though, to add up symbols of an ordinal scale. Conway's famous formula for adding up two *independent* games G and H is as follows (Conway, 1976),

$$G + H = \left\{G^L + H, G + H^L | G^R + H, G + H^R\right\},$$

where G^L and G^R denote the left and right part of G, respectively. The commas indicate that both players have two options, to move in either G or H.

Sums of *independent* games is a well-studied subject in combinatorial game theory, see for instance (Berlekamp e.a., 1982). It has been applied to Go endgames, where the games are (almost) independent, resulting in nearly perfect play and performance superior to the best professional players. On the contrary, research on sums of *dependent* games is still in its infant stage. There are currently two approaches, each having its own drawbacks: (1) assume that the games are independent and (2) merge the games into one bigger game.

To our knowledge, the only publication on multiple goals in two-player games is (Willmott e.a., 1999). It devises a method to solve conjunctions of goals, by means of hierarchical planning. Although this results in a greatly reduced search space, the method requires hand-coded domain knowledge and is very sensitive to gaps in the knowledge-base. Moreover, disjunctions and combinations of conjunctions and disjunctions of goals are not treated.

MULTI-GOALS

Multi-goals may appear in many games, at least in the mind of a strategic player, but also in real world situations. They come in many forms, sometimes expressing one wants to do two or more things simultaneously, another time it enumerates different means to achieve the same goal. Our definition is as follows.

Definition 2 *A multi-goal is a logical expression of two or more ordinal-scaled objectives in two-player games.*

A logical expression is a conjunction and/or disjunction. Ordinal-scaled means that the symbols (W, L, etc.) are partially ordered. A combinatorial game can be associated with a multi-goal just as well as with a single goal.

Example 3 $H = G_1\ AND\ (G_2\ OR\ G_3)$.

This example expresses a general multi-goal, not tied to a particular game. The G_n could perfectly well be associated with Chess goals like capture pawn or isolate queen. The multi-goal would have little sense, though, had the compound goals integer-valued outcomes. For such games, one would rather use a sum of games.
Multi-goal generation could be automated by means of hierarchical decomposition, or simply by hand-coding. With automated multi-goal generation, one would be close to a Go playing machine. We will not elaborate on it here.
Multi-goals are not at all new. Their status can be computed using the same algorithms as for single goals, e.g. alpha-beta- or proof-number search. The main differences between current single- and multi-goal search are the evaluation function and move generation. A typical current multi-goal evaluation function looks the same as a single-goal evaluation function from the outside, it evaluates a goal. Internally, the multi-goal evaluation function will do several single-goal evaluations (one for each compound goal G_n) and then evaluates the logical expression, compared to just one for a single-goal evaluation function. A multi-goal move generator will be some sort of combination of the move generators of the compound goals. For example, a simple, uninformed and straightforward method would be to generate the union of all the moves, generated by the respective compound move generators. With the triple-goal of example 3, this method would result in an average branching factor of around three times the average branching factor of one of the G_n's search tree.
The current method of computing multi-goals is not efficient. It does not employ possible independence of the compound goals. When the games in a multi-goal are independent, then they can be solved separately (with a relatively small branching factor) after which the logical expression can be computed in almost no time. With this method, the computation time can be sped up quite dramatically.
Thomsen (2002) ran an experiment on a simple double-goal. Solved separately, the goals were disproved with 80 nodes. Solved as a multi-goal, using a method comparable to the straightforward one above, the disproof took 2520 nodes. This shows it is indeed worth to implement a computational intelligent approach (Thomsen also pointed out that it is of even greater importance in multi-goals to use a transposition table).
We propose the following definition for a disjunction of two independent goals G and H (inspired by Conway's formula for addition):

Definition 4 $G\ OR\ H = \left\{G^L\ OR\ H, G\ OR\ H^L \mid G^R\ OR\ H, G\ OR\ H^R\right\}$.

The definition for a conjunction of two independent goals is obtained by replacing OR by AND. Definition 4 expresses that both players can choose whether they play in G or H (e.g. if Friend plays in G, then he moves the multi-goal from G OR H to G^L OR H). In order to compute multi-goals, we just need the next definition, in which g is an arbitrary game.

Definition 5
$$\begin{array}{l} W\ OR\ g = W \\ L\ OR\ g = g \\ W\ AND\ g = g \\ L\ AND\ g = L \\ U\ OR\ U = \overline{U} \\ U\ AND\ U = \underline{U} \end{array}$$

$\overline{U}$ is also an uncertain outcome, but we designate it an extended symbol, because it is less uncertain to win one of two uncertain games than just one. $\underline{\text{U}}$ is more uncertain than U, because it means winning both of two uncertain games. Summarizing the partial ordering of all the symbols: W > $\overline{U}$ > U > $\underline{\text{U}}$ > L and W > KO↑ > KO↓ > L. The ordering between uncertain and ko outcomes is subject to a strategy regarding risk. One is free to adapt this strategy at any moment to the state of the game.
Further we assume each player can win an uncertain game if he can make an extra move in it. This implies for instance that W|U = W||U|L and U|L = W|U||L (note that this is different from saying U^L = W and U^R = L).
The following four examples, applying definitions 4 and 5, all assume the games are independent. Black is player Friend.

Example 6

$W|L\ OR\ W|L =$
$W\ OR\ WL,\ WL\ OR\ W \mid L\ OR\ WL,\ WL\ OR\ L =$
$W, W \mid WL, WL =$
$W \mid WL.$

This example shows that it always possible to win one of two unsettled games. Friend does not have to play in this game to win it, Opponent just has threats. The third line expresses both players have two ways to achieve the same result. Moving in either of the two W|L games yields the same result for both players. A Go instance of this multi-goal is the upper side of figure 2. Black string 2 has one eye. It can get a second eye either by connecting to string 1 or by catching string 3. The game Connect(string 2, string 1) is W|L. Black wins it with a, but black loses after white a. The game Catch(string 3) is also W|L. The vital move for both

players is b. Black can always win on of the two goals, so string 2 lives. White only has a and b as (ko) threats. Playing both of them would kill string 2.

Example 7

WL AND WL =
{W AND WL} | {L AND WL} =
WL|L.

Winning both of two unsettled games is a lost game, as it would take two moves in a row. A threat is the only result for Friend. An instance concerns string 4, it lives when both C and D become an eye. The games Make_eye(C) and Make_eye(D) are both W|L, the vital point for both players is e, respectively f. So, string 4 is dead, e and f are threats for black to save his string.

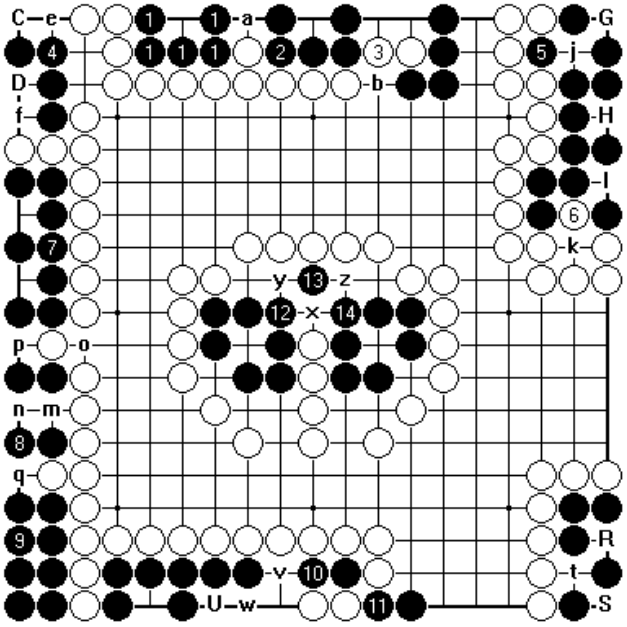

Figure 2: A collection of multi-goal examples.

Example 8

KO↑ OR KO↓ =
{W|KO↓} OR {KO↑|L} =
W OR KO↑|L, ... || KO↓ OR KO↓, W|KO↓ OR L =
W || W|KO↓, W|KO↓ =
W|WKO↓.

This is also a won game for Friend, regardless the amount of ko-threats at Opponent's disposal. All that is in it for Opponent is a threat to make a ko (in the exotic situation that Opponent has a double-ko elsewhere on the board, the supply of ko-threats is infinite; the outcome will then depend on the rule-set). An instance is the black group in the upper right. It has already one certain eye, H. The other potential eyes, G and I, are both subject to a ko fight. Make_eye(G) is KO↑, black can win it with j, white can move the ko fight to KO↓ with j. If white wins the ko with a second move at *1*, then G has become a so-called false eye. Even though it is surrounded by two black stones, G is not a suicide for white. A false eye is of no use for living. Make_eye(I) is KO↓, black will lose it if white covers at k, turning I into a false eye. Black can turn this game into KO↑ by catching stone 6 with k and possibly win it with a second move at *6*. However, as example 8 shows, black does not have to play in order to win the multi-goal. White cannot win both ko's. If white plays in one ko fight, black can simply play in the other.
Definition 4 applies to two-compound multi-goals. As the conjunction/disjunction of two games returns a game, a multi-goal of three (or more) goals can be computed by applying definition 4 first to two of the compound goals and then to the result and the third compound (and so on, were there more goals).

Example 9

WU AND (WL OR UL) =
WU AND {W OR UL, ... | L OR UL, WL OR L} =
WU AND {W | UL, WL} =
WU AND W|UL =
WU AND WU =
{W AND WU ||| U AND W|U} =
{WU ||| ... , U AND W || L, U AND U} =
WU|U<u>U</u>.

DEPENDENT GAMES

Conway's formula for addition and definition 4 for conjunction and disjunction of games apply to independent games. They embody that both players must choose which of the two games to play in. In general, a move can play in more than one game. Then the games are not independent and the above formulas do not apply. We will give four examples of a move which plays in two goals, in which Friend plays white.
In games with moving pieces, like Chess, it is common that moving a piece to achieve one goal renders it unusable to achieve another goal, for which it was also needed. This situation is an analogy of the *Sussman anomaly* in hierarchical planning (Sussman, 1975), where the postconditions of an action achieving one goal conflict with the (previously satisfied) preconditions of another goal. In games with non-moving pieces like Go, the Sussman anomaly is less common. It would occur if a winning move in one game is a losing move in another game, while a winning move does exist. A Go example is the left side of figure 2, where string 7 already lives. Connect(string 8, string 7) is W|W: white m, black n, white o, black p and string 8 is saved. Connect(string 9, string 8) is W|L. However, by connecting string 9 to

string 8) black takes away a liberty of string 8. Consequently, Connect(string 8, string 7) has become W|L and after white m, black n, white can capture a 14-stone string with p. We say these two games are *Sussman-dependent*.
Another case of goal dependency is when a move simultaneously achieves two goals. An instance is at the bottom right of figure 2, where black can turn both U and V into eyes by playing w. White can turn them both into false eyes with w.
When a friend move plays both in WL and in WL|LL, Go players call that move *sente*. It achieves one goal and at the same time threatens another goal. Opponent will have to reply if he wants to ensure the latter. (There is also another kind of sente move, playing just in one nominal-scaled game. It makes a little profit with the first move, but threatens big profit with a second.) In a way, a sente move achieves a goal for free, as it holds the initiative. An instance is white v. It wins Capture(string 10) and next threatens w, turning Make_eye(U) into L (playing w immediately does not work).
A double threat by Opponent plays in two W|WL games. A double threat can lead him to victory in one of two (dependent) WW games! Usually, Friend will then have to choose which game to give up, unless he has a double winning move to rescue both games simultaneously. In that case, we speak of an ineffective double threat. An example of an effective double threat is white x. Black would like to connect his one-eyed strings 12 and 14 in order to live. Connect(string 12, string 13) and Connect(string 13, string 14) are both won (W|WL), but they share threat x. After white x, black cannot guard against y and z simultaneously.
Current Combinatorial Game Theory has its focus on the value of games. The notation of a game does generally not include the move(s), which achieves a certain value. As said above, to determine dependence between games: winning and threatening moves are indispensable. To detect Sussman-dependency in Go, one needs to keep track of the losing moves as well. For the rest of this article, we will assume the games are not Sussman-dependent.

Definition 10 *A move is an* achieving move *if it is the first move in a game and achieves a better result compared to when the opponent would have moved first. A move achieving a win is called a* winning move.

In a W|L game, all moves achieving W are winning moves for Friend. All moves achieving L are winning moves from Opponent's point of view. In games like KO↓|L and U|L, the moves leading to KO↓ and U are achieving moves for Friend. A game like WL|LL does not have achieving moves, as the game is already a win for Opponent.

Definition 11 *A move is an* n-move *if it is one of n moves in a row, which together achieve a better result than when the opponent responds in the meantime. A 2-move is simply called a threat or a direct threat. An achieving move is also called a 1-move.*

The most straightforward direct threat is in games like WL|LL and WW|WL, where Friend respectively Opponent have to move twice in a row to turn the result around. KO↓L|LL or UL|LL is a threat for Friend to turn a lost game into a ko respectively an uncertain outcome. Cazenave (1996) identified WU|UU as a general threat for Friend. However, we prefer to also consider games like UL|LL and KO↓L|LL a threat.
3-moves correspond to games like WLLL|L and W|WWWU, these games can be moved to a direct threat. Generally, an n-move moves a game to an (n–1)-move game (n>1). A direct threat moves a game to unsettled.

Definition 12 *Two combinatorial games are* (n,m)-dependent *if n-moves of one game overlap with m-moves of the other, else they are* (n,m)-independent. *Moves in the overlap are called* (n,m)-moves.

Two unsettled games are (1,1)–dependent if a move plays in both of them. An example of (2,2)-dependency is when a double threat occurs. A sente move, holding the initiative, expresses (1,2)-dependency between two games.

Definition 13 *Two combinatorial games are* effectively (n,m)-dependent *if the opponent has no move which successfully answers an (n,m)-move in both games simultaneously, else they are* effectively (n,m)-independent.

Whether or not a (n,m)-move is effective has to be computed carefully, but we will not elaborate on that in this article.

Definition 14 *The* tally *of a game is the number of moves in a row, for which the game has been evaluated.*

Example 15
tally(W|L) = 1,
tally(WW|WL) = 2,
tally(WWWW|WWUL) = 3).

Please note that the number of symbols doubles with every increment of the tally. Now it is time to introduce an axiom on effectively dependent games.

Axiom 16 *If Friend (Opponent) has an effective (n,m)-move on combinatorial games G and H and if G and H are games at tally n respectively m, then Friend (Opponent) can move the multi-goal G OR H to G^L OR H^L (respectively to G^R OR H^R).*

For effectively independent games definition 4 still holds.

Example 17

Friend has an effective double winning move in two games G and H. Then:
$G\ AND\ H =$
$WL\ AND\ WL =$
$(WL)^L\ AND\ (WL)^L \mid L\ AND\ WL =$
$W\ AND\ W|L =$
$W|L.$

Compare W|L to WL|L for (1,1)-independent and effectively (1,1)-independent games.

Example 18

Opponent has an effective (2,2)-move in games G and H. Then:
$G\ AND\ H =$
$W|WL\ AND\ W|WL =$
$W\ AND\ W|WL \parallel (W|WL)^R\ AND\ (W|WL)^R =$
$W|WL \parallel WL\ AND\ WL =$
$W|WL \parallel WL|L \simeq$
$W|L$ *(simplified to tally 1).*

Both players can move to a game where the opponent only has a threat. Compare W|L to WL|L for (effectively) (2,2)-independent games.

ALGORITHMS

As one can see in the previous section, all achieving moves and direct threats are important in multi-goals. Below we will shortly describe two algorithms, one to compute multiple solutions and one to compute direct threats. Next, we will run them on some Go situations to determine (n,m)-dependency.
We used the generic proof-number search algorithm from the PubGo++ package (publicly available at www.bath.ac.uk/~eespjl/go.html), and extended it to our purposes. Proof-number search was invented by Allis (1994). It is neither a depth-first algorithm like alpha-beta, nor is it breadth-first. It is best-first, each iteration expanding the most promising frontier node, based on a pair of so-called proof-and disproof-numbers. Each node in the search tree has such a pair, recording the number of (grand)child nodes needed to prove (disproof) in order to prove the node itself. It is (almost) the A* analogue for two-player search.
A disadvantage of proof-number search is that it only has proven, disproved and uncertain outcomes. In order to find ko's, one could redo the search with another evaluation function, which would judge a ko outcome as proven. We have not yet implemented this. Another characteristic of proof-number search, often referred to as a disadvantage, is that the whole search tree must be stored in memory. We, however, see this as an advantage, at least as far as the computation of multiple solutions and threats is concerned. When the algorithm has found one solution, it has also spent some time on trying other solutions. These efforts do not go to a waist, we simply reuse relevant parts of the tree. To find threats, we post-process the search tree.
Our method for finding another solution for *any* node X in the tree is as follows:

1. temporarily detach the existing solution node(s) from X,
2. make X the new root of the tree (if it was not the root node already),
3. restart the search till another solution is found,
4. re-attach the already found solution(s) to X.

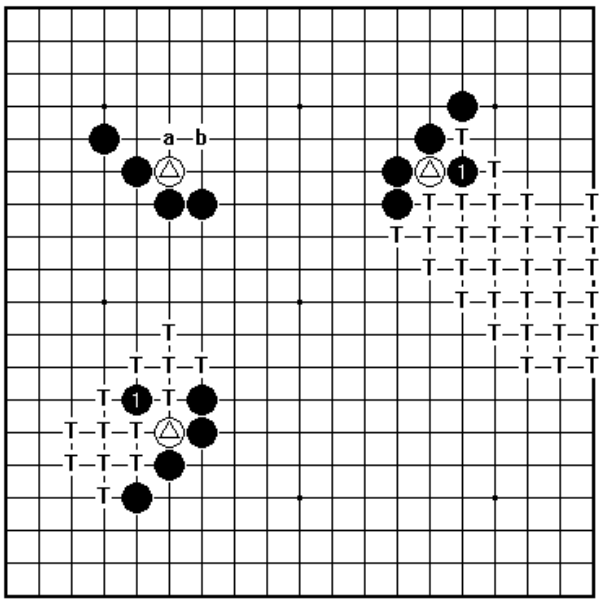

Figure 3: Black to capture the triangled stone. Winning moves and threats are depicted.

This algorithm is simple and elegant. For the computation of multiple winning moves we apply this method to the root node.
The idea behind our method for computing the threats to a proven node is that every intersection in a *proven* line of play is a potential threat. Had there been an opponent stone on that intersection, then that line of play could be thwarted. Lines of play not resulting in a proof are neglected, they did not achieve a result and therefore there is nothing to be threatened. This is the biggest difference with a very simple and straightforward method, described by Thomsen (2001), which would consider a threat every move that has been played in the search. His method does not need any tree post-processing, but simply records all the moves played in

a search. Disadvantage is that some nodes, possibly many, are falsely classified as a threat (although Thomsen's lambda-search method avoids most misclassifications indirectly by avoiding unnecessary lines of play).
The pseudo code of our depth-first method is as follows, where THREATS is a global variable collecting the threats:

```
FIND_THREATS(NODE){
  WHILE(CHILD = NODE.NEXTCHILD()){
    IF(CHILD.ISPROVEN()){
      THREATS.ADD(NODE.MOVE());}
    ELSE{CONTINUE WHILE;}   // SKIP THIS CHILD
      FIND_THREATS(CHILD);
  ENDWHILE;
}
```

After the above steps, we add all the neighbours of THREATS, since a move on these points also has the potential to disturb a winning line of play, as Thomsen (2001) already pointed out. However, these threats may appear to be ineffective.
Although our method for computing threats is different than Thomsen's, it yields similar results.

EXPERIMENTAL RESULTS

In this section we apply the two algorithms of the previous section. We keep it brief as we did not implement all the details of the theory in this paper yet, only the (essential) parts of finding winning and threatening moves. The goal is to capture the triangled stone in figure 3, black to play. The same situation is depicted three times under rotation. In the upper left quadrant the output of the multiple solution algorithm is given. We used a simple move generator. For the attacker (black) it adds moves at the liberties of the target string and intersections next to the liberties (in Go terminology: ladders and loose ladders). For the defender it adds the same moves plus attacks on vulnerable stones surrounding the target string (ataris). Two solutions were found, *a* and *b*. With both of them, black moves the capture game to W|WL.
The upper right shows the output of the threat finding algorithm after black has played solution *a*, the bottom left shows the output after solution *b*. Although many more moves than the T's were played during the search, the algorithm was able to find all the real threats. However, it did falsely identify a few threats, mainly because of the final step in the algorithm where it adds all the neighbours of THREATS (see previous section).

CONCLUSIONS AND FUTURE WORK

We defined multi-goals as logical expressions of ordinal-scaled objectives and defined a formula for the conjunction and disjunction of two independent games. In this formula, we can use uncertain and (Go-specific) unresolved outcomes (ko). The formula introduces some computational intelligence into the calculation of multi-goals, which current approaches lack. We formalized dependence between games, including sente moves and double threats. We proposed a new formula for the conjunction and disjunction of two effectively dependent games. Algorithms to compute multiple solutions and threats have been outlined, implemented and applied.
Our plans for the near future are to implement the formulas for dependent and independent games and some other issues necessary to fully automate the computation of multi-goals. Another points is to enhance proof-number search with unresolved outcomes like ko.

REFERENCES

Allis, L.V., 1994. *Searching for Solutions in Games and Artificial Intelligence*, PhD thesis, University of Limburg, Maastricht.

Berlekamp, E., J.H. Conway and R.K. Guy. 1982. *Winning Ways (for your mathematical plays).* Academic Press, New York.

Bouzy, B. and T. Cazenave, 2001. "Computer Go: an AI Oriented Survey". Artificial Intelligence, Vol 132(1), pp. 39-103.

Cazenave T., 1996. *Systeme d'Apprentissage par Auto-Observation. Application au Jeu de Go.* PhD thesis, Université Pierre et Marie Curie, Paris.

Conway J.H., 1976. *On Numbers and Games.* Academic Press, New York.

Sussman, G.J., 1975. *A Computer Model of Skill Acquisition*, Elsevier, New York.

Thomsen, T., 2001. "Lambda-Search in Game Trees - with Application to GO". ICGA journal.

Thomsen, T., 2002. Personal communication.

Willmott, S., J. Richardson, A. Bundy and J. Levine, 1999. "An Adversarial Planning Approach to Go." In H.J. van den Herik and H. Iida, eds., Computers and Games: Proceedings CG'98, Springer-Verlag, Japan.