Go-Moku and Threat-Space Search

L.V. Allis
H.J. van den Herik

M.P.H. Huntjens

University of Limburg
P.O. Box 616
6200 MD Maastricht, The Netherlands
{allis,herik}@cs.rulimburg.nl

Vrije Universiteit
De Boelelaan 1081
1081 HV Amsterdam, The Netherlands
matty@cs.vu.nl

Abstract

Many decades ago, Japanese professional Go-Moku players stated that Go-Moku (Five-in-a-row on a horizontally placed 15×15 board) is a won game for the player to move first. So far, this claim has never been substantiated by (a tree of) variations or by a computer program. Meanwhile many variants of Go-Moku with slightly different rules have been developed. This paper shows that for two common variants the game-theoretical value has been established.

Moreover, the Go-Moku program *Victoria* is described. It uses two new search techniques, threat-space search and proof-number search. One of the results is that *Victoria* is bound to win against any (optimal) counterplay if it moves first. Furthermore, it achieves good results as a defender against non-optimally playing opponents. In this contribution we focus on threat-space search and its advantages compared to conventional search algorithms.

1. Introduction

Japanese professional Go-Moku players have stated for many decades that the player to move first (i.e., Black) has an assured win (Sakata and Ikawa, 1981). Still, up to this contribution, as far as we know, no proof has been published nor has any Go-Moku program ever shown to be undefeatable when playing Black. As a case in point we mention the game between the Go-Moku 1991 World Champion program *Vertex* (Black) and the program *Polygon* (White), *Vertex* maneuvered itself into a position provably lost for Black (Uiterwijk, 1992a).

The intriguing difference between human strategic abilities and the performance of the state-of-the-art computer programs makes Go-Moku an attractive research domain. The investigations focus on the discovery of new domain-specific strategic knowledge and on the development of new search techniques. Both issues are discussed below although emphasis is placed on a new search technique.

The course of this article is as follows. In section 2, the rules of five Go-Moku variants are briefly discussed. In section 3, some insight is given into the way of thinking by human Go-Moku experts, the "thinking" of the strongest computer programs and the main differences between them. Section 4 contains a description of threat-space search. The results of this new search technique are presented in section 5. The Go-Moku program *Victoria* is described in section 6. The main result is given in section 7: how *Victoria* solved the common variant of Go-Moku. Section 8 contains conclusions.

2. The rules of Go-Moku

In Go-Moku, simple rules lead to a highly complex game, played on the 225 intersections of 15 horizontal and 15 vertical lines. Going from left to right the vertical lines are lettered from 'a' to 'o'; going from the bottom to the top the horizontal lines are numbered from 1 to 15. Two players, Black and White, move in turn by placing a stone of their own color on an empty intersection, henceforth called a *square*. Black starts the game. The player who first makes a line of five consecutive stones of his color (horizontally, vertically or diagonally) wins the game. The stones once placed on the board during the game never move again nor can they be captured. If the board is completely filled, and no one has five-in-a-row, the game is drawn.

Many variants of Go-Moku exist; they all restrict the players in some sense, mainly reducing the advantage of Black's first move. We mention five variants.

A. In the early days the game was played on a 19×19 board, since Go boards have that size. This variant is still occasionally played. However, the larger board size increases Black's advantage (Sakata and Ikawa, 1981).

B. An overline is a line of six or more consecutive stones of the same color. In the variant of Go-Moku played most often today, an overline does not win (this restriction applies to both players). Only a line of exactly five stones is considered as a winning pattern.

C. Black plays his first move usually at the center square and his second move diagonally

connected to his first move. In a commonly played variant it is forbidden to Black to play his second move in the 5×5 square of which his first move forms the center. This restriction reduces Black's advantage considerably, but not completely.

D. A professional variant of Go-Moku is Renju. White is not restricted in any way, e.g., an overline wins the game for White. However, Black is not allowed to make an overline, nor a so-called three-three or four-four (cf. Sakata and Ikawa, 1981). If Black makes any of these patterns, he is declared to be the loser. Renju is not a symmetric game; to play it well requires different strategies for Black and for White. Even though Black's advantage is severely reduced, he still seems to have the upper hand.

E. In an attempt to make the game less unbalanced, in Renju a choice rule for White has been introduced: Black must play his first move in the center, White at one of the eight squares connected to Black's stone; Black's second move is unrestricted. Then, White has the choice between continuing the play with the white stones, or swapping colors with Black. [When playing optimally, Black will, if possible at all, choose his second move such that the game-theoretical value of the resultant position is drawn.]

We illustrate our search techniques with the non-restricted variant of Go-Moku: an overline is allowed and sufficient to win, for either player. In section 7, we present the solution of this variant as well as of variant B. The optimal lines are given in Appendix A.

3. Expert knowledge

In this section, three types of Go-Moku knowledge and their applications are discussed. The first type consists of a number of definitions important for all players (humans and computers). We apply these definitions to threat sequences. The second type contains strategies taken from human Go-Moku players' thinking. The third type embodies the choice-of-move selection process of conventional Go-Moku programs. Then, we examine to what extent move-selection approaches by humans and by computers are different. As a result it transpires which ability is lacking in conventional Go-Moku programs.

3.1. Definitions and threat sequences

In Go-Moku, a threat is an important notion; the main types have descriptive names: the *four* (diagram 1a) is defined as a line of five squares, of which the attacker has occupied any four, with the fifth square empty; the *straight four* (diagram 1b) is

a line of six squares, of which the attacker has occupied the four center squares, while the two outer squares are empty; the *three* (diagram 1c and 1d) is either a line of seven squares of which the three center squares are occupied by the attacker, and the remaining four squares are empty, or a line of six squares, with three consecutive squares of the four center squares occupied by the attacker, and the remaining three squares empty; the *broken three* (diagram 1e) is a line of six squares of which the attacker has occupied three non-consecutive squares of the four center squares, while the other three squares are empty. If a player constructs a four, he threatens to win at the next move. Therefore, the threat must be countered immediately. If a straight four is constructed, the defender is too late, since there are two squares where the attacker can win on his next move. With a three, the attacker threatens to create a straight four on his next move. Thus, even though the threat has a depth of two moves, it must be countered immediately. If an extension at both sides is possible (diagram 1c), then there are two defensive moves: both directly adjacent to the attacking stones. If only one extension is possible then three defensive moves are available (diagram 1d). Moreover, against a broken three, three defensive moves exist (diagram 1e).
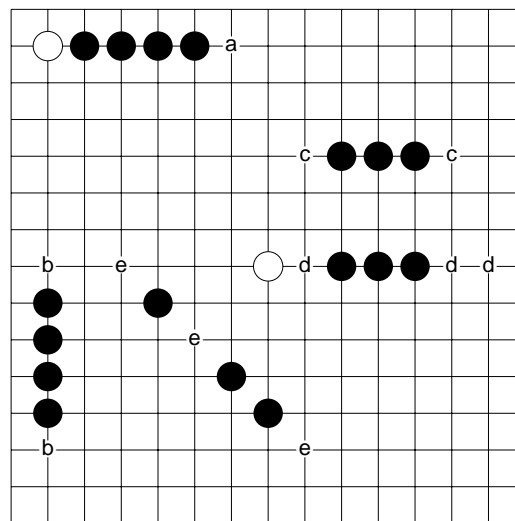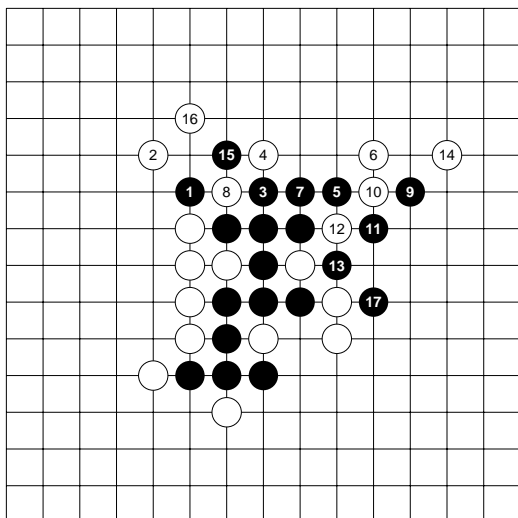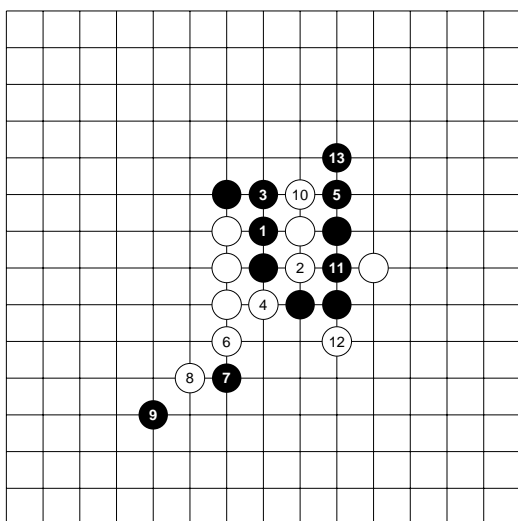

Diagram 1: Threats.

To win the game against any opposition a player needs to create a double threat (either a straight four, or two separate threats). In most cases, a threat sequence, i.e., a series of moves in which each consecutive move contains a threat, is played before a double threat occurs. A threat sequence leading to a (winning) double threat is called a *winning threat sequence*. Each threat in the sequence forces the defender to play a move countering the threat. Hence, the defender's possibilities are limited.

In diagram 2a a position is shown in which Black can win through a winning threat sequence

consisting of fours only. Since a four must be countered immediately, the whole sequence of moves is *forced* for White.

Diagram 2a: A winning threat sequence of fours.

Diagram 2b: A winning threat sequence of threes.

In diagram 2b a position is shown in which Black wins through a winning threat sequence consisting of threes, occasionally interrupted by a white four. As mentioned earlier, White has at each turn a limited choice. During the play, he can create two fours as is shown in diagram 2b. Still, his loss is inevitable. We remark that the exact winning threat sequence by Black is dependent of White's responses to the threes. Therefore, in such cases, preference is given to the notion of *winning threat trees*. In practice, for the difference between the notions *sequence* and *tree*, inessential fours by the defender are disregarded (see 6. below).

3.2. Human expert analysis

During the second and third Computer Olympiad (Levy and Beal, 1991; Van den Herik and Allis, 1992) we have observed two human expert Go-Moku players (A. Nosovsky, 5th dan, and N. Alexandrov, 5th dan). These Russian players are involved in two of the world's strongest Go-Moku playing programs (*Vertex* and *Stone System*). While observing the experts, it became clear that they are able to find very quickly sections on the board where a winning threat sequence can be created, regardless of the number of threes which are part of the winning sequence. The length of these winning sequences are typically in the range of 5 to 20 ply.

The way a human expert finds winning sequences so quickly can be broken down into the following steps.

1. A section of the board is chosen where the configuration of the stones seems favorable for the attacking player. It is then decided whether enough attacking stones collaborate making it useful to search for a winning sequence. This decision is based on a "feeling", which comes from a long experience in judging patterns of stones (cf. De Groot, 1965).

2. Threats are considered, and especially the threats related to other attacking stones already on the board. Defensive moves by the opponent are mostly disregarded.

3. As soon as a variation is found in which the attacker can combine his stones to form a double threat, it is investigated how the defender can refute the potential winning threat sequence. Whenever the opponent has more than one defensive move, examination is started whether the same threat sequence works in all variations. Moreover, it is investigated whether the opponent can insert one or more fours neutralizing the attack.

4. If only some variations do not lead to a win via the same threat sequence, examination is started whether the remaining positions can be won via other winning sequences.

5. In practical play, a winning threat sequence often consists of a single variation, independent on the defensive moves.

6. Notably, the size of the search space is considerably reduced by first searching for one side (the attacker). Only if a potential winning threat sequence is found, the impact of defensive moves is investigated. This approach is supported by the analyses given in Sakata and Ikawa (1981). When presenting a winning threat sequence, they only provide the moves for the attacker, thus indicating that the sequence works irrespective of the defensive moves. Possible

fours which the defender can create without refuting the threat sequence are neglected all together.

In positions without winning threat sequences, the moves to be played preferably increase the potential for creating threats, or, whenever defensive moves are called for, the moves chosen will reduce the opponent's potential for creating threats. The human evaluation of the potential of a configuration is based on two aspects: (1) direct calculations of the possibilities, e.g., if the opponent does not answer in that section of the board and (2) a so-called good shape, i.e., configurations of which it is known that stones collaborate well.

3.3. Computer programs' strategies

The strongest Go-Moku programs use (variants of) the α-β search algorithm (Knuth and Moore, 1975). In each position only a restricted number (normally 10-15) of the best-looking moves are considered. The selection is based on heuristics. Searching a variation ends when a win (loss) has been found, or when a pre-determined depth has been reached; then a heuristic evaluation is performed. Some programs also use forward pruning in branches where an intermediate evaluation leaves little hope for success. When threats are involved, the defender has a limited number of moves; the strongest programs then search up to a depth of 16 ply. In the opening phase of the game, all programs use an opening book.

There are some important differences between programs. For instance, *Polygon* (Uiterwijk, 1992b) searches first for a winning threat tree without using a heuristic evaluation function. Using transposition tables *Polygon* reaches in some variations depths of 20 to 30 ply (or even more). If no winning tree is found, an *n*-ply deep search is performed using a strategic heuristic evaluation function (*n* is mostly a small number); it then selects a move. Before the strategic move is played, it examines whether the opponent has a winning tree after the move is played. If so, the move is rejected, and a new move is generated by the strategic search.

Another instance is the 1991 World Champion *Vertex*. It does not use transposition tables nor special search techniques. A standard search depth of 16 ply, a fine-tuned selection of the most-promising 14 moves per position, and an extended opening book have resulted in a strong program.

3.4. Man vs. machine

Obviously, both humans and programs spend some time to search for winning threat sequences. Humans use methods of optimistic search and verification, programs use conventional tree-search techniques (Newell and Simon, 1972). In the latter case,

superior calculation power tries to make up for the large amount of extra work and to compensate the heuristic evaluation function, which is clearly inferior to human evaluation. A challenge for AI research is investigating how experts can be outsmarted by clever techniques and not achieving the human playing strength by heavy searching (cf. Levinson *et al.*, 1992). The aim could be realized by formalizing the human's complicated methods and then applying the surplus of calculation power.

In diagram 3a we provide an extreme example. Black has 16 places where he can create a four. In total, Black has $8! \times 2^8 \approx 10,000,000$ ways of creating a series of 8 moves (with 8 forced responses). When transposition tables are used, still more than 6,000 variations exist. For humans, however, it is immediately clear that none of the variations leads to a winning threat sequence. Somehow humans understand that executing all these threats leads to nothing.

In diagram 3b, a position similar to diagram 3a is given. Black now has a winning threat sequence. It is conceivable that a Go-Moku program only after long searching finds the winning threat sequence. Its tree might be polluted by unnecessary threats carried out in non-relevant sections of the board.
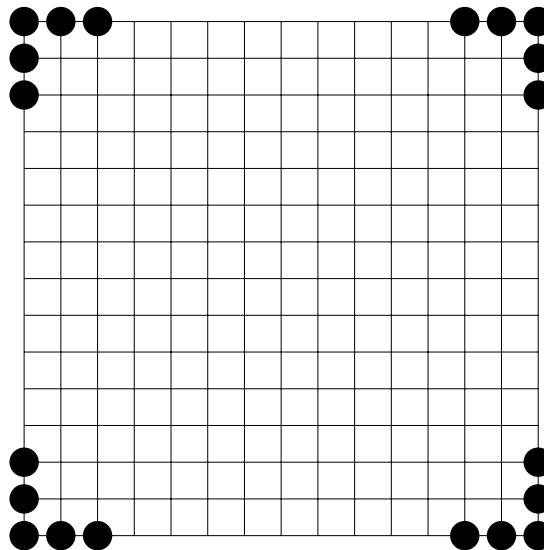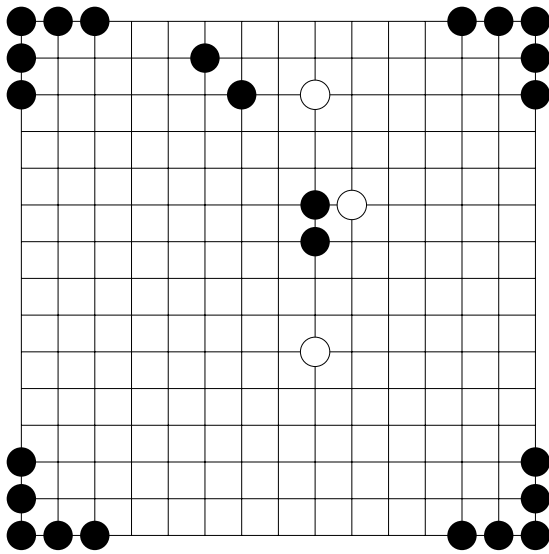

Diagram 3a: A position with many loose threats.

Diagram 3b: A position with real threats.

We conclude that conventional tree-search algorithms do not mimic human experts when searching for a winning threat sequence. Below we provide a model attempting to formalize the human search strategy. This model needs the introduction of threat-space search.

4. Threat-space search

A winning threat sequence consists of threats. Therefore we concentrate on the space of all threats. The size of the space often comprises several millions of positions. Hence, emphasis is placed on (1) reducing the size, and (2) searching through the remaining space as efficiently as possible.

A substantial reduction comes from the determination of the defensive moves connected to a threat. From section 2, we know: after a four, one defensive move is possible; after a three, two or three are possible. If we consider, for instance, after a three all possible countermoves, the number of variations rapidly increases. From section 3.2, however, it appears that human experts mostly find winning sequences in which the choices by the opponent are irrelevant. This idea is used to reduce the space. Instead of choosing between the defensive moves, we allow the opponent to play all possible countermoves at the same time. If we still find a winning threat sequence, it is sure that the opponent's moves are inessential. The drawback might be that in some positions no winning threat sequence is found, while still one exists. In section 5 this drawback is treated and a remedy is given.

In brief, we have reduced the search space to a space of attacking moves (threats) only, each of them answered by *all* directly defensive moves. For a precise description of threat-space search we introduce six notions, their definitions follow below.

1. The *gain square* of a threat is the square played by the attacker.

2. The *cost squares* of a threat are the squares played by the defender, in response to the threat.

3. The *rest squares* of a threat are the squares containing a threat possibility; the gain square excepted.

4. Threat *A* is *dependent* on threat *B*, if a rest square of *A* is the gain square of *B*.

5. The *dependency tree* of a threat *A* is the tree with root *A* and consisting of dependent nodes only, viz. the children of each node *J* are the threats dependent on *J*.

6. Two dependency trees *P* and *Q* are in *conflict*, if within dependency tree *P* a threat *A* exists and within dependency tree *Q* a threat *B*, in such a way that (1) the gain square of *A* is cost square in *B*, *or* (2) vice versa, *or* (3) a cost square in *A* is also cost square in *B*.

We exemplify the definitions. In diagram 3b, playing *e15* creates a four (with gain square *e15*, cost square *d15*, and rest squares *a15*, *b15* and *c15*). After the moves *e15* and *d15*, playing *i11* creates a four (with rest squares *e15*, *f14* and *g13*). Thus the four with gain square *i11* is dependent on the four with gain square *e15*, since *e15* is rest square within the threat with gain square *i11*.

The dependency tree of threat *i11* is a tree whose root is the four with gain square *i11*; the only child of the root is the threat with gain square *e15*. Since this last threat consists of rest squares which are already present on the board, it does not depend on any threats.

A clear example of definition 6 is the threat with gain square *e15* (and cost square *d15*) and the threat with gain square *d15* (and cost square *e15*). These two threats cannot both be executed in one and the same winning threat sequence. Therefore, these threats are in conflict. If threats have a large dependency tree, it is a great deal of work to examine whether the dependency trees are in conflict. However, the extra work outweighs the work involved in the normal investigation.

Threat-space search can now be described with the help of two principles.

1. Threat *A* being independent of threat *B* is not allowed to occur in the search tree of threat *B*.

2. In the threat-space search tree only threats for the attacker are included. After a potential winning threat sequence has been found, it is investigated whether the sequence can withstand any counterattack.

In Table 1, we have shown the threat-space search tree of the position of diagram 3b.

| Depth	Type of threat	Gain square	Cost squares
1	Four	l15	k15
1	Four	k15	l15
1	Four	e15	d15
2	Four	i11	h12
3	Straight Four	i8	i7
2	Four	h12	i11
1	Four	d15	e15
1	Four	o12	o11
1	Four	o11	o12
1	Four	a12	a11
1	Four	a11	a12
1	Three	i11	i7,i8,i12
2	Four	h12	e15
2	Four	e15	h12
3	Five	d15	
1	Three	i8	i7,i11,i12
1	Four	o5	o4
1	Four	o4	o5
1	Four	l1	k1
1	Four	k1	l1
1	Four	e1	d1
1	Four	d1	e1
1	Four	a5	a4
1	Four	a4	a5

Table 1: The search tree of diagram 3b.

Each line in table 1 contains the depth, the type of threat, the gain square and the cost squares. In the search tree 18 independent threats exist: 16 fours in the corners of the board and two threes in the range of *i7-i12*. For 16 of these threats (15 fours and 1 three) the gain square does not contribute anything to creating new threats. Hence the 16 nodes are terminals in the threat-space search tree, since the first principle of threat-space search states that a child in a search tree must be dependent on its parent. Only two threats produce a gain square exploitable in subsequent threats. We discuss them below.

After *e15* (with cost square *d15*), two new fours can be created, viz. on *i11* and *h12*. The gain square *h12* (with cost square *i11*) results in no new threats. After *i11* (with cost square *h12*), a straight four can be created at *i8*. Since a straight four guarantees a win, a potential winning threat sequence is found.

After *i11* (with cost squares *i7*, *i8* and *i12*), two new fours can be created, viz. at *h12* and *e15*. The gain square *e15* (with cost square *h12*) leads to the creation of a five at *d15*. Thus a second potential winning threat sequence has been found. The total threat-space search tree consists of 24 nodes only, with all potential threat sequences found.

In the example of diagram 3b two potential winning threat sequences have been found in a process where each subsequent threat is dependent on stones already on the board: every new gain square originated from a previous threat. In some positions, however, the gain squares of two or three independent threats can be combined to create a new threat. In such cases independent nodes should be linked.

The resultant combination can be the ancestor of new threats. We note that it is only useful to link independent nodes when the gain squares potentially combine to a new threat, i.e., they must lie on a single line, and close to each other for a possible five-in-a-row.
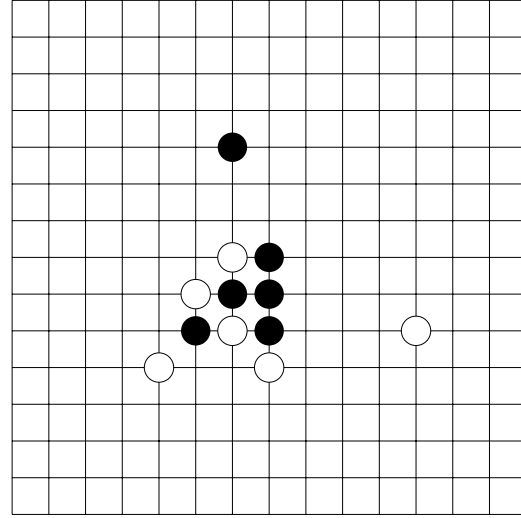
Diagram 4: A winning gain-square combination.

Diagram 4 shows a position where Black can win by combining gain squares of independent threats. In Table 2 we show the initial threat-space search tree. The tree does not contain straight fours or fives. This means that no potential winning threat sequence has been found.

Depth	Type of threat	Gain square	Cost squares
1	Four	h10	h9
1	Four	h9	h10
1	Four	j10	i9
1	Four	i9	j10

Table 2: Initial search tree for diagram 4.

With the four gain squares of Table 2 we can create six combinations of two gain squares. We remark that *h9* and *h10* are both gain squares and lie close enough on the same line, but their dependency trees conflict. Therefore, the combination *h9-h10* is not added to the threat-space search tree. For the same reason, *i9-j10* is not added to the search tree. The combination *h9-j10* does not lie on a line. The three combinations where the gain squares are close enough on the same line, and whose dependency trees do not conflict are *h10-i9*, *h10-j10* and *h9-i9*. In Table 3 we have shown the three potentially beneficial combinations. For each combination, we have, if possible, developed a new search tree, using the principles of threat-space search. It turns out that the combination *h10-i9* can be used to create a straight four at *f12*. The combination of threats {*h10*, *i9*} followed by *f12* thus forms a potential

winning threat sequence.

Depth	Type of threat	Gain sq.	Cost sq.
1	Four	*h10*	*h9*
2	combination with i9	*i9*	*j10*
3	Straight Four	*f12*	*e13*
2	combination with j10	*j10*	*i9*
1	Four	*h9*	*h10*
2	combination with i9	*i9*	*j10*
1	Four	*j10*	*i9*
1	Four	*i9*	*j10*

Table 3: Combinations of threats for diagram 4.

The two examples clearly indicate the working of threat-space search. We now provide the general description. From a given position a search tree of dependent threats is developed. Whenever a branch ends in a straight four or five it is examined whether the potential winning sequence can be refuted. If none of the variations leads to a win for the attacker, combinations are made between all independent nodes, with the condition that the gain squares involved lie on a single line and close to each other. This process is repeated until a winning threat sequence has been found, or no new potentially beneficial combinations can be created. Generalizing the idea, we state that threat-space search linearizes part of the search process.

5. Results of threat-space search

For Go-Moku no standard set of test positions is available, comparable to the Bratko-Kopec test for chess (Kopec and Bratko, 1982). We have chosen 12 positions earlier investigated by the program *Polygon* (Uiterwijk, 1992b) for testing the efficiency and efficacy of threat-space space. For this purpose we have developed *Victoria Threat-Space Search*, meaning that *Victoria TSS* does not use any additional search techniques. Here, two remarks are in order.

First, in the test positions the defending player has almost no interrupting threats. Moreover, the attacking player has only a few threats not related to the area of interest. Obviously, in positions with many independent threats on both sides the difference between threat-space search and conventional search is expected to be more manifestly.

Second, the number of nodes searched by *Polygon* are not directly comparable to the number of nodes searched by *Victoria TSS*, the reason being (1) *Polygon* uses knowledge rules reducing the depth of search, (2) *Polygon* uses transposition tables, (3) *Victoria TSS* only generates moves locally (in relation to the gain square of the current parent node); *Polygon* generates moves globally (threats at any section of the board, independent of the parent node). As a result we mention that on the same hardware, *Victoria TSS* searches 5,000 nodes per second and *Polygon* 500.

Finally we remark that *Polygon* has adjustable knowledge levels. Its 1-ply and 3-ply knowledge rules are comparable with *Victoria TSS*'s knowledge: a five is a 1-ply rule, and the straight four is an example of a 3-ply rule (Uiterwijk, 1992c). In Table 4 we compare the results on the 12 test positions. The measures are expressed in number of nodes searched. The *Polygon* version is described in Uiterwijk (1992b); we distinguish between the version using only the 1-ply and 3-ply rules, and the version using the knowledge rules up to 7 ply.

Pos.	Poly. 3-ply	Poly. 7-ply	Vict. TSS
1	433	419	134
2	2863	2367	(n.s.) 976
3	1715	1585	526
4	1810	1797	303
5	5821	888	153
6	7267	4826	643
7	1658	1429	83
8	16811	2875	193
9	4635	518	677
10	6274	6049	(n.s.) 104
11	775	505	(n.s.) 558
12	8294	6776	504

Table 4: Measuring the efficacy and efficiency.

In three positions (2, 10 and 11) *Victoria TSS* did not find a winning line (indicated by n.s., not solved), the reason being our reduction of the search space described in section 4. After (n.s.) we provide the number of nodes necessary to traverse the restricted search space. As an aside, we remark that in its tournament version *Victoria* uses a combination of threat-space search and proof-number search (Allis *et al.*, 1994). This combination results in finding the winning line in all three positions mentioned. Details are to be found in Allis (1994).

On the efficacy we may conclude that our implementation of threat-space search now and then misses a winning line, which has to be compensated for by starting the proof-number search procedure. For a discussion on efficiency we confine ourselves to the nine positions solved by all program versions. As a main result we note that *Victoria TSS* has searched less nodes in all nine positions than *Polygon* with the 3-ply knowledge rules. This holds also true for the comparison with *Polygon*'s version with the 7-ply knowledge rules, except for position 9.

The extremes in efficiency are found in position 1, with *Polygon 3-ply* searching 3.2 times more nodes, and in position 8, with *Polygon 3-ply* searching 87 times more nodes. For the comparison with *Polygon 7-ply*, the figures range from 0.8 (position 9) to 17.2 (position 7). Considering the number of nodes searched as a measure of efficiency, the competition

is in favor of *Victoria TSS*. Moreover, if we take into account that *Victoria TSS* searches ten times faster since it is not hemmed by a burden of knowledge, we may conclude that threat-space search is a valuable new search technique.

6. Victoria's architecture

The implementation of threat-space search, as described in section 4, has resulted in a module capable of quickly determining whether a winning threat sequence exists (averagely 0.1 CPU-seconds on an IBM RS6000). This module can be used as a first evaluation function. If the evaluation value reads a "win for the attacker" then there is a win. If no winning line has been found by the module, the proof-number search procedure is started. All this means that the threat-space search procedure seen as a heuristic evaluation procedure is admissible (Nilsson, 1980).

Hence, it is now possible to construct a search tree from the initial position using the techniques of threat-space search and proof-number search, which arrives at the same game-theoretical result as a full-width game tree would. We define a node to be internal in the search tree if no winning threat sequence (tree) can be found for the player to move. The main problem to be faced is the large branching factor (averagely over 200), especially when no threat needs to be answered (Van den Herik, 1983, p. 253). Then all defensive moves must be investigated in order to guarantee a correct evaluation of internal nodes

To reduce the branching factor, in positions with Black to move we have applied the null-move heuristic (Beal, 1989). Black may only make moves which, after a null move by White, are followed by a winning threat sequence. Such moves thus constitute hidden threats. This procedure normally reduces the number of possible countermoves to a few dozen.

The selection of this type of hidden threat moves is rather straightforward and can be done by heuristics. *Victoria* assigns points for creating patterns of two or three stones in a row. The *N* moves with the largest number of points are examined to see if they result in a win after a null move by White. After some testing *N*=10 proved to be a reasonable choice.

Since we can be sure that every selected black move constitutes a threat, the vast majority of White's moves can be disregarded. Still, we must make sure not to reject by accident a white move potentially playing an important part in his defense. Therefore, we have created a module deducting from a winning threat sequence the white moves with any defensive value. These are (1) all moves in the winning sequence itself, (2) all moves connected to white

stones on the board, and (3) all moves which come connected to the cost squares in the winning threat sequence. The module generally reduces the number of potential white moves from over 200 to between 10 and 50. Each of the remaining moves is investigated on the fact whether it really refutes the winning sequence (by executing the move, and then performing a threat-space search). When Black has more than one possible winning sequence, the number of defensive white moves usually lowers further. A position in which all defensive moves are thus eliminated is a win for Black. If not all defensive moves are eliminated the search is developed for *each* defensive white move.

The module described above solves to a large extent the problem of the huge branching factor. We have found that only after the first two black moves, and in some variations after the third and fourth black move, Black does not threaten to win after a white null move. In these cases all 200+ moves must be examined. By doing so we have solved the game of Go-Moku (see section 7).

7. Results

Although we have described above how the size of Go-Moku's game tree can be reduced considerably, still a tree of several millions of positions remains to be searched. To accomplish this feat, a number of practical problems must be overcome:

1. the tree must be split in a few hundred subtrees.

2. the results of each calculation (the solution of a subtree) has to be stored in a database;

3. the results of all calculations must be merged into a final tree, containing the solution;

4. the solution in the final tree must be inspected on consistency and completeness.

We remark that inspection of the solution as mentioned under 4 serves two goals: (1) determining the correctness of the white-move selection module (cf. section 6), and (2) establishing that no mistake has been made in merging all hundreds of subtree results.

The calculations were performed in parallel, on 10 SUN SPARCstations 2 of the Vrije Universiteit in Amsterdam. Each of the machines was equipped with 64 or 128 Mbytes of internal memory and a swap space of over 200 Mbytes. Our processes could only run overnight. As a result, some processes not finished at 8 a.m. were killed, and had to be restarted at 6 p.m. Still, over 1000 CPU-hours a week were available for solving Go-Moku.

First, the non-restricted variant of Go-Moku was examined, in which an overline is a winning pattern. The calculations also served as a testing ground for the program, and indeed a few programming errors were discovered and repaired.

Second, variant B (cf. section 2) was examined. Again a testing period was necessary to investigate the effects of overlines in the evaluation function and in the white-move selection module.

Both variants have been solved: in each case a winning tree for Black has been found and stored in a database. Table 5 shows the statistics of both solving procedures. Inspection of the solution tree took approximately 1 CPU day per variant of Go-Moku. The optimal line starts with a human-understandable opening (11 plies). In the middle game, Black and White alternatively create threats, with Black successfully increasing his threat potential. After 35 plies it is all over.

	non-restricted	variant B
CPU time	1.1 million seconds	1.3 million seconds
pos. investigated	5.3 million	6.3 million
solution-tree size	138,790	153,284
solution-tree depth	35	35

Table 5: Statistics on solving Go-Moku.

As is well-known, the Computer Olympiads (Levy and Beal, 1989, 1991; Van den Herik and Allis, 1992) provide an excellent test bed for game-playing programs. At the fourth Computer Olympiad (London, 4-11 August 1992), Victoria entered the Go-Moku tournament, where the competition is played according to the rules of variant B.

When playing Black, Victoria won all its games. It started playing from the database and when out of moves, it found a winning threat sequence using the threat-space search module. *Victoria* also won half of its games playing White. This result sufficed to win the gold medal. The games played at the 4th Olympiad are reproduced in Appendix B.

8. Conclusions

Threat-space search has been developed to model the way human expert Go-Moku players find winning threat sequences. Earlier, in a modest version, it has been successfully applied to Qubic (Allis and Schoo, 1992). Threat-space search finds deep winning lines of play based on threats in averagely 0.1 CPU seconds. The price paid for this success is small: in some cases where a winning line exists, it is not found. To overcome this problem proof-number search is evoked. Threat-space search and proof-number search are applicable to other games as well as to the area of theorem proving. Implemented in the tournament program *Victoria*, they solved the game of Go-Moku: *Victoria* always wins, when playing Black.

Acknowledgements

References

Allis, L.V. (1994). *Games and Artificial Intelligence*. Ph.D. thesis in preparation, Department of Computer Science, University of Limburg, Maastricht, The Netherlands.

Allis, L.V., Meulen, M van der, and Herik, H.J. van den (1994). Proof-Number Search, accepted for publication in *Artificial Intelligence*. Also published in a provisional version as Report CS 91-01, Department of Computer Science, University of Limburg, Maastricht, The Netherlands.

Allis, L.V. and Schoo, P.N.A. (1992). Qubic Solved Again. *Heuristic Programming in Artificial Intelligence 3: the third computer olympiad* (eds. H.J. van den Herik and L.V. Allis), pp. 192-204. Ellis Horwood Ltd., Chichester, England.

Beal, D.F. (1989). Experiments with the Null Move. *Advances in Computer Chess 5* (ed. D.F. Beal), pp. 65-79. Elsevier Science Publishers B.V., Amsterdam, The Netherlands.

Groot, A.D. de (1965). *Thought and Choice in Chess* (ed. G.W. Baylor) Translated from the Dutch version from 1946. Second edition 1978. Mouton Publishers, The Hague.

Herik, H.J. van den (1983). *Computerschaak, Schaakwereld en Kunstmatige Intelligentie*. Ph.D. thesis, Academic Service, The Hague, The Netherlands.

Herik, H.J. van den (1990). *Synthesis of Reliable Plans from Low-Order Knowledge*. Description of SION project no. 612-332-021, The Netherlands Organization for Scientific Research.

Herik, H.J. van den, and Allis, L.V. (eds.) (1992). *Heuristic Programming in Artificial Intelligence 3: the third computer olympiad*. Ellis Horwood Ltd., Chichester, England.

Knuth, D.E. and Moore, R.W. (1975). An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293-326.

Kopec, D. and Bratko, I. (1982). The Bratko-Kopec Experiment: A Comparison of Human and

Computer Performance in Chess. *Advances in Computer Chess 3* (ed. M.R.B. Clarke), pp. 57-72. Pergamon Press, Oxford, England.

Levinson, R.A., Beach, R., Snyder, R., Dayan, T., and Sohn, K. (1992). Adaptive-Predictive Game-Playing Programs. *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 4, pp. 315-337.

Levy, D.N.L., and Beal, D.F. (1989). *Heuristic Programming in Artificial Intelligence: the first computer olympiad*. Ellis Horwood Ltd., Chichester, England.

Levy, D.N.L., and Beal, D.F. (1991). *Heuristic Programming in Artificial Intelligence 2: the second computer olympiad*. Ellis Horwood Ltd., Chichester, England.

Newell, A., and Simon, H.A. (1972). *Human Problem Solving*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Nilsson, N.J. (1980). *Principles of Artificial Intelligence*. Tioga Publishing Company, Palo Alto, Ca.

Sakata, G. and Ikawa, W. (1981). *Five-In-A-Row. Renju*. The Ishi Press, Inc., Tokyo, Japan.

Uiterwijk, J.W.H.M. (1992a). Go-Moku still far from Optimality. *Heuristic Programming in Artificial Intelligence 3: the third computer olympiad* (eds. H.J. van den Herik and L.V. Allis), pp. 47-50. Ellis Horwood Ltd, Chichester, England.

Uiterwijk, J.W.H.M. (1992b). Knowledge and Strategies in Go-Moku. *Heuristic Programming in Artificial Intelligence 3: the third computer olympiad* (eds. H.J. van den Herik and L.V. Allis), pp. 165-179. Ellis Horwood Ltd, Chichester, England.

Uiterwijk, J.W.H.M. (1992c). *Personal Communication*.

Appendix A

Below we provide example lines of the solution to the Go-Moku variants announced in section 2. The sequence of moves is numbered in the diagrams. Diagram 5a contains the longest line of the non-restricted Go-Moku solution and Diagram 5b the longest line of our standard Go-Moku solution.
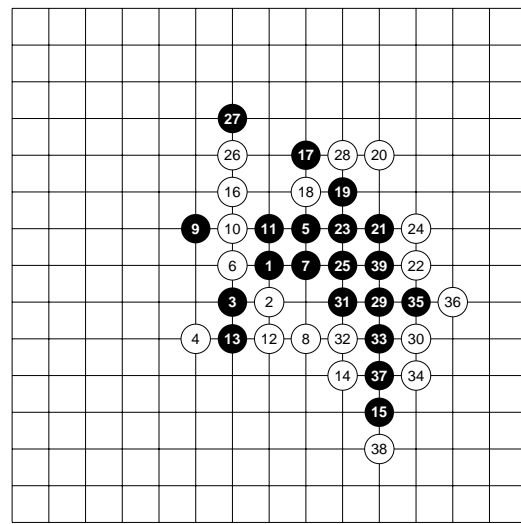

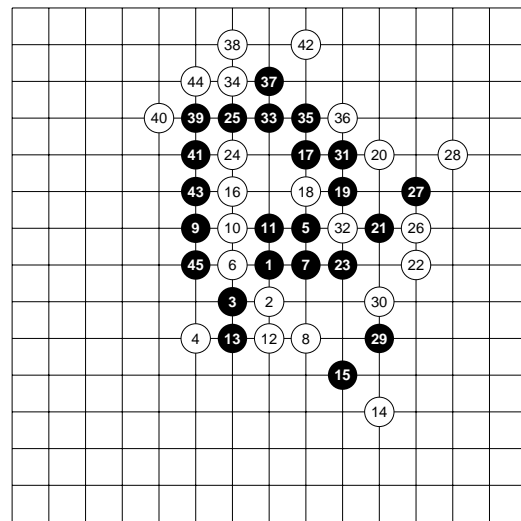**Diagram 5a**: Non-restricted Go-Moku solution.

Diagram 5b: Standard Go-Moku solution.

Appendix B

Below we have reproduced *Victoria*'s gamescores of all eight games played at the 4th Computer Olympiad in London (5-11 August, 1992).
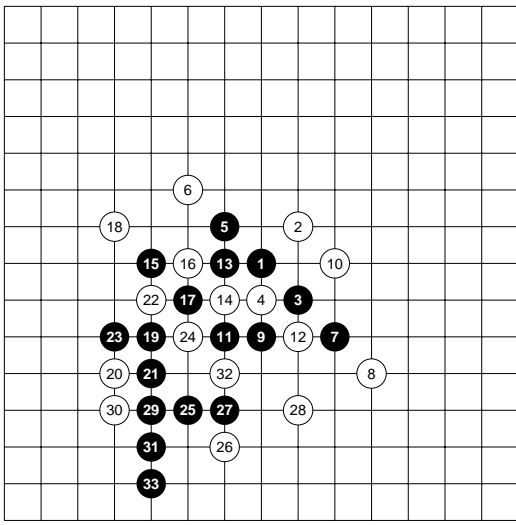
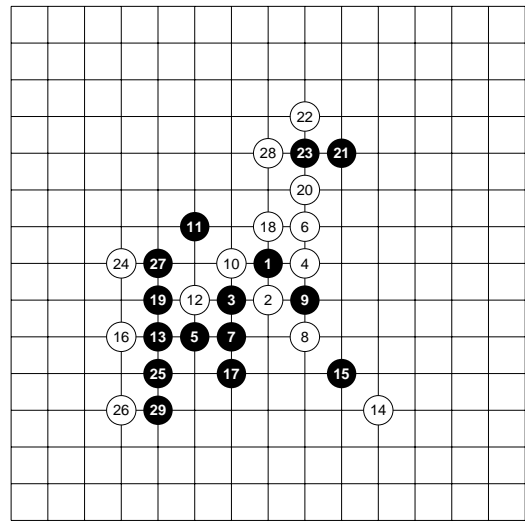**Diagram 6a**: *Victoria* (B) vs. *Neuron* (W)
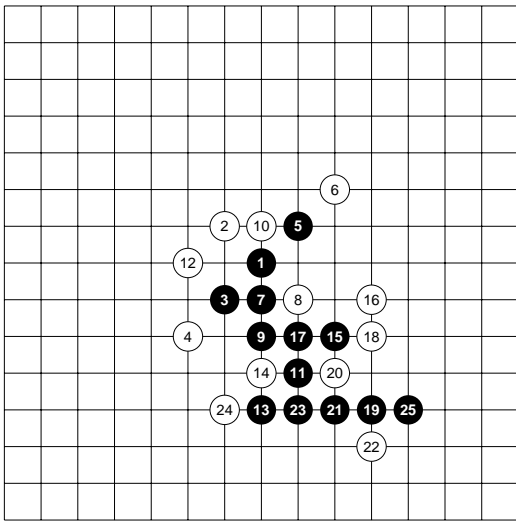
Diagram 6b: *Victoria* (B) vs. *Xokk* (W)

Diagram 6c: *Victoria* (B) vs. *Polygon* (W)

Diagram 6d: *Victoria* (B) vs. *Zero Club* (W)
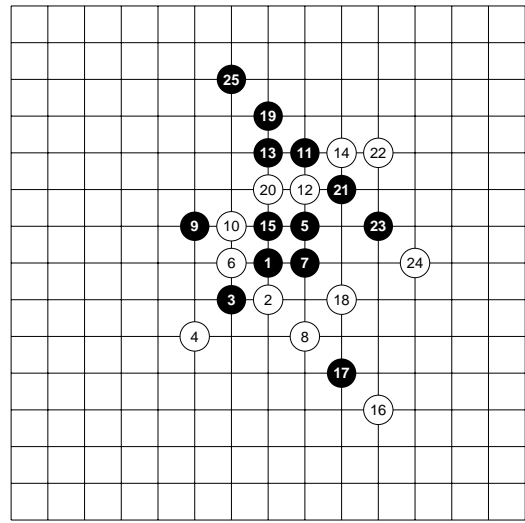
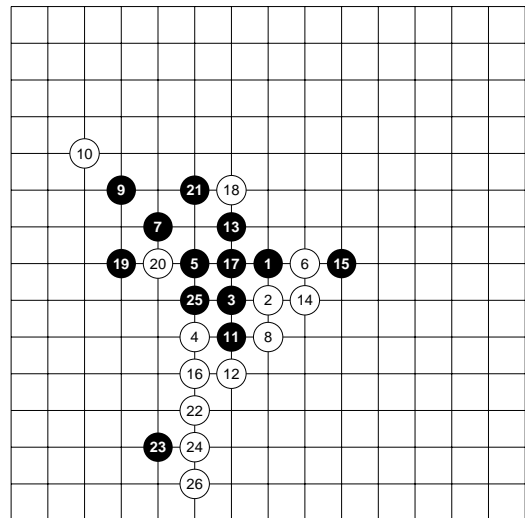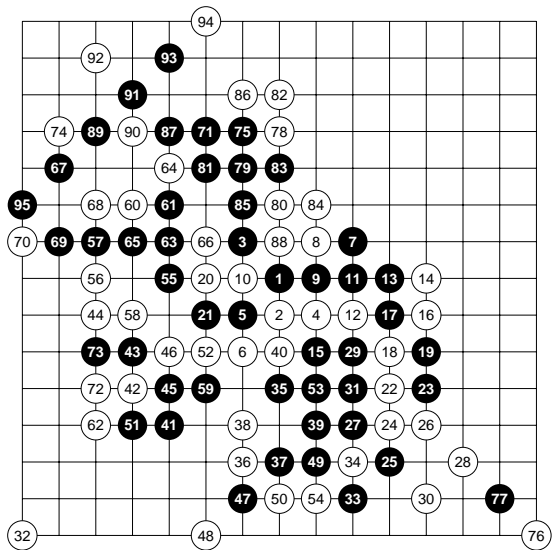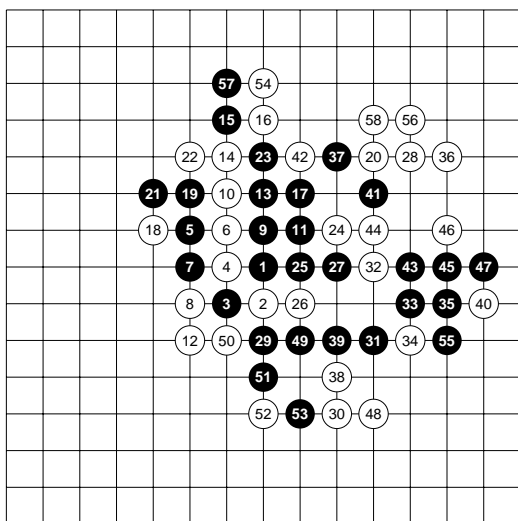**Diagram 6e**: *Neuron* (B) vs. *Victoria* (W)

Diagram 6f: *Xokk* (B) vs. *Victoria* (W)

Diagram 6g: *Polygon* (B) vs. *Victoria* (W)

Diagram 6h: *Zero Club* (B) vs. *Victoria* (W)