A New Family of k-in-a-row Games

I-Chen Wu and Dei-Yen Huang

Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan
{icwu, teyen}@csie.nctu.edu.tw

Abstract. First, this paper introduces a new family of k-in-a-row games, *Connect*(m, n, k, p, q). In Connect(m, n, k, p, q), two players alternately place p stones on an $m \times n$ board in each turn except for that the first player places q stones for the first move. The player who gets k consecutive stones of her own first wins. The traditional game five-in-a-row, also called Go-Moku, in the free style is Connect(15,15,5,1,1). For simplicity, Connect(k, p, q) denotes the game Connect$(\infty, \infty, k, p, q)$, played on infinite boards.
Second, this paper analyzes the characteristics of these games, especially for the fairness. In the analysis of fairness, we first exclude the ones which are apparently unfair or solved. Then, for the rest of games, we argue that $p = 2q$ is a necessary condition for fairness in the sense that one player always has q more stones than the other after making each move. Among these games, Connect(6,2,1) is most interesting to this paper and is named *Connect6*.
Finally, this paper proposes a threat-based strategy to play Connect(k,p,q) games and implements a computer program for Connect6, based on the strategy. In addition, this paper also illustrates a new null-move search approach by solving Connect$(6, 2, 3)$ that the first player wins. This result also hints that for Connect6 the second player usually does not place the initial two stones far away from the first stone played by the first player.

1 Introduction

Traditionally, the game *k-in-a-row* is defined as follows. Two players, say B and W, alternately place one stone, black and white respectively, on one empty *intersection*, henceforth called a *square*, of an $m \times n$ board; and B plays first. The one who gets k consecutive stones first (horizontally, vertically or diagonally) of her own wins. Such games are also called mnk-games in [11]. A well-known and popular game is *five-in-a-row*, and also called *Go-Moku*. Go-Moku in the free style [1, 11] is a (15,15,5)-game. For combinatorial analysis, researchers [7, 14] investigated the games allowing each player to place p stones for one move.

This paper introduces a new family of k-in-a-row games, *Connect*(m, n, k, p, q). In Connect(m, n, k, p, q), two players alternately place p stones on an $m \times n$ board for each move except for that the first player places q stones for the first move. Still, the player who gets k consecutive stones of her own first wins. Games in the

family are called *Connect games* in this paper. Obviously, Go-Moku in the free style is Connect(15,15,5,1,1). For simplicity, *Connect*(k, p, q) denotes the game Connect$(\infty, \infty, k, p, q)$, played on infinite boards.

For Connect games, the major difference from traditional k-in-a-row games is to have an extra parameter q, a key that significantly affects the fairness. The higher q, the higher chances B has to win. In [11], Herik, Uiterwijk, and Rijswijck gave a definition of fairness as follows: A game is considered a *fair* game if it is a draw and both players have a roughly equal probability on making a mistake. From this, we argue that $p = 2q$ is a necessary condition for fairness, in the sense that one player always has q more stones than the other after making each move. Among these games, Connect(6,2,1) is most interesting to this paper and is named *Connect6*. More about fairness are discussed in Section 2.

This paper is organized as follows. Section 2 discusses the issue of fairness. Section 3 describes other characteristics of these games. Section 4 proposes a threat-based strategy to play Connect(k, p, q) games and implements a computer program for Connect6, based on the strategy. Section 5 illustrates a new null-move method by solving Connect(6,2,3) that B wins. This result also hints that W usually does not place the initial two stones far away from B's first stone. Section 6 concludes our work.

2 Fairness

This section is organized as follows. Subsection 2.1 reviews the fairness problem of Go-Moku. Subsection 2.2 reviews the Connect games solved based on combinatorial analysis, and also solves some more Connect games. Subsection 2.3 points out some unfair Connect games based on empirical experiments. Subsection 2.4 discusses the fairness of Connect6.

2.1 Fairness of Go-Moku

Fairness has been a major issue for Go-Moku, even though it is a popular game. In the free style of rules (without any restriction on B), it has been well known that the game favors B. To reduce the unfairness, Japanese Professional Renju Association [12] added some rules to restrict the play of B for professional players. For example, B is prohibited to play double three and double four (see the definitions in [1, 3]). The game with these restrictions is called *Renju*. In fact, Renju still favors B, from the experiences of professionals. Theoretically, it was proved that B wins in the free style [1, 3], and that B wins even under these restrictions [20].

The Renju International Federation (RIF) changed the rules [15] to make it fairer by imposing some opening rules for the first five moves. Furthermore, RIF requested a proposal [16] for better opening rules again in 2003. These indicate that it is still hard to define a fair rule for the game. In fact, adding more rules also makes it harder to learn the game.

The fairness problem for Go-Moku or Renju also causes an important side effect, reducing the board size. It was argued in [17] that a larger board increases black's advantage which results in the standard board size of 15×15. However, on the other hand, a smaller board reduces the complexity of the game as described in Section 3. Consequently, it becomes easier to solve the game.

2.2 Solved Games

In addition to [1, 3, 20] mentioned above, many researchers were engaged in studying the fairness of k-in-a-row. W ties [23] when $k \geq 8$. Many solved mnk-games are listed in [11, 19].

For simplicity of combinatorial analysis, many researches [6, 7, 13, 14] followed an asymmetric version of rules, called *Maker-Breaker*, where W is not allowed to win. This is because either B wins or W ties in Connect(k, p, p) as proved in [7, 13]. In contrast to Maker-Breaker, the version of normal rules is called *Maker-Maker*. Let δ denote $k - p$. In the Maker-Breaker version, it was proved in [14] that W ties in a condition, roughly like $\delta = \Omega(\log_2 p)$ (cf. Theorem 1 of [14]), and B wins in a condition, roughly like $\delta = O(\log_2 p / \log_2 \log_2 p)$ (cf. Theorem 2 of [14]). The above result implies that in the Maker-Maker version W still ties in the condition of $\delta = \Omega(\log_2 p)$, but does not imply that in the Maker-Maker version B still wins in the case of $\delta = O(\log_2 p / \log_2 \log_2 p)$. We can easily extend Pluhar's result [14] to the following Corollary.

Corollary 1. *For Connect(k, p, q), let k and p satisfy the condition defined in Theorem 1 of [14]. For all q, where $1 \leq q \leq p$, W ties.* ∎

Now, we want to investigate those Connect games that either B or W wins. First, B wins when $p < \lfloor q/\delta^2 \rfloor (4\delta + 4)$. For example, if B places δ^2 stones on $\delta \times \delta$ squares as a group, W needs $4\delta + 4$ stones to defend the group. Thus, the above result is obtained when B lets $\lfloor q/\delta^2 \rfloor$ groups be far away from one another.

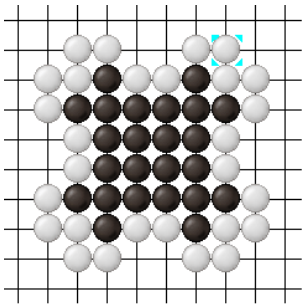

Figure 1. Required defensive white stones when $q = 24$ and $\delta = 4$.

If q is not a multiple of δ^2, we can possibly obtain tighter results. For example, for 4×4 squares, B can add 8 extra stones as shown in Figure 1 such that

W needs one more white stone to defend for each extra black stone. Thus, if $(q \bmod \delta^2) \leq 8\lfloor q/\delta^2 \rfloor$, then B wins when $p < \lfloor q/\delta^2 \rfloor(4\delta + 4) + (q \bmod \delta^2)$. Otherwise, B wins when $p < \lfloor q/\delta^2 \rfloor(4\delta + 4) + 8\lfloor q/\delta^2 \rfloor$. Thus, we obtain the following Corollary.

Corollary 2. *Let $\delta = k - p$. For Connect(k, p, q) game, B wins when $p < \lfloor q/\delta^2 \rfloor(4\delta + 4) + \min(q \bmod \delta^2, 8\lfloor q/\delta^2 \rfloor)$.* ∎

For some cases, we can obtain some even tighter results, based on the above method. For example, for Connect(19,17,7), it does not satisfy the condition of Corollary 2, but B still wins. In addition, it is also possible that W wins. For example, Connect(12,10,3). More results can be obtained based on the above principle and are not elaborated in this paper.

Since Connect games will become even more unfair when $q > p$, we usually assume $q \leq p$. Then, when $p < \lfloor q/\delta^2 \rfloor(4\delta + 4)$, we obtain $\delta < 2 + \sqrt{2}$ or $\delta < 4.82$. Thus, Corollary 2 becomes useless when $\delta \geq 5$ and $q \leq p$. For example, when $\delta = 5$ and $q = 25$, but W only needs p=24 to defend. An open problem is whether it can be proved that either B or W wins when $\delta \geq 5$ and $q \leq p$.

2.3 Empirically Unfair Connect Games

This subsection makes some empirical experiments to investigate the fairness of some Connect games in the following two ways. First, we try to prove informally that either B or W wins. Second, if we cannot prove, we try to see whether one player keeps obtaining initiatives leading to a win. If so, we call that the game *favors* that player.

Table 1. The empirical results for Connect games with $k \leq 9$ and $\delta = 3$.

$q(\leq p)$	$k=4$ $p=1$	$k=5$ $p=2$	$k=6$ $p=3$	$k=7$ $p=4$	$k=8$ $p=5$	$k=9$ $p=6$
1	B	B	W	W	W	W
2		B	W	W	W	W
3			B	FB	FB	FW
4				B	B	FW
5					B	B
6						B

Our empirical experiments for $k \leq 9$ show that most games with $\delta \leq 3$ are unfair, as shown in Table 1 for $\delta = 3$. In this table, B (W) indicates that it is informally proved that B (W) wins; and FB (FW) indicates that the game favors B (W).

The game histories of these experiments are recorded in [21]. Among these game histories, the one for Connect(9,6,4) illustrated in Figure 2 shows an interesting result. Even though B has four stones initially, W still wins by placing

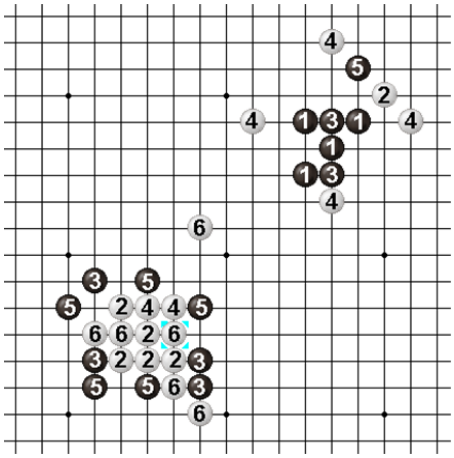

Figure 2. Favoring W for Connect(9,6,4).

five of its six stones far away from B and then keeping obtaining the initiative subsequently. The phenomenon of playing away from the major battle field is called *breakaway* in this paper. In Figure 2, W makes an *initial breakaway*. If initial breakaways do not incur penalty, the game may become unfair. For example, in Figure 2, if B plays in the lower left part subsequently, the game is like Connect(9,6,5) with W playing first.

2.4 Fairness of Connect6

In practice, based on our empirical experiments on Connect6 so far, we have not been able to identify which player the game favors. Following are two more arguments about its fairness.

- As described in Section 1, we argue that $p = 2q$ is a necessary condition for fairness, in the sense that one player always has q more stones than the other after making each move. Connect6 satisfies the condition.
- We argue that the following is a necessary condition for fairness: the initial breakaway does not apparently favor W. In Section 5, we prove that B wins for Connect(6,2,3). This hints that the initial breakaway does not favor W for Connect6. Note that if B does not win for Connect(6,2,3), W can place the initial two stones far away from the first black stone. Thus, if B goes to defend W's two stones, the game is just like Connect(6,2,2) with W playing first, which favors W.

Surely, it is expected to see more evidences, either fair or unfair, or more practical experiences in the future.

3 Game Characterics

In this Section, we investigate the characteristics of Connect6 and some Connect games by following the definitions in [11]. We list the characteristics of Connect6 as follows.

- The rules of Connect6 are very simple to learn. Renju includes some prohibited moves and International Renju even includes some opening rules.
- Connect6 is potentially fair based on the arguments in Subsection 2.4.
- Connect6 is symmetric, if the first move by B is not considered.
- Both state-space and game-tree complexities for Connect6 are very high as described below.

Connect6 has an infinite board, so both state-space and game-tree complexities are infinite too. In order to make it countable, we use a Go board for Connect6, instead, that is, Connect(19,19,6,2,1). Both state-space and game-tree complexities for it are still much higher than those in Go-Moku and Renju, in the sense that two stones per move make the branch factor increase by a factor of half of the board size. Based on the standard used in [11], the state-space complexity of Connect(19,19,6,2,1) is 10^{172}, the same as that in Go. If a larger board is used, the complexity is much higher.

Now, let us investigate the game-tree complexity. For Connect(19,19,6,2,1), assume that the averaged game length is still 30, the same as the estimation for Go-Moku [1]. The number of squares is about 300, and the number of choices for one move is about $(300 \times 300/2)$. Thus, the game-tree complexity is about $(300 \times 300/2)^{30} \sim 10^{140}$, much higher than that for Go-Moku. Also, if a larger board is used, this complexity is much higher.

The game-tree complexity grows much larger, as the value p increases. For example, consider Connect(8,4,2) in case that it is fair. Based on the above calculation with a 19×19 Go board used, the game-tree complexity grows up to 10^{260}. In fact, in our empirical experiments, a higher value p usually requires a larger board size, that makes the state-space complexity even higher.

4 Threat-Based Strategy

For Connect games, the threat-based strategy is a common strategy used to play. Subsection 4.1 describes the threats for Connect6, while Subsection 4.2 generalizes the threats for all Connect games. Subsection 4.3 briefly describes our Connect6 program.

4.1 Threats for Connect6

Definition 1. *For Connect6, assume that one player, say W, cannot connect six. B is said to have t threats, if and only if W needs to place t stones to prevent B from winning in B's next move.* ∎

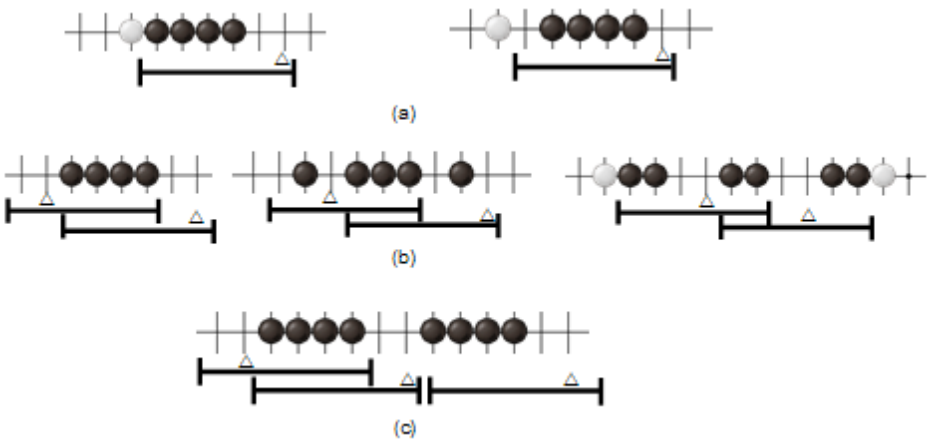

Figure 3. Threat patterns for Connect6. (a) One threat, (b) two threats, and (c) three threats.

For Connect6, we define threats in Definition 1. In Figures 3(a), 3(b), and 3(c), B has one, two, and three threats, respectively. In the case of three threats, B wins since W needs three stones to defend but only have two stones for a move. Thus, the winning strategy of a player is to have at least three threats.

Now, let us investigate how to count the number of threats in one single line for simplicity. For example, in the right one of Figure 3(a), B has only one threat (not two), because W only needs to place one stone at the square above "△". The algorithm to count the number of threats in one line is as follows.

1. For a line, slide a window of size six from the left to right.
2. Repeat the following step for each sliding window.
3. If the sliding window contains neither white stones nor marked squares and at least four black stones, add one more threat and mark all the empty squares in the window. Note that in fact we only need to mark the rightmost empty square. The window satisfying the condition is called a *threat window*.

In Figure 3, the squares above "△" indicate the marked squares, if we only mark the rightmost empty one. Lemma 1 (below) shows that the algorithm is correct.

Lemma 1. *For Connect6, the above algorithm counts the number of threats correctly.*

Proof. First, any two threat windows found by the above algorithm do not cover the same empty squares, since one empty square will be marked at most once. Thus, for each threat window, W needs to place at least one stone to prevent B from connecting six. That is, if the above algorithm finds t threat windows, then there are at least t threats.

Second, we want to prove that if the above algorithm finds t threat windows for B, then it suffices to defend by placing one stone at the rightmost empty square for each threat window, as illustrated in Figure 3. Assume by contradiction that B still wins after the t stones, that is, there still exists at least one sliding window that contains at least four black stones and no white stones. Then, according to the above algorithm, one of these sliding windows must be a threat window. Thus, the rightmost empty square of this window must be marked and be one of the t squares, contradictory to the assumption. ∎

Lemma 2. *In Connect6, consider one single line only. Placing one stone on that line increases threats by at most two.*

Proof. Let B place a stone S on a line. It suffices to prove that the stone is covered by at most two threat windows from the above algorithm. Let T_1 be the first threat window covering S. If the threat window T_2 next to T_1 exists and covers S, the empty squares covered by T_2 must be to the right of S and the empty squares covered by T_1 must be to the left of S, since any two threat windows do not cover the same empty squares (as describe above). Thus, apparently, the next threat window T_3, if it exists, must not cover S, since the empty squares of T_2 is not to the left of S for the same reason. So, S is covered by at most two threat windows. ∎

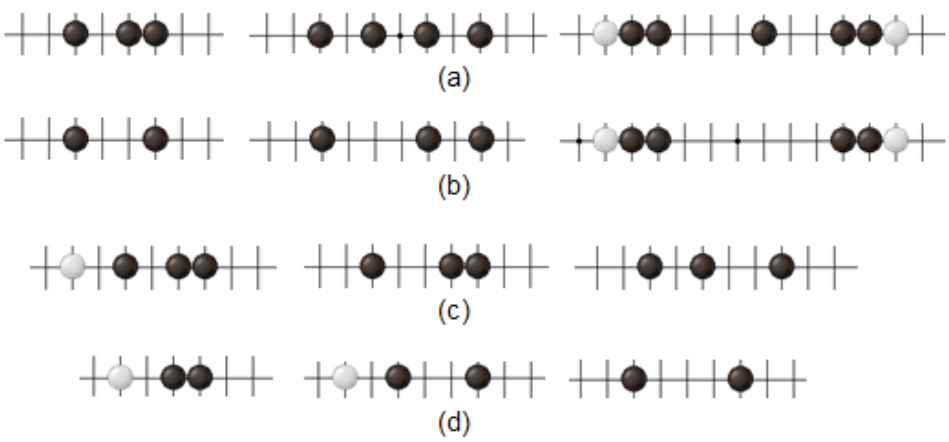

Figure 4. Live-l and dead-l threats for Connect6. (a) Live-3, (b) live-2, (c) dead-3, and (d) dead-2 threats.

Lemma 2 shows that placing one stone on a line increases threats by at most two. From this lemma, we can evaluate the value of one line by counting how many stones one player must place subsequently in order to cause one threat or two threats. For example, in Go-Moku or Renju, a three is called *live-three*

if it has two open ends and can create two threats by adding one stone, and *dead-three*, if it has only one open end and can create one threat only by adding one stone. Following the similar concept, we define *dead-l* and *live-l* threats in Definition 2 (below). For example, Figure 4 illustrates the cases of live-3, live-2, dead-3, dead-2 threats.

Definition 2. *In Connect6, a line includes a dead-l threat for one player, say B, if B only needs to add extra* $4-l$ *stones to create one threat. Similarly, a line includes a live-l threat for B, if B only needs to add extra* $4-l$ *stones to create two threats.* ∎

In Connect6, live-3, live-2, dead-3, and dead-2 threats are also important, since a move including two stones can make them become real threats. Especially, when players attack with real threats, it is better to associate these threats with more live and dead threats. This is a very useful strategy, also used in our Connect6 program in Subsection 4.3.

4.2 Threats for Connect Games

This subsection generalizes the work in Subsection 4.1 to Connect(k, p, q).

Definition 3. *In a line pattern of Connect*(k, p, q)*, assume that one player, say W, cannot connect up to* k*. B is said to have* t *threats, if and only if W needs to place* t *stones to prevent B from winning in the next move.* ∎

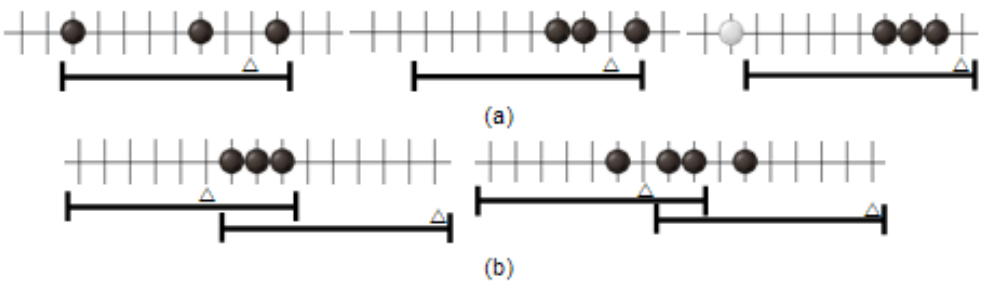

Figure 5. Threats for Connect(9,6,3). (a) One threat and (b) two threats.

Definition 3 defines the number of threats for general Connect games. For example, for Connect(9,6,3), B has one threat in Figure 5(a) and two in Figure 5(b). In general, the winning strategy of a player is to have at least $(p+1)$ threats, since the opponent only has p stones to defend. For example, for Connect(9,6,3), one player needs to have 7 threats to win the game.

The algorithm in the previous subsection can be slightly modified to count the number of threats for Connect games, as follows.

1. For a line pattern, slide a window of size k from the left to right.
2. Repeat the following step for each sliding window.
3. If the sliding window contains neither white stones nor marked squares and at least $\delta(= k-p)$ black stones, add one more threat and mark all the empty squares (or the rightmost one only) in the window.

The above algorithm counts the number of threats correctly for Connect(k, p, q), as in Lemma 3 (below), whose proof is similar to that of Lemma 1 and therefore omitted. Similarly, placing one stone on a line increases threats by at most two, as in Lemma 4 (below), whose proof is also omitted. Similarly, for Connect(k, p, q), dead-l and live-l threats are defined in Definition 4 (below).

Lemma 3. *For Connect games, the above algorithm counts the number of threats correctly.* ∎

Lemma 4. *In Connect(k, p, q), consider one single line only. Putting one stone on that line increases threats by at most two.* ∎

Definition 4. *In Connect(k, p, q), a line includes a dead-l threat for one player, say B, if B only needs to add extra $(\delta - l)$ stones to have one threat. Similarly, a line includes a live-l threat for B, if B only needs to add extra $(\delta - l)$ stones to have two threats.* ∎

Now, let us go back to review Go-Moku, Connect(5,1,1). In Go-Moku, since players can place one stone ($p = 1$) only for each move, players cannot defend live-4 threats. Furthermore, in the case of not winning, players must defend a live-3 threat. Otherwise, the opponents can place one stone to make it a live-4 to win the game. Since players must defend live-3 threats in this case, live-3 threats can be viewed as *delayed threats*.

4.3 Programs for Connect Games

In this subsection, we will first describe the algorithm to generate moves for Connect games, and then describe the search techniques used.

Generating Moves

For Go-Moku or Connect games with $p = 1$, players usually order all the empty squares based on some criteria, e.g., the threats mentioned in Subsections 4.1 or 4.2, and then choose the best one to place a stone. However, for Connect games with $p \geq 2$, players cannot simply choose the best p squares to play. A common example for Connect6 is that the two squares may form two live-3 threats from two live-2 threats respectively. But, in most cases, it is better to use two stones to have one live-2 threat become two threats. That is, players need to consider the value of placing two stones together. For this problem, we use the following algorithm to generate moves for Connect6.

1. Order all the empty squares into a list L in a descending order according to the evaluated values.

2. Choose the first (best) w ones from L, $(s_1, s_2, ..., s_w)$.
3. For each square s_i, repeatedly do the following two steps.
4. Place a stone at s_i and then order all the empty squares into a new list L_i according to the re-evaluated values.
5. Choose the first w_i ones from L_i, and then, for each square s in the w_i squares, put a pair of squares (s_i, s) into the candidate list L_C. Note that if (s, s_i) is already in L_C, skip it.
6. Order the squares in L_C according to the re-evaluated values and choose the first w' pairs.

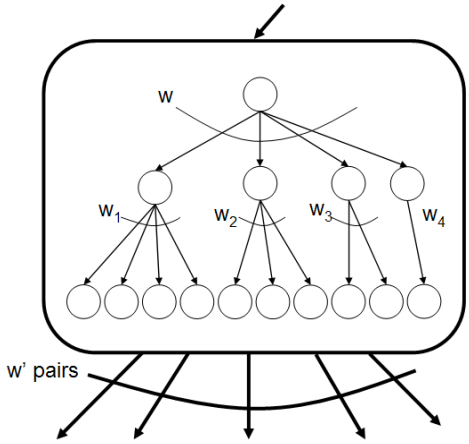

Figure 6. Tree search for generating moves for Connect6.

We suggest a monotonically-decreasing function for w_i, e.g., $w_i = w - i + 1$, as illustrated in Figure 6. Note that in Figure 6 the smaller circles are called *subnodes* to be distinguished from the bigger rounded rectangle, representing a move node. Now, we can see that the time complexity of a move node is quite high since one move node may include many subnodes. For example, if $w = 10$ and $w_i = w - i + 1$, the number of subnodes is 55; and if $w = 30$, it is about 500.

Amazons games [11] are the games that also require one player to do two operations for one move. So, programs for Amazons games may also require search trees with height two to generate moves for each move node. However, for Connect(k, p, q) with larger p, we need a search tree with higher height for each move node. This results in a much higher time complexity for larger p. Avetisyan and Lorentz [4] proposed a null-move technique for the first operation to generate moves. For Connect games, it is still an open issue to reduce the number of subnodes inside one node.

Search

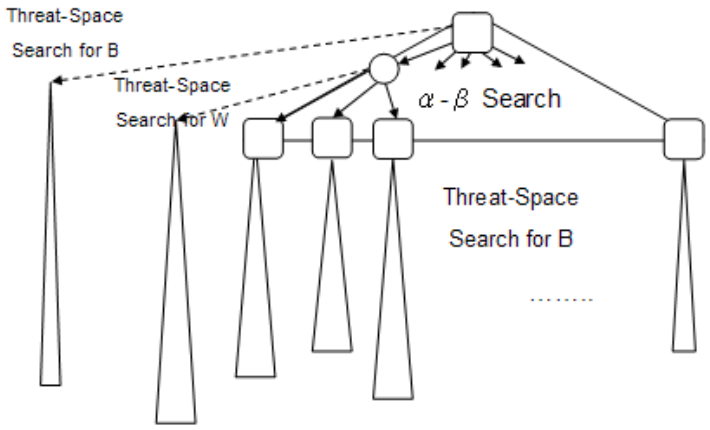

Figure 7. A two-level tree search for Connect6.

Like Go-Moku, threat-based search is also important in Connect games. The search techniques, developed in the past, such as threat-space search and proof-number search [1], are also useful for Connect games. For Connect6, we use a two-level search technique, one level for normal alpha-beta search and the other for threat-space search as shown in Figure 7. In our program, the depth of the alpha-beta search tree is about 3.

5 Null-Move Heuristic

The null-move heuristic [1, 5, 8–10, 18] is usually used with threat-space search and proof-number search to determine implicit threats, such as Go-Moku and Go. The basic idea is to let one player, say W, make a null-move and then apply the threat-space search to finding winning sequences of threat moves for B. If the sequences are found, we need to determine the relevant zone for W to defend. The relevant zone is called *R-Zone* in [18]. However, for Connect games with $p \geq 2$, it is necessary to modify the above null-move heuristic.

For Connect games, this section illustrates a new null-move approach ([22] in greater detail), by solving the game Connect(6,2,3) that B wins. Figure 8 shows a winning sequence of B's threat moves after a null-move by W. Note that in this figure W has four stones for each move, such as four 2's, because all the three kinds of defenses by W include the four squares, and such a defense will greatly reduce the search tree as mentioned in [1].

In Figure 8, the shadowed zone, called *R1-Zone*, indicates that W must place at least one of two stones in the zone in order to defend B's threat sequence possibly. Namely, for all moves (s_1, s_2) that W may defend B's threat sequence, either s_1 or s_2 must be in the zone. For safety, the zone should be large enough

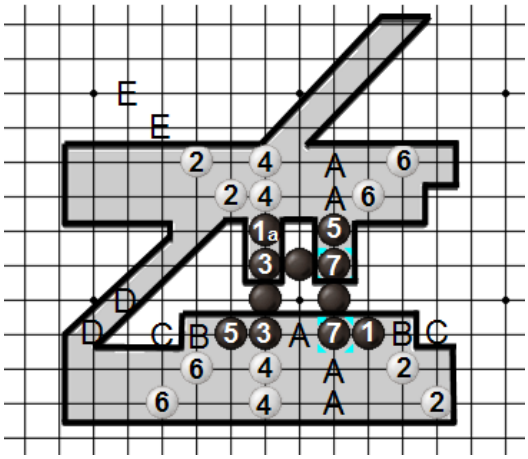

Figure 8. A winning sequence of B's threat moves after a null-move by W.

to cover all possibilities. However, on the other hand, we want to minimize it to reduce the cost of search. Following are our rules to make R1-Zone.

1. All black and white stones except for the initial three black stones are in R1-Zone, since it is possible for W to defend by placing a stone on any of these squares.
2. All defensive squares for the final threats are in R1-Zone. For example, all As and Bs in Figure 8. Note that for the single threat between two Bs in Figure 8, R1-Zone includes both Bs, but not Cs for the following reasons. Since both Bs can be used to block the threat, both Bs are included. But, since Cs cannot block the threat without the middle A, Cs does not have to be in the zone (note that the middle A is already in the zone).
3. For each pair of two empty squares, if two white stones placed on them can build a threat, the two squares are in the zone. For example, Ds in Figure 8. However, those Es are not in the zone, because for the upper left two 2's we actually place one stone only and thus two extra white stones at Es cannot build a threat.

After determining R1-Zone, for each square in R1-Zone we run a semi-null-move process, illustrated by the square $1a$ in R1-Zone in Figure 8. First, W places one stone at $1a$, but, makes a "null-move", called *semi-null-move* in this paper, for the second stone. Figure 9 shows the winning sequence of B's threat moves after the semi-null-move, and a new relevant zone, called R2($1a$)-Zone. R2($1a$)-Zone can be obtained based on the rules for R1-Zone, except for Rules 2 and 3 with slight changes as follows. For Rule 2, for a single threat, we only consider the squares that can block one threat. For Rule 3, for one empty square, if one white stone placed on it can build a threat, the square is in the zone. For all s in R2($1a$)-Zone, the pairs $(1a, s)$ are added into a defense list $L_{6,2,3}$, unless

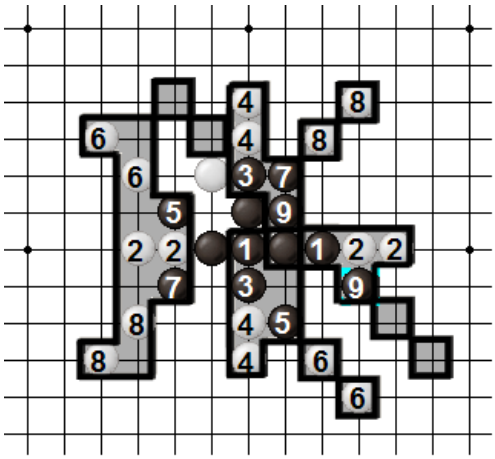

Figure 9. A winning sequence of B's threat moves after a semi-null-move.

the redundant one $(s, 1a)$ exists already. After all semi-null-move processes are done, $L_{6,2,3}$ includes all the moves that W may defend B's attack. Then, if all the moves in the list cannot be played to defend B's attack, B wins. In our experiments for Connect(6,2,3), there are 61 squares in R1-Zone, and there are 1514 pairs in $L_{6,2,3}$. By going through each pair, we finally prove Corollary 3 (below). An important implication of Corollary 3 is to hint that for Connect6 an initial breakaway does not favor W as described in Subsection 2.4.

Corollary 3. *For Connect(6,2,3), B wins.* ∎

6 Conclusion

The contribution of this paper is summarized as follows.

1. This paper introduces a new family of k-in-a-row games, Connect(m, n, k, p, q). Among these games, Connect(6,2,1) or Connect6 is potentially fair, based on some arguments in Subsection 2.4. Thus, Connect6 has the potential to become popular.
2. This paper proposes a threat-based strategy to play Connect(k, p, q) games and implements a computer program for Connect6, based on the strategy.
3. This paper illustrates a new null-move search approach to solve Connect(6,2,3) with B winning. This result hints that for Connect6 an initial breakaway does not favor W.

In addition, this paper also leaves several open problems, such as fairness, null-move heuristic, and reducing the time complexity for Connect games. We expect to see fruitful research related to the games in the future.

References

1. Allis, L. V.: Searching for solutions in games and artificial intelligence. Ph.D. Thesis, University of Limburg, Maastricht (1994)
2. Allis, L.V., Meulen, M. van der, Herik, H.J. van den: Proof-number search. Artificial Intelligence 66 (1) (1994) 91–124
3. Allis, L. V., Herik, H. J. van den, and Huntjens, M. P. H.: Go-Moku Solved by New Search Techniques. Computational Intelligence, Vol. 12 (1996) 7–23
4. Avetisyan, H.,Lorentz, R.: Selective Search in an Amazons Program. Computers and Games (2002) 123-141
5. Beal, D.F.: Experiments with the Null Move. Advances in Computer Chess 5 (ed. D.F. Beal), Elsevier Science Publishers B.V., Amsterdam, The Netherlands (1989) 65–79
6. Beck, J.: On positional games. J. of Combinatorial Theory Series A 30 (1981) 117–133.
7. Csirmaz, L.: On a combinatorial game with an application to Go-Moku. Discrete Math. 29 (1980) 19–23.
8. Cazenave, T.: Iterative Widening. Proceedings of IJCAI-01, Vol. 1 (2001) 523–528
9. Cazenave, T.: Abstract Proof Search. Computers and Games. (eds. T. A. Marsland and I. Frank) Vol. 2063 of Lecture Notes in Computer Science, Springer. ISBN 3-540-43080-6 (2001) 39–54
10. Cazenave, T.: A Generalized Threats Search Algorithm. Computers and Games. Vol. 2883 of Lecture Notes in Computer Science, (2003) 75–87
11. Herik, H. J. van den, Uiterwijk, J.W.H.M., Rijswijck, J.V.: Games solved: Now and in the future. Artificial Intelligence, Vol. 134 (2002) 277–311
12. Japanese Professional Renju Association: History of Renju Rules. `http://www.renjusha.net/database/oldrule.htm`
13. Pluhar, A.: Generalizations of the game k-in-a-row. Rutcor Res. Rep. (1994) 15–94
14. Pluhar, A.: The accelerated k-in-a-row game. Theoretical Computer Science, 271 (1-2) (2002) 865–875
15. Renju International Federation: The International Rules of Renju. `http://www.renju.nu/rifrules.htm` (1998)
16. Renju International Federation: MOM for the RIF General Assembly. `http://www.renju.nu/wc2003/MOM_RIF_030805.htm` (2003)
17. Sakata, G. and Ikawa, W.: Five-In-A-Row. Renju. The Ishi Press, Inc., Tokyo, Japan. (1981)
18. Thomsen, T.: Lambda-search in game trees - with application to Go. ICGA Journal, Vol. 23 (4) (2000) 203–217
19. Uiterwijk, J.W.H.M., Herik, H.J. van den: The advantage of the initiative, Information Sciences 122 (1) (2000) 43–58
20. Wagner, J., Virag, I.: Solving Renju, ICGA Journal, Vol. 24 (1) (2001) 30–34
21. Wu, I-C., Huang, D.-Y.: Web pages for Connect6. `http://connect6.csie.nctu.edu.tw` (2005)
22. Wu, I-C., Huang, D.-Y.: Null-move search for Connect Games. In preparation. (2005)
23. Zetters, T.G.L.: Problem S.10 proposed by R.K. Guy and J.L. Selfridge, Amer. Math. Monthly 86 (1979), solution 87 (1980) 575–576