Bitboard knowledge base system and elegant search architectures for Connect6

Shi-Jim Yena,⇑, Jung-Kuei Yangb, Kuo-Yuan Kao, c Tai-Ning Yangd

a Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan
b Department of Applied Foreign Languages, Lan Yang Institute of Technology, I Lan, Taiwan
c Department of Information Management, National Penghu University, Penghu, Taiwan
d Department of Computer Science and Information Engineering, Chinese Culture University, Taipei, Taiwan

Abstract

Efficiency is critical for game programs. This paper improves the search efficiency of Connect6 program by encoding connection patterns and computing the inherent information in advance. Such information is saved in a bitboard knowledge base system, where special bitwise operations are designed. This paper also proposes efficient methods of generating threat moves and the Multistage Proof Number Search. The methods reduce the time complexity of generating threat moves. The search improves the search performance by developing candidate moves in stages according to their importance. In brief, this paper proposes an efficient knowledge base system and elegant search architectures for Connect6. It is expected that the proposed methods can be applied to all kinds of Connect-k games.

Keywords:
Bitboard
Connect6
Connect-k games
Threat Space Search
Proof Number Search

1. Introduction

Since Wu [30] developed k-in-a-row or Connect-k games in 2005, Connect (19, 19, 6, 2, 1) or Connect6, which is derived from Gomoku, has been a popular research topic [30–32,38]. In Connect (m, n, k, p, q), two players alternately place p stones on an m × 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 [31].

Connect6 has two important features: numerous candidate moves and sudden-death property. Numerous candidate moves lead to complex search and the sudden-death characteristic increases the search complexity. The possibility of sudden-death should be considered in every game position. The player who neglects this feature may lose the game. Victory in Connect6 requires achieving a configuration of six or more consecutive stones positioned in a straight line. The first player who achieves this goal will win this game. Threatening attempts to establish such a position is an important message. Wu and Huang mentioned that threats are the key to play Connect6 as well as Connect-k games. This paper follows the definitions of threats and threat moves by [30] and [31].

This paper aims to provide efficient search architecture for Connect-k games. Firstly, the bitboard knowledge base system and bitwise operations are proposed. Secondly, this paper describes how to generate threat moves efficiently. Finally, the Multistage Proof Number Search (PNS) is proposed. The proposed methods are implemented on a Connect6 program, named Kavalan. Kavalan won all the silver medals in 2010 Computer Olympiad, 2010/2011 TAAI and 2011 TCGA tournaments [18,33,37,39].

This paper is organized as follows: In Section 2, we review the background and the related work. Multistage Search, Threat Space Search and Proof Number Search are discussed. Section 3 and Section 4 present the bitboard knowledge base system and related bitwise operation algorithms for Connect6. Section 5 describes the issue for the methods of generating threat moves. Section 6 provides the Multistage PNS. Section 7 contains the experimental results and our discussions. Finally, Section 8 is our conclusion.

2. Background and related work

This section discusses Multistage Search, Threat Space Search and Proof Number Search.

2.1. Multistage search in Connect6

Search tree is a tool for describing the changing positions of players in a game, and AND/OR search tree can be used to describe searches in the case of Connect6. The player who plays moves first is called offensive side or Attacker and the other player is called defensive side or Defender as in Fig. 1. The root node is level 0 of the search tree. Meanwhile, level 1 of the search tree comprises the candidate moves of offensive side that are generated according to the root position.

In Connect6, each level of the search tree involves a large number of candidate moves. The state space of the search tree is

http://dx.doi.org/10.1016/j.knosys.2012.05.002
2.2. Threat space search

Threat Space Search (TSS) [30] is a common searching method in Connect-k games. TSS focuses on the moment when Attacker plays threat move, at which point Defender should play the corresponding blocking move. TSS uses the threat moves to control the response moves of Defender. Attacker then performs a deeper search. TSS is divided to double-threat TSS and single-threat TSS based on the number of threats generated by the move.

For double-threat TSS, Defender needs two stones for blocking threats. In contrast, for single-threat TSS, Defender needs only one stone for blocking. Therefore, Defender is free to place the other stone. The branching factors of double-threat TSS are smaller than that of single-threat TSS.

In a node of a Connect6 search tree, the subtree of the node formed by double-threat TSS is called T2 subtree. In a T2 subtree, if Attacker can identify to achieve a win, Attacker is said to have identified its T2 solution. Otherwise, this T2-subtree is labeled T2 Fail.

In a node of a Connect6 search tree, the subtree of the node formed by Attacker’s double-threat and single-threat moves is called TSS subtree. In a TSS subtree, Attacker is said to have obtained its TSS solution when it identifies a process to achieve a win. Otherwise, this TSS subtree is labeled TSS Fail.

According to the blocking moves played by Defender while searching, double-threat TSS is divided into Normal TSS and Conservative TSS, as detailed below:

2.2.1. Normal TSS

In a game position, a defense set is a set of Defender’s moves that block threats of Attacker. If Defender can only place two stones of the defense set based on the rules of Connect6, this search method is called Normal TSS, or simply TSS. In Fig. 2, the defense set includes the stones 1–4. This search should expand the move as in (A), (B), and (C) of Fig. 2, separately. If the T2 solution is very deep, the state space of the search will be large.

2.2.2. Conservative TSS (CTSS)

CTSS assumes Defender can place all stones of the defense set at a time [30], like (D) in Fig. 2. Because there is only one child node of defensive side, CTSS can search toward very deep levels.

2.3. Proof-Number Search

Proof-Number Search (PNS), developed by Allis [2] and [4], is a reliable algorithm that aims to prove or disprove a binary goal. For Connect6, this binary goal is the TSS solution. PNS was successfully used to prove or solve theoretical values of game positions for many games, such as Checkers, Chess, Connect-Four, Go, Gomoku, Lines of Action, Renju, and Shogi [2–5,14,22,26,27,29]. The core idea of standard PNS is to order moves efficiently. To achieve such an ordering, the PNS algorithm selects moves based on two numbers in each node: proof number and disproof number. PNS uses the AND/OR search tree. Table 1 lists the rules that PNS uses to select the most-proving node. In the OR-Node, PNS selects the minimum proof number from its successors; while in the AND-Node, it selects the minimum disproof number.

This kind of search skill focuses on the node with fewer branches, which helps ensuring effectiveness and speed. The selection of the most-proving node R in Fig. 3 is given by the bold path.

3. Bitboard knowledge base design

Many researches use bits to encode the board states and the related bitwise operations to model the real problems [12,19,28,35,36]. This paper uses the bitboard concept to design a knowledge base for Connect-k games. In Connect-k games, connection information of cells is rather important. This property is different to Chess, Shogi and other games [9,10,23].

Connection is an important property for Connect-k games. The definition of connection was mentioned in [35,36]. The same idea of the two papers is using a data structure to record the connection information of each line of a game position. Then, these connections were stored in a knowledge base in advance. Xu’s paper [35] defined the connection mathematically and focused on decreasing the space complexity of the knowledge base of Connect-k games with \( n \times n \) board. Xu’s paper [35] also studied on a precise classification criterion for connection and a precise classification for the cells.

This paper follows the results of our previous work [36]. We define the data structures of the Line, Connection, and shape. Then, the shape information table and Connection set table are designed for the knowledge base. Compared to Xu’s paper [35], our paper

![Fig. 1. An AND/OR search tree. The rectangle represents an OR-Node (Offensive side), and the circle represents an AND-Node (Defensive side).](image)

![Fig. 2. The double-threat pattern and its blocking stones. (A), (B), and (C) refer to the normal defense, while (D) refers to the conservative defense.](image)

### Table 1

<table>
<thead>
<tr>
<th>Rules for AND-node</th>
<th>Rules for OR-node</th>
</tr>
</thead>
<tbody>
<tr>
<td>( pn(n) = \sum_{s \in \text{successor}(n)} pn(s) )</td>
<td>( pn(n) = \min_{s \in \text{successor}(n)} (pn(s)) )</td>
</tr>
<tr>
<td>( dn(n) = \min_{s \in \text{successor}(n)} (dn(s)) )</td>
<td>( dn(n) = \sum_{s \in \text{successor}(n)} dn(s) )</td>
</tr>
</tbody>
</table>
focuses on improving the search performance by the bitboard knowledge base and related bitwise operation algorithms for Connect-\(k\) games with \(m \times n\) board.

3.1. Basic analysis of board in Connect-\(k\) games

Bitboard is a way to model states of board by binary encoding, and uses bitwise operations to accelerate the process and to improve the efficiency of searching. A popular application of bitboard is the Chess design. It uses 64 bits to represent 64 locations on the board, and employs all kinds of search by using the efficiency of bitwise operators.

For an \(m \times n\) board, the number of vertical lines is \(m\), and the number of horizontal lines is \(n\). The intersection is named cell (1 for short). A cell is either empty or occupied by a stone. Cells could be labeled by the board coordinate or the board index. The board coordinates of cells are constructed by horizontal \(X\)-axis and vertical \(Y\)-axis. Both of them start from the upper left corner of the board. The board coordinate indexes of cells are represented by \(b_{x,y}\).

*Fig. 4A* shows the board coordinates of cells in a \(9 \times 9\) board. Another way to label the cells is the index. The board index number is starting from zero on the upper left corner of the board as well as accumulating one from left to right, top to down. *Fig. 4B* shows the index of the cell.

3.1.1. Line

A cell-array is an array of consecutive cells. There are four directions on a board: vertical, horizontal, slash west and slash east, as in Fig. 5. A Line is the maximal cell-array with the same direction in a straight line on the board. If we want to check whether the two cells \((x_1, y_1)\) and \((x_2, y_2)\) are in the same Line or not, their board coordinate should be examined in the four direction. Four conditions for that of the four directions are stated as follows, respectively.

**Vertical direction:** \(x_1 = x_2\)

**Horizontal direction:** \(y_1 = y_2\)

**Slash West direction:** \(x_1 - y_1 = x_2 - y_2\)

**Slash East direction:** \(x_1 + y_1 = x_2 + y_2\)

Each Line has two end cells. This paper uses bigger index to denote start-cell and the smaller index for terminal-cell. The length of a Line is the number of cells from start-cell to terminal-cell. A cell \((x_1, y_1)\) exists in four Lines with respect to four directions. The lengths of the four Lines are calculated with respect to the four directions as follows, respectively.

**Vertical Lines:** \(n\)

**Horizontal Lines:** \(m\)

**Slash west Lines:**

\[
\begin{align*}
&\text{if } (x - y) + n < \text{Min}(m, n) \\
&\text{if } \text{Min}(m, n) \leq (x - y) + n \leq \text{Max}(m, n) \\
&\text{if } m - (x - y) < (x - y) + n
\end{align*}
\]

**Slash east Lines:**

\[
\begin{align*}
&\text{if } (x + y) < \text{Min}(m, n) - 1 \\
&\text{if } \text{Min}(m, n) - 1 \leq (x + y) \leq \text{Max}(m, n) - 1 \\
&\text{if } \text{Max}(m, n) - 1 < (x + y)
\end{align*}
\]

The Line has four attributes: direction, start-cell, terminal-cell and length. When the length of a Line is smaller than \(k\) in a Connect-\(k\) game, this Line is useless. Otherwise, the Line is useful. In an \(m \times n\) board, the number of useful lines is \(3m + 3n - 4k + 2\).

3.1.2. Connections of a game position

In a game position, if all the stones in a cell-array are the same color, this cell-array is pure. Connection is a segment of a Line of a game position. It is the longest pure cell-array of the pure cell-arrays where contain the same cell state in the Line [35]. In other words, given a specific stone, the maximal region that the stone can extend in a useful Line is its Connection with respect to the Line [36]. Several Connections may exist in a Line in the same time. Two Connections may contain the same empty cell(s) in a Line.

The four attributes of Line are also used in Connection. In a game, each value of the four attributes of Line is fixed, but that of Connection is changeable with respect to different game position. Except for the four attributes, Connection has another two attributes: the stone color and the shape. Shape of a Connection is a binary array, where each array bit corresponds to each state of the cell array in the Connection. If the cell state is empty, the corresponding bit is set to 0. Otherwise, the bit is set to 1.

This paper uses binary encoding to transfer shape to shape code as an unsigned integer. In Fig. 6A, the shape is encoded as "000111100", so the number is 60.

If we only use an unsigned integer value to identify the shape in a Connection, errors will happen in some cases. In Fig. 6, the two shapes with different lengths, "0001111100" and "01111100", have the same encoding value, 60, but the two shapes are different to

![Fig. 3. PNS tree. The numbered pair comprises the proof (left) and disproof (right) numbers.](image)
each other. This mistake can be fixed by comparing the length of the two shapes. Thus, when a shape is encoded, the length of the shape (shape length) is also recorded.

3.2. Design of the bitboard knowledge base

The goal of Connect-k games is to form a Connection with a consecutive k stones. Most of the searching methods used in Connect6 program is threat-based search, such as TSS [30,31] and Relevance-Zone-Oriented Proof (RZOP) search [32]. In those searching methods, shape information is very important because it can speed up the understanding of threats on the board, and offers possible moves for threat-based search. Shape information is developed base on pattern. Pattern is a special kind of shape. If a shape has a special meaning, it is called pattern. For the meaning and development of pattern, please refer to [30,31,35].

Connection set table and shape information table are the two main important components of the bitboard knowledge base. When playing a game, the program analyzes the game position and stores the information of all Connections of a game position in the Connection set table. The evaluation value of an empty cell may fall into different Connections; the cell gets different score with respect to different Connection from the shape information table. We sum up these scores as the cell’s final evaluation value.

3.2.1. The shape information table

This paper encodes each shape of all possible Connections in Connect6 and calculates its inherent information in advance. This information is put into the shape information table. Any Connection’s shape information can be retrieved from the table in constant time.

The number of all possible shapes with length \( n \) is \( 2^n \). This paper starts the encoding from the shortest shape with length \( k \) for Connect-k games, and the range of the values in the encoding is from 0 to \( 2^k - 1 \). In the shape information table, the encoding shape index is from the shortest shape to the longest shape. In Connect-k games, the shape index is calculated by the following formula:

\[
\Phi(n, c) = 2^n - 2^k + c
\]

where \( n \) is the length of the shape, \( c \) is the shape code. For example, if the length of a Connection is 10, and the shape code is 128, the shape index will be 1088 in Connect6. Fig. 7 shows that the structure of the shape information table. There are two fields in the information table: number of threats and type. In Connect6, the length of the table is \( 2^{26} - 2^6 - 1 \).

4. Bitwise operation algorithms on the bitboard information

Most programming languages offer bitwise operators. Bitwise operators are very efficient. Based on the proposed bitboard

<table>
<thead>
<tr>
<th>Shape index</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>o Empty</td>
</tr>
<tr>
<td>1</td>
<td>0 Dead 1</td>
</tr>
<tr>
<td>2</td>
<td>0 Dead 1</td>
</tr>
<tr>
<td>3</td>
<td>0 Dead 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( 2^k - 1 )</td>
<td>Connected</td>
</tr>
</tbody>
</table>

Fig. 7. The structure of the shape information table.

3.2.2. Connection set table

A Connection set is a set of all Connections in a specific Line. For a game board, each useful Line is assigned a serial number as Line index. The Line index is given as the formula in Table 2. Starting from 0 and based on the order of the four direction: vertical, slash west, horizontal and slash east. The Line index is the index of the Connection set table. Given the board coordinate of a cell, the corresponding four Connection sets can be found in the Connection set table. The formulas are shown in Table 2. Fig. 8 illustrates the concept of addressing method from the board coordinate of cells to the index of related Connection set.

Table 2

<table>
<thead>
<tr>
<th>Direction</th>
<th>Line index formula</th>
<th>Line index of cell ((x,y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical</td>
<td>From 0 to ( m - ) ( x )</td>
<td>( m + n - x - y - k )</td>
</tr>
<tr>
<td>Slash west</td>
<td>From ( m ) to ( 2m + n - 2k )</td>
<td>( 2m + n - 2k + y + 1 )</td>
</tr>
<tr>
<td>Horizontal</td>
<td>From ( 2m + n - 2k + 1 ) to ( 2m + n - k )</td>
<td>( 2m + n - k + x + y + 1 )</td>
</tr>
<tr>
<td>Slash east</td>
<td>From ( 2(2m + n - k) + 1 ) to ( 3m + 3n - 4k + 1 )</td>
<td>( 2m + n - k + x + y + 1 )</td>
</tr>
</tbody>
</table>

Fig. 6. The shape code of the shape in the Connection.

Fig. 8. Illustration of the Connection set table and the Line index.
knowledge base, three bitwise operation algorithms are proposed to improve three basic Connect6 functions: modifying Connections after playing a stone, counting the number of threats in a Connection, respectively.

The bitmask is a data structure for bitwise operations in bitwise computing. Take the bitmask insertion (MASK (Insert) for short) as an example in Connect6. If a stone is inserted into a Connection with the same color, the bitwise operation needs a bitmask that can represent the location of the cell related to its shape.

As described in the encoding of a shape, the cells occupied by stones are encoded as “1”; therefore, the location where the player places a stone on the cell is also encoded as “1” in the MASK (Insert) and the other cells are encoded as “0”. Fig. 9 shows an example of the MASK (Insert) and the location where the player inserts a stone into the Connection is the ninth cell from the right end cell of the Connection.

4.1. Modifying a Connection after placing a stone

When placing a stone on a cell, the program will modify the related Connections. We find out the indexes of the Lines belonging to the updated cell, and search the related Connections based on these indexes. When modifying a Connection for an updated cell with the same color, we only change the shape of the Connection. At this time, all we need to do is to change the shape by using bitwise operator “OR” to compute the new shape. Fig. 9 shows the operation in this case.

It is more complex to modify a Connection for an updated cell with different color stone. In this case, it may result two or three new Connections from the old Connection. The attributes of each new Connection should be calculated, separately. If the length of a new Connection is less than k, the Connection is useless. The algorithm is as follows:

Step 1: Use bitwise AND to compute the Mask (Right) and the shape, we get the new shape that is in the right side of the original Connection: 11100000.
Step 2: Use bitwise AND to compute the Mask (Left) and the shape, we get the new shape that is in the left side of the original Connection: 000011.
Step 3: Finally, scan from the inserting cell to its left and right side, the shape in the middle part of the Connection can be sure.

Fig. 10 is an example for this case. The example only explains how to get the right shape of the new Connection.

4.2. Checking whether the game is over

There are many ways to check whether the game is over or not. For example, we can check from the four directions of the updated cell. If there is a consecutive k stones, the game is over. This paper uses bitwise operations on the checking process. The starting point is the location from the updated cell to k – 1 distance toward the start-cell of the Connection. This is based on the situation if there is sufficient distance between the updated cell and the start-cell of the Connection. Fig. 11 shows how to check whether the game is over.

Because the required length of the consecutive stones is k, the checking length of the bitmask is k in every inspection. Therefore, when placing a stone on a cell, the program will modify the related Connections. We find out the indexes of the Lines belonging to the updated cell, and search the related Connections based on these indexes. When modifying a Connection for an updated cell with the same color, we only change the shape of the Connection. At this time, all we need to do is to change the shape by using bitwise operator “OR” to compute the new shape. Fig. 9 shows the operation in this case.

It is more complex to modify a Connection for an updated cell with different color stone. In this case, it may result two or three new Connections from the old Connection. The attributes of each new Connection should be calculated, separately. If the length of a new Connection is less than k, the Connection is useless. The algorithm is as follows:

Step 1: Use bitwise AND to compute the Mask (Right) and the shape, we get the new shape that is in the right side of the original Connection: 11100000.
Step 2: Use bitwise AND to compute the Mask (Left) and the shape, we get the new shape that is in the left side of the original Connection: 000011.
Step 3: Finally, scan from the inserting cell to its left and right side, the shape in the middle part of the Connection can be sure.

Fig. 10 is an example for this case. The example only explains how to get the right shape of the new Connection.

4.3. Counting the number of threats

Wu’s algorithm to count the number of threats in one Connection is as follows [31]:

1. For a Connection, slide a window of size six from start-cell to terminal-cell.
2. Repeat the following step for each sliding window.
3. If the sliding window contains no marked cell and at least four occupied cells, add one more threat and mark all the empty cells in the window. Note that in fact we only need to mark the rightmost empty cell. The window satisfying the condition is called a threat-window.

Fig. 12A shows the concept of the algorithm. The algorithm can correctly figure out the number of threats in a Connection. This paper uses two ways to improve the algorithm as follows:

First, when the sum of stones is less than four in sliding window, we can jump over some checking cells according to the
5. Generating threat move

As mentioned in [30–32], threats are the key to play Connect-6 as well as Connect-k games. For a game position with \( n \) Connections, the time complexity of traditional method for generating threat moves is \( O(n^2) \). This section introduces two efficient algorithms for generating two kinds of threat moves, respectively. The time complexities of the two algorithms are \( O(n) \), which is better than the traditional method.

5.1. Connection diversity

Connections are divided into three types as follows:

- **Threat-Connection**: If Attacker can generate \( k \) consecutive stones by placing one or two stones on a Connection, this Connection is a threat Connection. If Defender can block the threat by a single stone, the Connection is a single-threat Connection. A single-threat Connection has a threat. Meanwhile, a double-threat Connection has two threats and can only be defended by two Defender’s stones.

- **Indirect-Attack-Connection**: The Connection is neither a Threat-Connection nor an Attack-Connection. If one stone is enough to make threats in TSS, the Indirect-Attack-Connection can be promoted to the Attack-Connection by another stone. Then TSS can generate more threat moves.

- **Attack-Connection**: The Connection itself has no threat, but Attacker can generate the Threat-Connection on it by one or two stones. The Attack-Connection provides the basis of TSS. In TSS, if a threat move can generate another Attack-Connection, TSS can perform a deeper search on the new Attack-Connection.

Checking whether an intersection cell exists between Connections associated with a position is a complex problem. It is necessary to examine whether an intersection exist between Attack-Connections or not. In Fig. 13, if Attacker places a stone on the intersection cell, the two dead-3 Connections will transform to two dead-4 Attack-Connections, respectively.

Checking whether an intersection cell exists between Connections associated with a position is a complex problem. It is necessary to examine whether an intersection exists between each pair of Attack-Connections. Suppose the number of Attack-Connections is \( n \), the time complexity of the examination is \( O(n^2) \).

5.2. Threat-cell and threat-pair

This subsection provides the definitions of Threat-Cell and the Threat-Pair. Threat-Cell and Threat-Pair are the main concepts for generating threat moves in this section.

**Definition 1.** In a game position, if threats can be generated after playing a stone on an empty cell, this cell is labeled Threat-Cell. If a Threat-Cell can form one threat, that cell is labeled Threat-Cell. If the number of threats formed by a Threat-Pair equals one, the pair is labeled Threat-Pair set, the set of used Threat-Pair.

**Definition 2.** In a game position, if two empty cells can generate a threat, the cell pair is labeled Threat-Pair. If the Threat-Pair can generate one threat, the pair is labeled Threat-Pair set, the set of used Threat-Pair. If the number of Threat-Pair equals two, the Threat-Pair is labeled Threat-Pair, the set of used Threat-Pair.

Table 3 lists transitions of different Attack-Connection patterns. The table lists the changes in patterns after playing one or two stones, and related Threat-Cell and Threat-Pair.

In TSS, Attacker should generate one or two threats so Attacker can continue TSS. When generates the threat moves, Attacker should also consider Defender’s threat. If Defender has no threat,
Attacker seeks the threat move by using two stones. Section 5.3 describes the algorithm for generating the threat move using two stones. If Defender has one threat, Attacker must use a stone to block that threat. Then Attacker has only one stone for generating the threat move. The algorithm for generating the threat move using a single stone is in section 5.4.

5.3. Generating the threat move using two stones

Suppose Defender has no threat, Attacker can seek to deploy two stones for the threat move. The data used in this subsection are T2-Cell set, T1-Cell set, T2-Pair set and T1-Pair set.

5.3.1. Algorithm for generating the threat move using two stones

Step 1: Search all Attack-Connections for a specific position, and divide them into live-3, dead-3, live-2, dead-2 and live-1 sets according to its Pattern, respectively.

Step 2: If the live-3 set is not empty then deal with the live-3 set as follows:

(a) If there are more than one live-3 connections, take two Connections to form the winning move and end the search. Otherwise, resolve the situation as follows:

(b) Generate corresponding Threat-Cells and Threat-Pairs based on every live-3, and enter them into the T2-Cell set, T1-Cell set, T2-Pair set and T1-Pair set.

Step 3: If the dead-3 set is not empty, deal with every Connection in the dead-3 set sequentially as follows:

(a) Generate the T1-Cell set.

(b) If T2-Cell set has previously been generated, combine this T1-Cell set and T2-Cell set as a double-threat move. If these two sets contain an intersection cell, that cell must be winning cell, forming a winning move to end the search.

(c) Combine this T1-Cell set with the formerly generated T1-Cell set to create the T2-Pairs, and assign these pairs to the T2-Pair set.

(d) If an intersection cell exists between this T1-Cell set and the formerly generated T1-Cell set, include this cell in the T2-Cell set. If this cell is included in the T2-Cell set, it must be the winning cell, forming a winning move and thus ending the search.

(e) Examine the intersection cell of this T1-Cell set with the formerly generated T1-Cell set and T2-Pair set.

(f) Generate the Threat-Pair of the dead-3 Connection and individually include it in the T1-Pair and T2-Pair sets.

Step 4: If the live-2 set is not empty, generate a correspondent Threat-Pair for every Connection in the live-2 set in order, and include that pair in the T2-Pair set and T1-Pair set.

(a) Examine the intersection of the T2-Pair set with the formerly generated T1-Cell and T2-Cell sets.

(b) Examine the intersection of the T1-Pair set with the formerly generated T1-Cell and T2-Cell sets.

Step 5: If the dead-2 set is not empty, generate the correspondent Threat-Pair according to Connections in the dead 2 set in order, and assign it to the T1-Pair set.

(a) Examine the intersection of the T1-Pair set with the formerly generated T2-Cell and T1-Cell sets.

Step 6: According to the T2-Cell set, T1-Cell set, T2-Pair set and T1-Pair set generated from steps 2 to 5, generate individual correspondent threat moves.

(a) Generate double-threat moves for the cells in the T2-Cell set and Indirect-Attack-Connection. Because a T2-Cell can form two threats using a single stone, the other stone can promote the Indirect-Attack-Connection, for example turning live-1 to live-2.

(b) Generate double-threat moves based on the T2-Pair in the T2-Pair set.

(c) Generate single-threat moves for the cells in the T1-Cell set and Indirect-Attack-Connection. Because a T1-Cell can generate a single threat using a single stone, the other stone can promote the Indirect-Attack-Connection.

(d) Generate single-threat moves for the T1-Pair in the T1-Pair set.

5.3.2. Analysis

Suppose the number of Attack-Connections is n. The time complexity of Step 1 is O(n). The time complexity is O(n) from Steps 2 to 5, and that of Step 6 is O(1). Thus, the time complexity of the whole algorithm is O(n).

Some properties of the Threat-Cell and Threat-Pair used in the algorithm are stated as follows. Since the proofs of Properties 3–7 are similar to that of Properties 1 and 2, this paper ignores the proofs. In each of the following Properties, we assume the two given sets are not empty and generated from two different Connections.

Property 1. Given the two T2-Cell sets S1 and S2, for any two different cells c1 ∈ S1 and c2 ∈ S2, Attacker can win by playing on c1 and c2.

Proof. Because the T2-Cell can form two threats using a single stone, place stones on c1 and c2 will make two double-threat patterns in the two Attack-Connections, respectively. Blocking a double-threat pattern requires two stones, which are on the same line. Only one intersection cell exists in two lines, blocking the two Threat-Connection requires at least three stones.

Property 2. Given the T1-Cell set S1 and T2-Cell set S2, if there is an intersection cell c0 of the two sets, Attacker can win by playing on the c0.

Proof. The cell c0 ∈ S2 can generate the double-threat pattern, and c0 ∈ S1 can generate the single-threat pattern. Therefore, c1 and c2 can generate three threats. However, if c0 is empty after Defender plays the two stones, Defender may block the three threats by playing on c1 and another stone.

Property 3. Given the two T1-Cell sets S1 and S2, for any two different cells c1 ∈ S1 and c2 ∈ S2, Attacker can make a double-threat pattern by playing on the c1 and c2.

Property 4. Given the T2-Cell set S1 and T2-Pair set S2, for any cell c1 ∈ S1 and any cell pair (c1p1, c1p2) ∈ S2 if c1 is equal to either c1p1 or c1p2, Attacker can win by playing on the c1p1 and c1p2.

Property 5. In Property 4, if S2 is a T1-Pair set, Attacker can also win by playing on the c1p1 and c1p2.

Property 6. In Property 4, if S1 is a T1-Cell set, Attacker can also win by playing on the c1p1 and c1p2.

Property 7. In Property 4, if S1 is a T1-Cell set and S2 is a T1-Pair set, Attacker can make a double-threat pattern by playing on the c1p1 and c1p2.

Property 8. It is impossible for the T2-Pair or T1-Pair sets share the same intersection Pair.
The analysis of this algorithm is similar to the algorithm in Section 7. For two T1-Pair sets, since each of the T1-Pair set is formed by a single Attack-Connection, a T1-Pair set generated by Attack-Connection must be located on the same straight line as the Connection definition. A maximum of one intersection can exist for two lines on the board. Thus, giving two T1-Pairs, the Connections forming these T1-Pairs must be the same Connection, contradicting the definition of Connection. Conversely, the intersection Pair differs between the T2-Pair and T1-Pair sets. □

5.4. Generating the threat move using a single stones

In a game position, if Attacker should use a single stone to block the threat of Defender, only one stone is available to generate the threat move. Attacker generates double-threat and single-threat moves based on every blocking cell. The algorithm is as follows. The analysis of this algorithm is similar to the algorithm in Section 5.3. The time complexity is also $O(n)$. 

5.4.1. Algorithm for generating the threat move using a single stone

Step 1: Use the following steps to deal with every blocking cell in order.

Step 2: Examine the new position after playing a stone on the blocking cell. Search all the Threat-Connections and Attack-Connections of Attacker, and classify the Attack-Connections according to the Pattern.

(a) Calculate the sum of the threats generated by the blocking cell based on the Threat-Connection.

(b) If the blocking cell generates more than two threats, it is termed a winning cell, and makes a winning move to end the search.

Step 3: Generate the T2-Cell set for every live-3, and generate the T1-Cell set for every dead-3.

(a) If the generated T2-Cell set intersects with the previous one, this intersection must indicate a winning cell, forming a winning move together with the blocking cell to end search.

(b) When generating the T1-Cell set, if an intersection exists with the previous T1-Cell set, this cell is a T2-Cell, and thus is included in the T2-Cell set. If this T2-Cell intersects with the T2-Cell set, it must be a winning cell, and thus forms a winning move together with the blocking cell to end the search.

Step 4: If a blocking cell generates two threats according to Step 2, the situation is dealt with as follows:

(a) If the T2-Cell or T1-Cell set is not empty, a winning move is performed together with the blocking cell to end the search.

(b) Otherwise generate double-threat moves sequentially for the Connections live-2, live-1, dead-2 and dead-1.

Step 5: If a blocking cell generates one threat according to Step 2, the situation is dealt with as follows:

(a) If T2-Cell set is not an empty set, a winning move is formed together with the blocking cell to end the search.

(b) If T1-Cell set is not an empty set, a single-threat move is formed together with the blocking cell.

(c) Check if the Threat-Connection can play another stone to form a double-threat move.

Step 6: If the blocking cell generates no threat according to Steps 2, this algorithm generates corresponding threat moves based on the T2-Cell set and T1-Cell set formed in Step 3.

(a) Check if an intersection cell exists between the T2-Cell set and the T1-Cell set generated from different Connections. If such a cell exists, this cell is a winning cell, which forms a winning move together with the blocking cell and thus ends the search.

(b) Make the blocking cell and T2-Cell set form double-threat moves.

(c) Make the blocking cell and T1-Cell set form single-threat moves.

(d) Check if the Threat-Connection can play another stone to form a double-threat move.

6. Multistage Search and PNS implementation

Multistage Search is based on the concept that, at any game position, different candidate moves can be developed in different stages according to their importance, respectively. This design is mainly used for sudden-death games. For Connect-k games, the priority of candidate move depends on number of threats by the move. Monte-Carlo Tree Search (MCTS) is a best-first search, which uses stochastic simulations [1,6–8,15,25]. It has advanced the development of computer Go substantially [6,16,40]. The 2-Stage MCTS for Connect6 was proposed in [38]. This paper proposes Multistage PNS. The two searching methods are compared in Section 7.

PNS was proposed by Allis et al. [2,4]. PNS was successfully used to prove or solve of game positions for many games [2–5,14,22,26,27,29]. Many variations of PNS were proposed, such as DF-PN, PDS, PN, PN, and parallel PNS [5,11,13,21,22,24,27,29,34]. This paper provides the Multistage PNS search for Connect6. In practice, the proposed Multistage PNS may be combined with some of the variations to improve the efficiency.

6.1. Search architecture

Three kinds of moves exist in Connect6: double-threat, single-threat and non-threat moves. Thus, Multistage Search is divided into four strategies with respect to the combinations of the three kinds of candidate moves, as listed in Table 4.

Fig. 14 shows the architecture of Multistage PNS. Candidate moves of each node are generated via stages. The timing of the generation of different candidate moves are based on the stage

![Fig. 14. One stone can transform the dead-4 into a live-5.](image-url)
strategy and the stage-transition condition. The implementation of 3-stage PNS in Connect6 is explained in the next subsection.

6.2. Three-stage PNS implementation

6.2.1. First stage – Double-threat move search

The first stage focuses on the double-threat solution. As the definitions in Section 2, the T2-subtree is a tree where Attacker only searches for the double-threat moves of Attacker. If Attacker cannot identify any suitable double-threat move in an OR-Node, this node is labeled T2 Fail. In Fig. 15, because Attacker cannot identify any double-threat move in node T, node T is T2 Fail. In a T2-subtree, if one of the successors of Defender node is T2 Fail, Defender node is T2 Fail. In Fig. 16, Defender node B is T2 Fail. If all the successors of Attacker node are T2 Fail, the Attack node is T2 Fail. In Fig. 16, Attacker node j is T2 Fail. If the root node of T2-subtree is T2 Fail, Attacker of this T2-subtree fails in searching double-threat moves. Attacker node A is T2 Fail in Fig. 16.

In the T2-subtree, the calculation of the proof and disproof number of each node is the same as that of PNS, as listed in Table 1. When Attacker node of the T2-subtree is T2 Fail, this node should add single-threat moves to the calculation. The proof and disproof numbers of the entire T2-subtree thus become the proof and disproof number of the TSS-subtree. Finally, if TSS-subtree is TSS Fail, the calculation of the proof and disproof number must consider the non-threat moves.

6.2.2. Second stage – single-threat move search

The second stage focuses on the single-threat solution. In Connect6, it is harder to find TSS solution than that of T2 solution because Defender only needs one move to block the single-threat. Therefore, another move of Defender is free to play on any empty cell.

The TSS Fail is judged using the same criteria as the T2 Fail. This paper takes Attacker node of the TSS-subtree as the criterion. If Attacker cannot keep generating threat move, TSS is Fail. In the node of the TSS-subtree, if TSS fails, non-threat moves will be incorporated into the calculation of the proof and disproof numbers.

6.2.3. Third stage – non-threat move search

We design some criteria for increasing search efficiency of generating non-threat moves. To avoid generating too many candidate moves, we generate candidate moves based on a deeper search on a probing cell and assess whether the proof number of an AND-node exceeds a certain value. We add new candidate moves when it is appropriate. Proof number represents that proving nodes must exceed a certain value. We add new candidate moves when it is appropriate. Proof number represents that proving nodes must not exceed a certain value. We add new candidate moves when it is appropriate. Proof number represents that proving nodes must exist under this node. If the proof number is too big, this paper begins seeking new candidate moves.

6.2.4. Node evaluation

Node evaluation is designed to assess the node position and determine the action of generating candidate moves based on the position. Regarding the algorithm used to form threat moves, please refer to Section 5. When using CTSS seeks double-threat solution, the search speed is very fast; therefore, when performing node evaluation, our system specifically performs CTSS for double-threat moves.

7. Experiments and discussion

There are many strong state-of-the-art Connect6 programs, such as Cloudict.Connect6, Morethanfive, NCTU6, and TD6 [17,18,33,37]. This paper does not compare the performance with them, because the strength of Connect6 programs may mainly depend on the heuristic and evaluation functions and we do not have their functions.

This paper only describes the experiments on bitboard design and Multistage PNS. We ignore that of the algorithms for generating threat moves. The algorithm for generating threat move can decrease the time complexity from \(O(n^2)\) to \(O(n)\), where \(n\) is number of Attack Connections. Therefore, a program with the algorithm for generating threat move is surely fast than that of the known traditional methods on Connect6.

This paper gathered all the puzzles of the two main puzzle sets (2007 series and 2008 series) from the Taiwanese Connect6 website [20], and examines 58 puzzles as the experimental test benchmark for the proposed algorithms. The benchmark is basically the same as in [38] except the extra added puzzles in 2007 series. Those puzzles are the most up-to-date puzzles published on the website. Among the 58 puzzles, this paper excludes 2007-Q4-1-5, 2008-Q1-1-3, and 2008-Q3-1-4 because they lack the T2 or TSS solution. The remaining 55 puzzles are divided into two types, T2 and TSS solutions. The experiments were performed on Intel 2.0 GHz machine with 2 GB memory.

7.1. Analysis of the bitboard knowledge base system

The system computes all diversity of connection patterns that may occur in advance, and stores them in a knowledge base. In searching, the system does not need to compute the information of a Connection because the information can be retrieved by looking it up in the knowledge base. It is obvious that the bitboard design can conduct lots of computation in advance. The time of update and access for information of a cell is constant. Besides, the system also supports some very efficient Bitwise operation algorithms for Connect6.

For the comparison, we use 2-stage PNS (Type II) as the search algorithm and the puzzles which solution can be found by the search algorithm (total 24 puzzles of 2008 series in Tables 7 and 8). Table 5 shows the result, and it shows that the improved
efficiency is about 10 times. Therefore, bitboard knowledge base significantly improves the search efficiency.

7.2. Multistage PNS experimental results

The accuracy and efficiency of the Multistage PNS are analyzed in this section. Since Multistage PNS is a best-first search, it requires of maintaining the whole search tree in memory. Thus, many variations were proposed to avoid this problem [5, 21, 22, 34]. In this paper, the total memory space of the Multistage PNS tree is restricted to 60,000 nodes. This number is enough to solve most puzzles in the experiments. The limit time for each side in a game is 30 min in most tournaments. The restricted number is also enough to let Kavalan finish each game in those tournaments. The 55 test puzzles contains T2 solution and TSS solution puzzles. Five algorithms are implemented for performing the comparison. The first and second algorithms are brute-force algorithms: depth-first search (DFS) and breadth-first search (BFS). Basic DFS has no depth limit, but the outcome is bad for these puzzles with T2 and TSS solutions. Therefore, this paper tests the results of limiting the search depth of DFS to 14. Besides, the search of DFS and BFS is limited to threat moves, and excludes other candidate moves (non-threat moves). The third algorithm is Standard PNS. It generates candidate moves by using heuristic knowledge to order empty cells and uses them to generate moves. The fourth algorithm is type-I 2-stage PNS, and generates double and single-threat moves in stages. Finally, the fifth algorithm is type-II 2-stage PNS, and does not especially distinguish the threat moves.

This paper limits the memory space of search tree developed from the five algorithms to 60,000 nodes, the number of probing cells to 12. The information of the five algorithms is as in Table 6.

In the experiment, the bitboard knowledge base system is used in every algorithm. The search time and the number of searching nodes (inside the parenthesis) of the results are listed in Tables 7 and 8. The unit of time is second. The searching nodes include the nodes used in CTSS for node evaluation as described in Section 6.2.4. If the CTSS fails, the memory space of the nodes in the CTSS will be released. Thus, the total number of searching nodes may be larger than 60,000. The experimental results of 2-Stage MCTS (Type-II) are from [38] except the puzzles of 2008 series.

From the experimental results, the search efficiency of brute-force algorithms is low. They cannot find the solution on most TSS solution puzzles, especially DFS. Therefore, the brute-force algorithms are only suitable on simple puzzles. In comparison to the success rate of finding the solution of puzzles, standard PNS is worse than 2-stage PNS. Thus, standard PNS is not suitable for sudden-death game.

For the three 2-stage algorithms, Type-II 2-stage MCTS is generally better in Table 7. However, the difference is not significant because the state space of the T2 solution is small.

Table 5
The efficiency of the knowledge base system.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Type-II Multistage PNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time</td>
<td>(24 puzzles of 2008 series)</td>
</tr>
</tbody>
</table>

- Use the system: 474,367 (s)
- Does not use the system: 4512,006 (s)

Table 6
The search strategy of different algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Search strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>• DFS with depth limit to 14.</td>
</tr>
<tr>
<td></td>
<td>• When DFS searches T2 solution, it focuses the candidate moves on double-threat moves.</td>
</tr>
<tr>
<td></td>
<td>• When DFS searches TSS solution, it generates double and single-threat moves at the same time.</td>
</tr>
<tr>
<td>BFS</td>
<td>• When BFS searches T2 solution, it focuses the candidate moves on double-threat moves.</td>
</tr>
<tr>
<td></td>
<td>• When BFS searches TSS solution, it generates double and single-threat moves at the same time.</td>
</tr>
<tr>
<td>Standard PNS</td>
<td>• Using empty cell score to generate candidate moves</td>
</tr>
<tr>
<td></td>
<td>• If the proof number of an OR-Node is bigger than 66, it adds new candidate moves.</td>
</tr>
<tr>
<td>Type-I 2-stage PNS</td>
<td>• Generating single-threat moves after double-threat moves search fail.</td>
</tr>
<tr>
<td>Type-II 2-stage PNS</td>
<td>• Generating double-threat and single-threat moves at the same time.</td>
</tr>
</tbody>
</table>

52 S.-J. Yen et al. / Knowledge-Based Systems 34 (2012) 43–54
The paper describes how to generate threat moves efficiently. Suppose efficiency is improved by about 10 times in Connect6. Secondly, this algorithms is proposed. The experimental result shows that the efficiency for Connect6 suggest that Multistage Search has satisfactory performance for sudden-death games. Although this paper only uses a primitive version of PNS to develop the Multistage PNS, the proposed Multistage PNS may be combined with other variations of PNS to improve the efficiency.

For these puzzles with TSS solution, the two 2-stage PNS algorithms do not differ significantly from each other. However, in some special game positions including numerous double-threat moves without any T2 solution, an excessive amount of time is wasted in the first stage of Type-I 2-stage PNS.

In Table 8, in comparison to Type-II 2-stage MCTS, 2-stage PNS performs better for 16 of the given 31 puzzles, but worse for 10 of the others. It is hard to conclude which one is better. The reason is similar to the anomaly phenomenon mentioned in [34]. When solving some puzzles or playing in a competition with a multi-core computer in practice, a possible solution is to use the two search algorithms simultaneously.

8. Conclusions

Connect6 is an interesting board game. This paper proposes a bitboard knowledge base system and an efficient search architecture for Connect-k games. The proposed methods have been implemented in a Connect-k program Kavalan.

We summarize the conclusions of this study as follows: Firstly, a bitboard knowledge base system with bitwise operation algorithms is proposed. The experimental result shows that the efficiency is improved by about 10 times in Connect6. Secondly, this paper describes how to generate threat moves efficiently. Suppose that the number of Attack-Connections is $n$ for a game position, this algorithm significantly lowers the time complexity of generating threat moves from $O(n^2)$ to $O(n)$.

Finally, a 2-stage PNS is proposed. The experimental results on Connect6 suggest that Multistage Search has satisfactory performance for sudden-death games. Although this paper only uses a primitive version of PNS to develop the Multistage PNS, the proposed Multistage PNS may be combined with other variations of PNS to improve the efficiency.

Acknowledgments

The authors thank anonymous reviewers for their valuable comments, and thank the National Science Council of the Republic of China (Taiwan) for financial support of this research under contract numbers NSC 96-2628-E-259-020-MY3 and NSC 98-2221-E-259-021-MY3.

References


