A Defensive Strategy Combined with Threat-Space Search in Connect6

Yun-Ching Liu and Shun-Shii Lin
Department of Computer Science and Information Engineering
National Taiwan Normal University, Taipei, Taiwan, R.O.C.
Email: cipherman@gmail.com, linss@csie.ntnu.edu.tw

***Abstract*—The study of k-in-a-row games has produced a number of interesting results, and one of its sub-category Connect6 proposed in 2005 has been of particular interest. Since 2006, Connect6 has been included as one of the major competition in the ICGA Computer Olympiad, and is gaining more popularity every year.**

In this paper, we briefly review current methods applied in Connect6 and related results. A defensive strategy is introduced along with a more strategically sensitive evaluation scheme. Threat-space search is an important algorithm applied in Connect6, some techniques for gaining more efficiency and accuracy will be introduced. The integration of the defensive strategy and threat-space search will also be investigated.

The combination of the defensive strategy and threat-space search is proved to be effective, and is able to compete with other top Connect6 programs. The program Kagami, which was implemented with these methods, won the fourth place in the 14th Computer Olympiads.

***Index Terms*—Connect6, k-in-a-row, Threat-space Search, Aritificial Intelligence**

I. Introduction

Games are ideal domains for exploring the capabilities of artificial intelligence (AI), although chess has long time been called the Drosophila of AI[9], the research of other games has also provided fruitful results.

The study of k-in-a-row games has produced a number of interesting results [11][4][10][13], and has also introduced new AI techniques [3][2]. Variants of k-in-a-row games have also been investigated, such as k-in-row games played on higher-dimensional boards[6][5][1], and the $Connect(m, n, k, p, q)$ family of k-in-a-row games, where two players place p stones in each turn on an $m \times n$ board except for that the first player places q stones for the first move, and the player who gets k consecutive stones of his own first wins[12].

One member of the k-in-a-row games, Connect6, proposed by Wu and Huang [11] in 2005, has been of particular interest. Connect6 is essentially a $Connect(19, 19, 6, 2, 1)$ game. Since 2006, Connect6 has been included as one of the major competition in the ICGA Computer Olympiad, and is gaining more popularity every year.

Many artificial intelligence methods applied in Connect6 are largely under the influence of the techniques used in solving Go-Moku[3][2], where threat-space search was proposed and applied. Wu and Huang proposed a Connect6 version of threat-space search, and generalized it to the family of k-in-a-row games[12][7]. It became the foundation of today's Connect6 programs, and much effort has been made to increase the speed and accuracy. The stage between the start of the game and a winning sequence found by threat-space search is where the evaluation heuristics have more influence. Allis used proof number search for this stage in Go-Moku, while in Connect6 most programs used $\alpha\beta$-search.

Connect6 is a relatively new board game, and the development of the evaluation heuristics is still in its infancy, therefore there are still many unknown territories to explore.

We will present a new evaluation scheme which is based on the observation of how a stone imposes its influence on the board, and it also has the potential to be extended to other k-in-a-row games and their variants. A defensive strategy will also be presented, although it is just a simple greedy algorithm based on the proposed evaluation scheme without using any kind of search algorithm, it already can achieve admirable results against some of today's top Connect6 programs. When the defensive strategy is combined with threat-space search, the

program can compete at the same level of the state-of-the-art programs. The program Kagami, which was based on these methods with the addition of some heuristics such as the null-move heuristic, won the fourth place in the 14th Computer Olympiad.

II. Evaluation Scheme

We will introduce a new evaluation scheme for Connect6 based on the observation of the influence of a stone on the board. The scheme is argued to be more strategically sensitive than traditional approaches.

A. Influence of a Stone

Apart from the first move, two stones are placed on the board with each move, and these stones change the development and nature of the previous position. What does the presence of a new stone alter?

A potential line is a line with a length of 6 on the board, with no stones or only a single kind of stone on it. Thus, it is a potential candidate to connect six for one of the players. If there are two kinds of stones on the line, then it can be sure that a winning six will not occur on it. We will regard a line which contains six consecutive stones to win the game as a winning line.

A straight forward and tactical point of view is that a stone played on the board increases the chance of connecting six for one side, or decreases the chance of winning for the opponent. But what is the scale or range of its influence? Instead of viewing the board as a collection of points, we regard it as a collection of potential lines. Therefore, the presence of a stone increases the probability of a potential line, which contains the stone, to become a winning line.

Hence, a stone's influence area should be the area that consists of all the potential lines which contain the stone. The largest possible area of influence is made up of four lines (horizontal, vertical, and two diagonals) of length 11, with the stone in the center as depicted in Figure 1.

Other points which are not included in the area are by no means not important at all. Rather, they differ from the included points by meaning. The included points are under more direct influence of the stone, thus may have more tactical meaning, whereas the ones that are not included are less directly related to the stone, and may only retain some strategic importance.

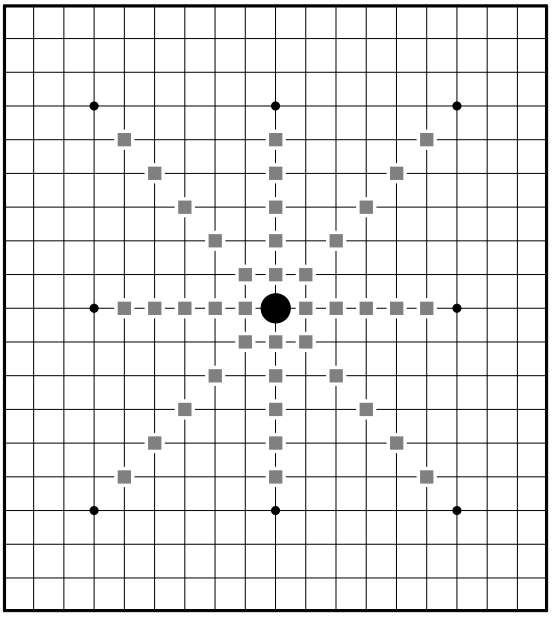


Fig. 1. Influence of a stone

Connectivity is another major issue in Connect6, since if one's stones are sparsely spread across the board, the chance of connecting six is low. If two stones are close to each other, there are less points between them for the opponent to cut their connection. Thus, the relation between two stones decreases with increasing distance, and this relation is in a sense formed by the influences of the two stones imposing on each other. It is therefore natural to imply that the influence of a single stone decreases with increasing distance.

The influence of a stone should stop when the border or an opponent's stone is encountered, since it can't make any connections with any stones beyond them. Therefore, the length of the lines in the statement of the influence area should be refined to lines with a length of 11 or less, as shown in Figure 2.

We will not try to take the points that are not in the influence area into consideration, since we believe that their strategic values may vary for different positions, and a perfect model may not exist. Even if it does exist, it may introduce unnecessary complexities.

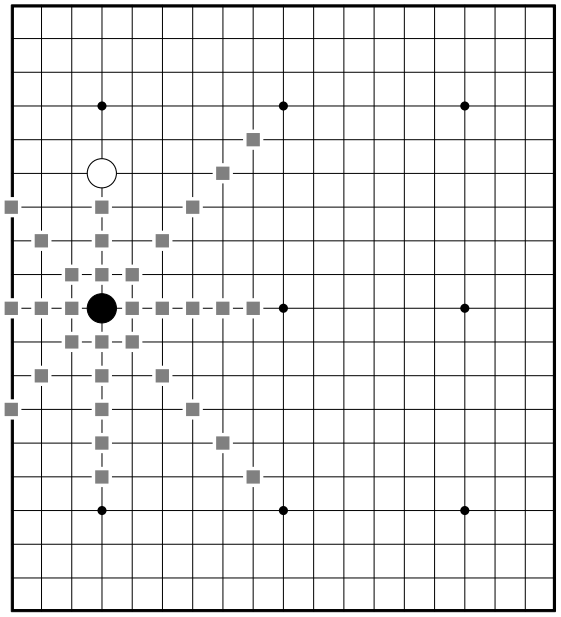


Fig. 2. The influence area encountering borders or opponent stones

B. The Evaluation Scheme

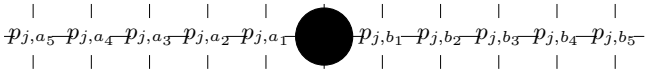


Fig. 3. Evaluation of a half-move

Suppose that the opponent is playing black, then the evaluation score for a black stone or a half-move is given by

$$E = d_i \sum_{j=1}^{4} (\prod_{k=1}^{5} p_{j,a_k} \prod_{k=1}^{5} p_{j,b_k}).$$

E is the evaluation score of the half-move, reflecting the influence it has. i is the number of directions that have no opponent stones, and d_i is the weighted value in that respect. The value j corresponds to the index of directions to be considered, so there are 4 directions (1 horizontal, 1 vertical, and 2 diagonal). p_{j,a_k} and p_{j,b_k}, where $1 \leq k \leq 5$, are the states of the points, which correspond to the points marked in Figure 3 on each respective line.

The values of p_{j,a_k} and p_{j,b_k} are determined in an orderly decreasing fashion from near to far, i.e. from p_{j,a_1} to p_{j,a_5} and from p_{j,b_1} to p_{j,b_5}. If the point is an empty point, the value ϵ is given, and if the point is occupied by a black stone, then a corresponding weight w_k is given. The values of w_k decrease with increasing k. Once the border or a white stone is encountered at a certain k, the remaining values $w_k, w_{k+1}, ..., w_5$ are set to 1.

Note that the values on the same line are combined by using multiplication and the four directions are combined by using addition. This reflects the fact that connectivity has higher priority over directional liberty. Although in some cases, directional liberty may have higher priority, especially in earlier stage of the game, these cases can be addressed by applying the degree parameter d_i.

A fundamental relation needs to be preserved: a half-move which contains occupied stones in its influence area should not be overpassed by a half-move which only has empty points in its influence area. Please see Figure 4 for a depiction. This concern may be crucial since empty points also have a weight.

A simple solution would be to let a portion of the weight of a stone be larger than the value of a line that contains only empty points. Therefore, the relation

$$\epsilon^{10} \leq w_l$$

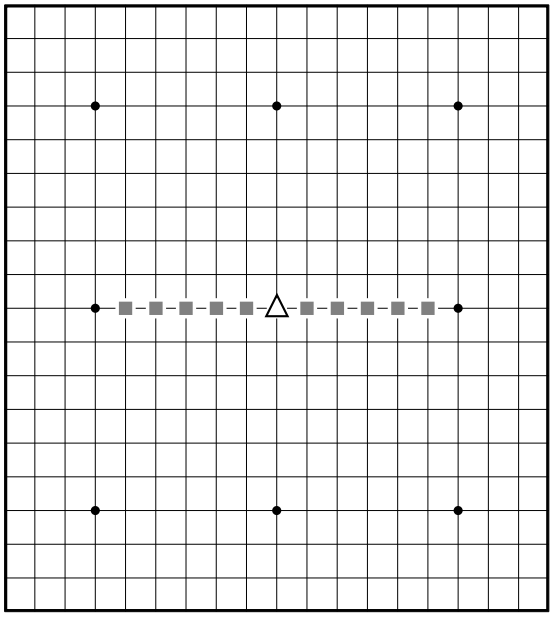
must be held, where $l \leq n \leq 5$. w_n is the weight that is equal to the value of the line with only empty points.

We decided that the weight of an empty point $\epsilon = 2$, then we use it as a base to determine the values of w_k. From experiments and experiences, we determined that $n = 3$, and $w_1 = 2^{12}$, $w_2 = 2^{11}$, $w_3 = 2^{10}$, $w_4 = 2^9$, $w_5 = 2^8$.

The values of degree weights are determined and tuned purely from experiments, which are $d_1 = 1.0$, $d_2 = 1.00000181862$, $d_3 = 1.00000363725$, $d_4 = 1.00000726562$. The values are fairly small, due to the exponential figures of the empty point weight and stone weights, and due to their nature of being mostly adjustments for some positions.

An example is shown in Figure 5, the evaluation value of triangled marked stone is determined by

$$E = d_3 \times [(\epsilon \times \epsilon \times 1 \times 1 \times 1 \times \epsilon \times w_2 \times w_3 \times \epsilon \times \epsilon)$$
$$+(\epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon)$$
$$+(\epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon)$$



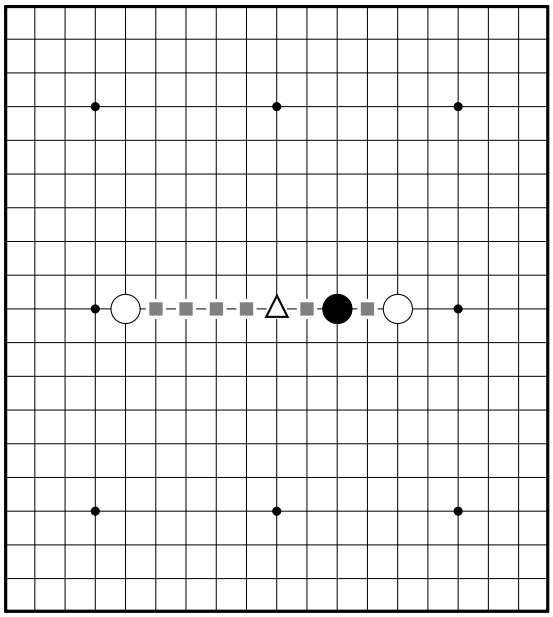


Fig. 4. Evaluation of the triangle position in the bottom should be higher than that in the top.

$$+(\epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon \times \epsilon)]$$

$$= 1.00000363725 \times [((1 \times 1 \times 1 \times 2 \times 2)$$

$$\times(2 \times 2^{11} \times 2^{10} \times 2 \times 2)) + 2^{10} + 2^{10} + 2^{10}].$$

There is one opponent stone in the horizontal direction, thus the degree parameter d_3 is applied. Apart from the horizontal direction, there are no stones in the vertical and diagonal directions, therefore their

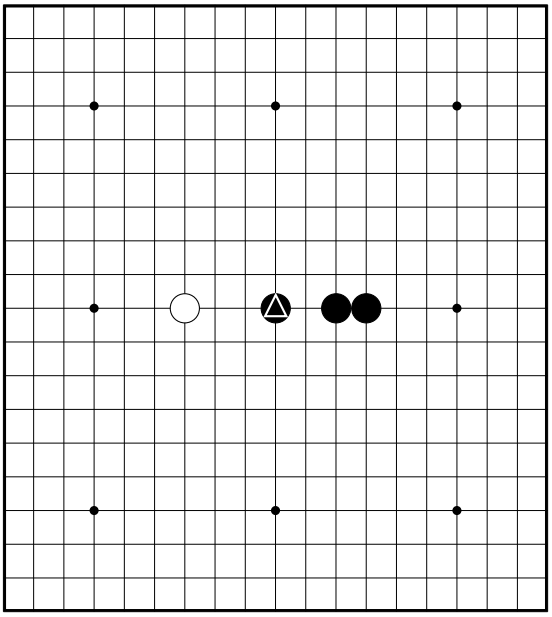


Fig. 5. Example of evaluation

values are the product of 10 empty point weights ϵ, that is 2^{10}. The value determined by the horizontal direction is thus $(1 \times 1 \times 1 \times 2 \times 2) \times (2 \times 2^{11} \times 2^{10} \times 2 \times 2)$.

C. Comparison to Current Evaluation Techniques

The evaluation schemes used in many programs are mostly pattern-based, for different configurations of stones on a line, a score is assigned[7] [8]. Then the scores of the four directions are combined (mostly by addition) to make up the evaluation score.

For example, if black places a stone at the triangle positions in the patterns A, B, and C in Figure 6, the patterns they create are all usually classified as a Live4 pattern, and its main characteristic is that white needs two stones to defend the threat. Hence, the half-moves at the triangle positions have the same value in most of current program's evaluations.

But with our proposed evaluation scheme, the three half-moves have different scores. Pattern A has the highest score, pattern B is the next, and pattern C is the lowest. If the half-moves have been played, they would create the same amount of threat. But due to the different number of empty points on the examined directions, the evaluation values

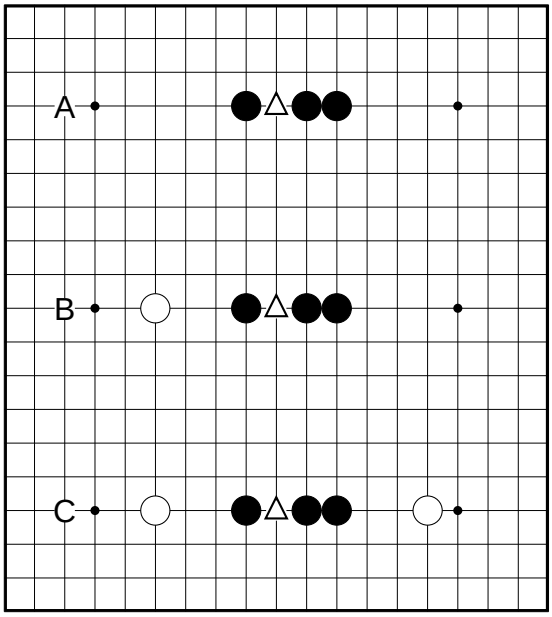


Fig. 6. Comparison of different evaluation scheme

vary since empty points also have a weight. And this reflects the fact that although they have the same tactical meaning (threats), they are distinct in strategic context, namely liberty or space.

Therefore, the proposed evaluation scheme is much more delicate, and may distinguish the strategic importance of the same pattern. It is more strategically sensitive than the traditional approaches, but retains the same tactical sensitivity at the same time.

The new scheme is also easy to be adapted to other k-in-a-row games, while the traditional methods may need more game-specific heuristics.

III. A Defensive Strategy for Connect6

We will introduce our defensive startegy which utilizes the evaluation scheme presented in the previous section, and present our experimental results.

A. The Defensive Strategy

We will make an assumption that the opponent has a purpose with every moves. If the opponent's intention is to make some abstract connections to other stones to produce an attacking pattern, it would be best to cut off these connections as soon as possible. That is by "following" the opponent's move to wherever he goes, and cut off potentially dangerous connections, one can minimize the chance that the opponent can make an attack. Of course, if the opponent's purpose is to defend, we may waste some stones on unnecessary defense if we follow this strategy. But then again, our purpose is to make a solid defense, thus the waste of stones is tolerable.

So we only consider the half-moves (or points) that are in the 5×5 square with the last two stones that the opponent played in the center as candidate half-moves. Therefore, at most 48 points or half-moves are considered. Figure 7 shows a position with white to move, and candidate half-moves are marked by filled squares.

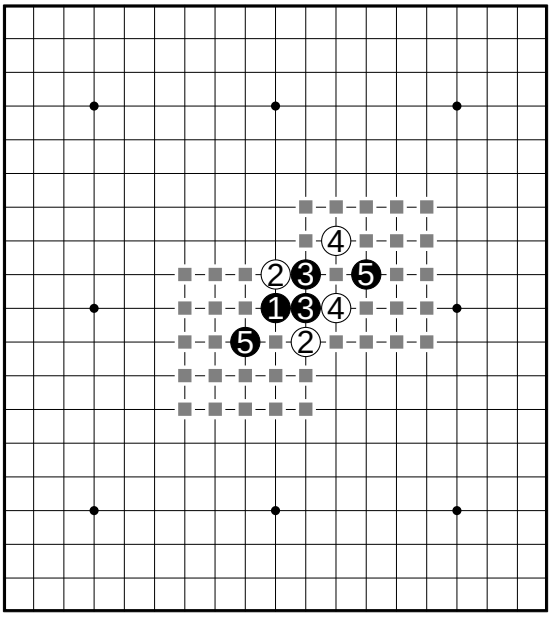


Fig. 7. Candidate half-moves (white to move)

The defensive strategy consists of two phases. First, it generates a half-move list and sorts the entries according to the values which are given by the evaluation scheme proposed earlier. Then it plays the half-move with the highest value, and do the same for the second half-move.

The overview of the proposed strategy is given in Figure 8. Note that it is only a simple greedy method based on the proposed evaluation, and no search technique of any kind is applied.

B. Experimental Results

To test how effective our defensive strategy is, we use it to play against two famous Connect6

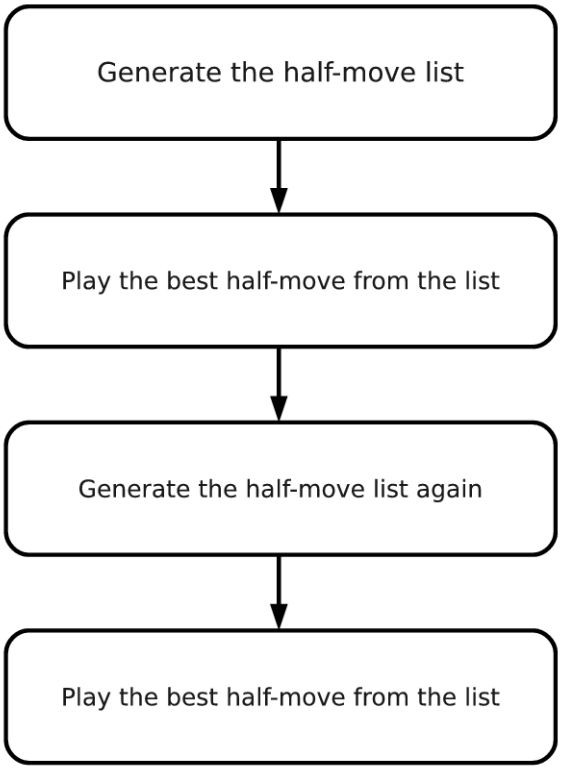


Fig. 8. Flowchart of a defensive strategy

programs:

- **NCTU6**: NCTU6 is a program developed by the Internet Application Laboratory led by Professor I-Chen Wu at the National Chiao Tung University. It won a gold medal in the 11th Computer Olympiad and another gold medal in the 13th Computer Olympiad. The version we did our experiments on is the 2006 public release version 1.0, with playing strength set to level 3 (the maximum strength is level 5).
- **X6**: X6 was developed by Shih-Yuan Liou and Professor Shi-Jim Yen at the Artificial Intelligence Lab. of the National Dong Hwa University. It won a silver medal in the 11th Computer Olympiad and a gold medal in the 12th Computer Olympiad. Experiments are conducted with version 1.4.0f, with playing strength set to 9 (Kill-Defend Search depth was set to 11, and 100 seconds to timeout)

To get the best performance possible of each program, we thought it would be better to run the programs on their original, native environments. However, the machines used are basically with similar computation power in order to avoid the unfair competition. Our program was run on a machine with AMD64 3000+ and 1GB RAM under Ubuntu Linux 9.04, kernel version 2.26.1. The opponent programs were run on a machine with Intel Core2 Duo 1.66GHz and 1 GB RAM under Windows XP Professional Service Pack 2. Ten games are played with each program. Our program plays black in the first 5 games and plays white in the last 5 games. The results are listed in Table I and II. In the tables, D, W, and L, means we draw, win and lose, respectively. Game length is the number of moves that are played. Although there are still empty points available on the board, we declare the game drawn around move 120, since the space left on the board is insufficient for either side to get a six.

TABLE I
RESULTS OF OUR DEFENSIVE STRATEGY AGAINST NCTU6 (LEVEL3)

Game	1	2	3	4	5	6	7	8	9	10
Result	D	D	D	D	D	D	D	D	D	D
Game Length	121	120	119	124	120	120	123	121	120	123

TABLE II
RESULTS OF OUR DEFENSIVE STRATEGY AGAINST X6

Game	1	2	3	4	5	6	7	8	9	10
Result	L	D	D	D	L	L	D	D	D	L
Game length	31	121	123	131	47	18	118	128	120	34

We can see that our program can draw perfectly against NCTU6 (level3). Our program can draw 60% of the games against X6 in its full power without using any kind of search, showing that our defensive strategy is very effective. However, it still loses 40% of the games, so there is still room for further improvement.

IV. COMBINING WITH THREAT-SPACE SEARCH

In order to complement the weakness of the proposed defensive startegy, threat-space search is applied as a verifier to check whether the decided defensive move is valid. If not, alternative moves, such as the second best move and so on, are tried until an effective move is found or a certain number of alternative moves is reached. This combination is proved to be able to effectively improve the performance of the defensive strategy.

A. Threat-space Search

Threat-space search was first proposed by Allis[2], and it was used as an evaluation function accompanying proof number search to solve Go-Moku. The main idea of threat-space search is similar to that of quiescence search, in which only forced moves are extended and explored.

In Connect6, if a player plays a move which threatens a connect six on the next move, then this move is called a threat move. For example, in Figure 9, the black player played the 2 stones marked with a, and is threatening to win on the next move by playing b_1 b_2, b_2 b_3, or b_3 b_4. Therefore, the move is a threat move, and then the opponent has to defend against the threat, otherwise he will lose the game.

Fig. 9. Example of threat move

There are different types of threats which are mainly classified by the number of stones needed for the defender to resolve the threat. Hence a single threat is a threat that the defender can resolve it by using one stone, and two threats requires two stones. Since a player can only place two stones on a single move, the defender cannot defend against three or more threats. In some positions, one can play a series of threats that leads to three or more threats, and thus win the game. Threat-space search is a search algorithm that only explores threat moves with the goal of finding such a series. If such a series is found, one can claim victory.

Although the branching factor is significantly lower than the usual uniform search, it is still undesirable to explore every possible defensive moves. Therefore, a way to simplify and lower the complexity of threat-space search is not to consider the defensive moves separately, but to consider them all at the once.

Conservative defense is to play all defensive moves at the same time. An example is shown in Figure 10. The black stones form a Live4 pattern, and it only needs two stones to defend from the threat, a combination of stone a and stone b, or two stone b.

Conservative defense effectively transforms threat-space search into a single agent search.

Fig. 10. Conservative defense

Since for every threat pattern, all possible defensive moves are played at the same time, therefore a pattern combined with its respective defensive moves can be combined into a single pre-defined pattern.

However, by applying conservative defense, the defense side is presumed to play all possible defenses at the same time, and thus more than two stones could be placed on each move. In Figure 10, the white's conservative defense plays four stones. Therefore, the set of solutions found by the threat-space search that applied conservative defense can only be a subset of the set of true solutions, since the extra defense stones can cause an early search failure.

In some variations, the forcing series consists of the class of two threats mixed with the class of one theat. But we only consider the series that contains two or more threats for the sake of simplicity. Hence the move generation procedure only generates the moves that can create two or more threats.

A pattern table is applied in our implementation of threat-space search in order to save computing time. Each entry in the table contains a key and the number of threats. The defensive moves are not saved in the table, and are computed during run-time. All possible configurations of stones on a line of length 11 are pre-processed, and the numbers of threats of these patterns are stored.

Full hashing is used, the hash function maps each configuration into a base-3 number ranging from 0 to $3^{11} - 1$ acting as a hash key. The weights of the corresponding positions on the line are shown in Figure 11.

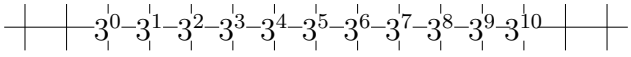


Fig. 11. Positional weights of hash function

If a point is empty, its value is 0. If a point has a white stone, its value is 1, and if a point has a black stone, its value is 2. The point value is multiplied by its respective positional weight and

then all weighted values are summed up to get the hash key. Borders are treated as the opponent's stones.

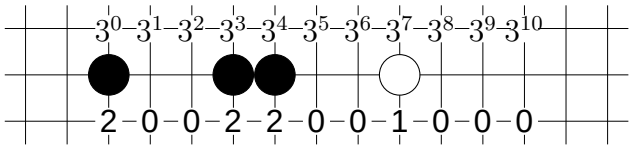


Fig. 12. Example of hash key

An example is given in Figure 12. The positional weights are given above the pattern, and the respective weights of the stones or empty point are shown below. Hence the hash key is calculated as follows:

$$HashKey = 2 \times 3^0 + 2 \times 3^3 + 2 \times 3^4 + 1 \times 3^7.$$

The pattern table is applied to retrieve the number of threats a stone can create in one direction. Since two stones are played in a single move, possible errors may occur when the two stones are placed on a single line, for they both may "contribute" to the same threat pattern. Thus the threat may be erroneously counted twice.

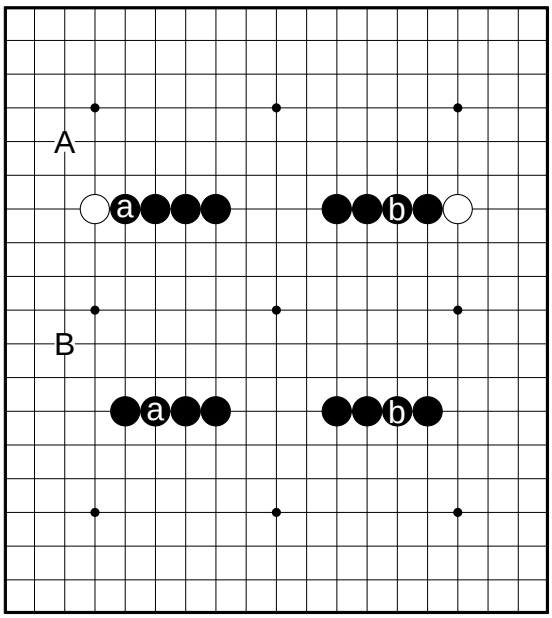


Fig. 13. Errors when two stones align

An example is pattern A in Figure 13, for the stones a and b they both retrieve a threat count of 1, therefore a move consists of stones a and b will be wrongly interpreted as a move that creates two threats if we only simply add up the numbers of threats. Pattern B is another example, which the pattern will be interpreted as a four threats.

A simple scheme may resolve the problem:

1) If both stones are not on the same line, simply add the two numbers of threats up.
2) If both stones are on the same line, and there are opposing stones between them, add the two numbers of threats together.
3) If both stones are on the same line, but without opposing stones between them, then
   a) take only half of their threat sum into account, if their distance is less than or equal to 6.
   b) subtract one from the sum of their threats, if their distance is between 7 and 11, inclusive.
   c) take the sum of their threats, if their distance is 12 or more.

If both stones are not on the same line or there are opposing stones between them, the threats they create are independent, hence the sum of their threats are the threats created by this full move. The correctness of the sum of threats will only be affected if it will lead to a wrong interpretation of one threat and two threats. If the sum of the threats is greater than or equal to 3, like pattern B in Figure 13, it won't matter if it is wrong, since this kind of threat will win the game. Therefore, we only need to deal with the patterns which the sum of the threats is 2, because if it is wrongly computed as a two threats, and in reality it is a single threat, the opponent actually won't be forced to move in defense, and thus it may mislead the search.

Patterns A, B, and C in Figure 14 are examples of the above 3 cases (a), (b), and (c). Suppose the move consists of the stones a and b. In pattern A, the distance between a and b is 2, which is less than 6, and from the pattern table, both of them have two threats. By the correction scheme, only half of their threat sum is set to be the pattern's threat number, that is 2. For pattern B, the distance between a and b is 9, which is between 7 and 11, and from the table we can get that both of them are single threat, thus the sum of threats is 2. By the correction scheme, we need to subtract 1 from

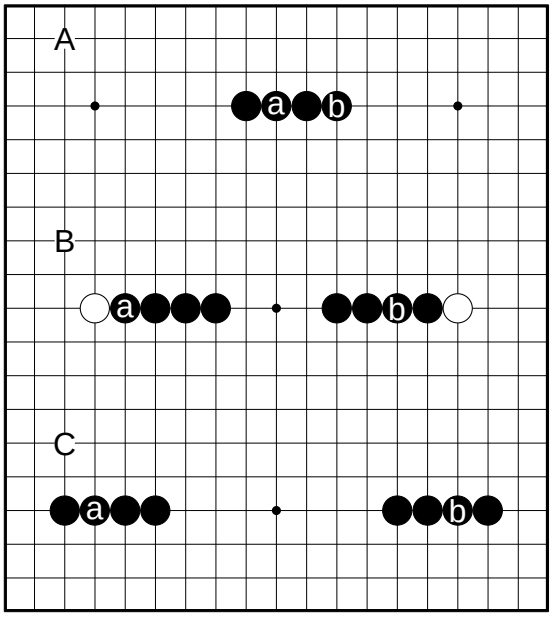


Fig. 14. Example patterns for the correction scheme

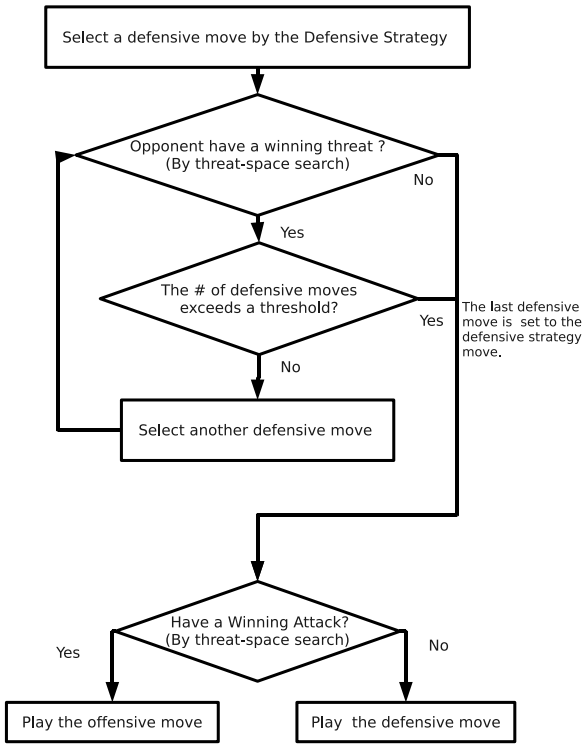


Fig. 15. Defensive strategy combined with threat-space search

the sum, and thus the pattern's threat number is 1. Finally, the distance between a and b in pattern C is 12, and their individual numbers of threats are 2, making the threat sum 4. By the correction scheme, the threat sum is the pattern's threat number, which is 4.

B. Combining with the Defensive Strategy

There is still a certain percentage of the positions that the static defensive strategy proposed in the previous section will fail. No matter how fine the parameters are tuned, due to inaccuracies in the model or any other reason, these kinds of positions may always exist.

Another drawback of only applying the defensive strategy is that, unless the opponent blunders badly, one can never win a game, even though there may exist a winning move.

Therefore, to address these problems, the combination of the defensive strategy and a threat-space search is proposed as a solution. Please see Figure 15 for a depiction.

A defensive move is always derived according to the defensive strategy, and it is verified by the threat-space search. If the verification fails, and the opponent has an immediate threat to win, another defensive move is chosen and verified again until it can defend the opponent's threat or the number of alternative moves exceeds a threshold value. If no effective defensive move can be found, then the last defensive move is set to be the move decided by the defensive strategy.

Finally an offensive threat-space search is to be carried out. If there is a winning move, then the winning move is played, else the defensive move is played. Note that we still don't attempt to create any attacking opportunities or winning threats.

The only situation to attack is that a winning sequence is found by the threat-space search, whereas victory is sure to come. The threat-space search mostly acts as a verifier to make up for the weakness of the static defensive strategy.

The defensive moves are ordered in decreasing order according to the scores given by the evaluation scheme given in the previous section, and at most 50 defensive moves are considered.

Of course this is not an optimal arrangement of the two modules, since it would be better to do the winning-attack search first. But before starting the search for a winning attack sequence, a check is needed to be performed first to make sure

that the opponent doesn't have a winning threat in the current position. This essentially splits the defensive part of the architecture into two parts, with the attacking module in the middle, and thus produces some complications. Since our purpose is to experiment the validity of the proposed strategy, and to verify the performance enhancement when it is combined with a threat-space search, we will avoid such complications and stick with this simple architecture shown in Figure 15.

C. Experimental Results

Using the ideas mentioned in this section, we developed a program to play against the following two programs:

- **X6**: The program was developed by Shih-Yuan Liou and Professor Shi-Jim Yen. It was introduced in the previous section. Experiments are conducted with version 1.4.0f, with playing strength set to 9 (Kill-Defend Search depth was set to 11, and 100 seconds to timeout)
- **MeinStein**: The program was written by Theo van der Storm. Mr. van der Storm passed away in January 2009, and it was subsequently maintained by Jan Krabbenbos. It won a silver medal in the 12th Computer Olympiad, and another in the 14th Computer Olympiad. After the 14th Computer Olympiad, the source code was released to the public domain in memory of Mr. van der Storm. The program was written in Java, and incorporated techniques such as $\alpha\beta$-search and quiescence search.

Same reason as the previous experiment, we run the programs on its native enviroment. Our program was run on a machine with AMD64 3000+ and 1GB of RAM under Ubuntu Linux 9.04, kernel version 2.26.1. The opponent programs were run on a machine with Intel Core2 Duo 1.66GHz and 1 GB RAM under Windows XP Professional Service Pack 2. Ten games were played with X6, 5 with black and 5 with white. Ten games were played against MeinStein also, but against five different settings, with both colors against each setting.

The results are listed in Table III and IV, where the Re-search entry is the number of defensive moves in the game that fail to defend and need to do another full defensive search. The maximum number of alternative defensive moves tried by the full defensive search is 50. The alternative defensive move list is sorted according to the evaluation score given by the evaluation scheme presented previously. In the result entry, D, W, and L, means we draw, win and lose, respectively.

TABLE III
RESULTS OF OUR DEFENSIVE STRATEGY COMBINED WITH THREAT-SPACE SEARCH AGAINST X6

Game	1	2	3	4	5	6	7	8	9	10
Result	D	D	D	D	D	D	D	D	D	D
Re-Search	0	0	0	1	0	0	0	1	2	0
Game Length	121	119	123	121	119	123	124	126	124	120

TABLE IV
RESULTS OF OUR DEFENSIVE STRATEGY COMBINED WITH THREAT-SPACE SEARCH AGAINST MEINSTEIN

Game	1	2	3	4	5	6	7	8	9	10
Result	W	L	W	L	L	W	W	W	L	W
search depth	5	5	4	4	6	6	5	5	5	5
quiescence	1	1	0	0	1	1	0	0	1	1
cutoff time	70	70	70	70	70	70	70	70	80	80
Re-Search	3	1	6	1	1	2	2	3	1	3
Game Length	52	14	51	58	16	63	81	54	16	54

It can be observed that our program can now perfectly draw against X6. In Table III, only three out of ten games needs to do a re-search.

In Table IV, the search depth row specifies the $\alpha\beta$-search depth, the quiescence specifies the depth of quiscience search, and the measure of cutoff time is in seconds of MeinStein. Our program won 60% of the games, while MeinStein only won 40%, showing that our program is slightly superior. Note that almost every game needs a re-search.

Therefore, from these two experiments, it can be seen that our defensive strategy can be effectively enhanced by combining with threat-space search, and this combination is able to compete with today's top programs.

V. CONCLUSION

The Computer Olympiad is a multi-game event, and all the participants are computer programs. The Olympiad was proposed by David Levy, and he also organised the first Olympiad in London in 1989. Connect6 became a tournament item in 2006. The 14th Computer Olympiad was held in Pamplona, Spain, May 2009. In the Connect6 tournament each

program must complete its moves for one game in 30 minutes.

TABLE V
14TH COMPUTER OLYMPIAD CONNECT6 TOURNAMENT RESULTS

Standing	Program
1	Bit
2	MeinStein
3	Bit2
4	Kagami
5	Kavalan
6	Nomi6

Kagami, developed by the author, is a program based on the proposed defensive strategy. Threat-space search and a simple null-move scheme are also integrated. Kagami entered the 14th Olympiad with the purpose of testing the effectiveness of these techniques in tournament situations.

As shown in Table V, six programs attended the tournament of Connect6 and Kagami finished fourth. Kagami is significantly faster than most of the other programs, and even manages to draw against the silver medalist MeinStein and scores a win from the bronze medalist Bit2.

Connect6 is a relatively new game, and there is still a lot to be investigated and discovered. It has a huge complexity comparable to Go, making it interesting and challenging. Although techniques inspired from other games such as Go-Moku, or some standard techniques such as $\alpha\beta$-search work well in Connect6, there is still a large room for innovation and improvement especially in the realms of strategy.

The proposed evaluation scheme is much more strategically sensitive compared to traditional evaluation methods, therefore is able to classify the value of a move into more detailed hierarchies. It is also general enough to extend to other k-in-a-row games. The defensive strategy based on the new evaluation scheme uses only a greedy algorithm, and can achieve admirable results against formidable opponents. Combining the defensive strategy with threat-space search can effectively complement its weakness, and is able to compete with today's top Connect6 programs.

VI. Acknowledgements

This research was supported in part by a grant NSC97-2221-E-003-008 from National Science Council, Taiwan, R.O.C.

REFERENCES

[1] L.V. Allis and P.N.A. Schoo, "Qubic solved again," Heruistic Programming in Artificial Intelligence 3: The Third Computer Olympiad, pages 192-204, 1992.

[2] L.V. Allis, "Searching for solutions in games and artificial intelligence," PhD thesis, University of Limburg, Maastricht, 1994.

[3] L.V. Allis, H.J. van den Herik, and M.P.H Hutjens, "Go-moku solved by new search techniques," Computational Intelligence, 12:7-23, 1996.

[4] E.R. Berlekamp, J.H. Conway, and R.K. Guy, "Winning ways for your mathematical plays, Volume 2," Academic Press, 1982.

[5] S.W. Golomb and A. W. Hales, "Hypercube tic-tac-toe," More Games of No Chance, 42:167-180, 2002.

[6] A.W. Hales and R.I. Jewett, "Regularity and positional games," Transactions of the American Mathematical Society, 106:222-229, 1963.

[7] D.-Y. Huang, "The study of artificial intelligence programming for gobang-like games," Masters thesis, National Chiao Tung University, June 2005.

[8] S.-Y. Liou, "Design and implementation of computer connective 6 program X6," Masters thesis, National Dong Hwa University, July 2006.

[9] J. McCarthy, "Chess as the drosophila of A.I.," Computers, Chess and Cognition, pages 227-237, 1990.

[10] J.W.H.M. Uiterwijk and H.J. van den Herik, "The advantage of the initiative," Information Sciences, 122(1):43-58, 2000.

[11] H.J. van den Herik, J.W.H.M. Uiterwijk, and J.V. Rijswijck. "Games solved: Now and in the future," Artificial Intelligence, 134:277-311, 2002.

[12] I-C. Wu and D.-Y. Huang, "A new family of k-in-a-row games," In the 11th Advances in Computer Games Conference (ACG11), Taipei, Taiwan, September 2005.

[13] T.G.L. Zetters. "Problem S.10 proposed by R.K. Guy and J.L. Selfridge," Amer. Math. Monthly, 86, solution 87(1980):575-576, 1979.