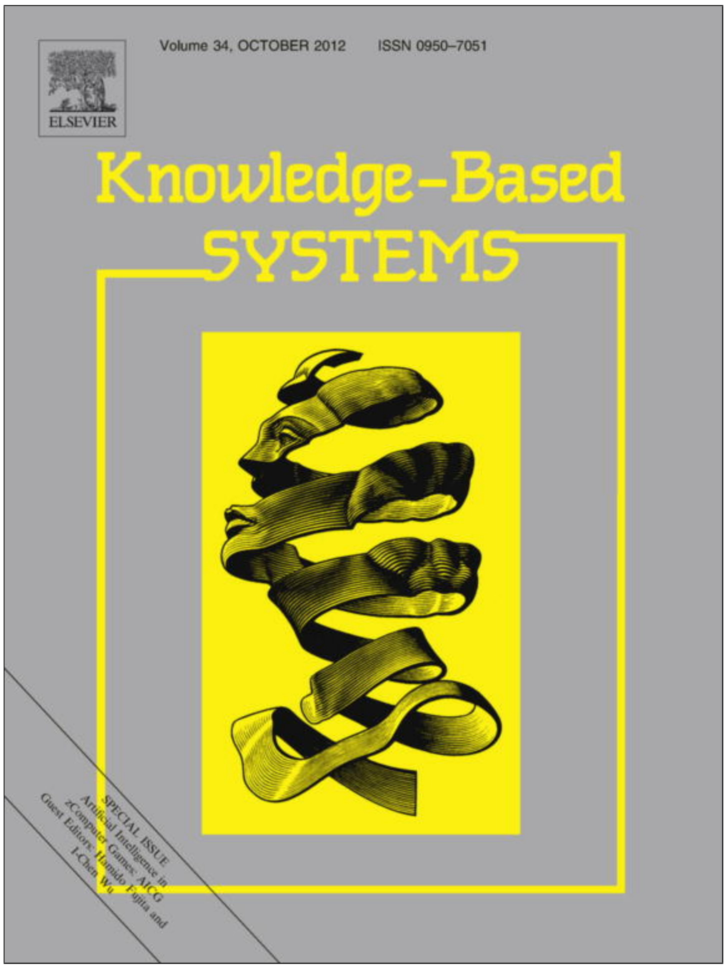

Volume 34, OCTOBER 2012 ISSN 0950-7051
ELSEVIER
Knowledge-Based SYSTEMS
SPECIAL ISSUE
Artificial Intelligence in Computer Games: AICG
Guest Editors: Hamido Fujita and I-Chen Wu

Knowledge-Based Systems 34 (2012) 43–54

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Bitboard knowledge base system and elegant search architectures for Connect6

Shi-Jim Yen [a,*], Jung-Kuei Yang [b], Kuo-Yuan Kao [c], Tai-Ning Yang [d]

[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*

A R T I C L E I N F O

Article history:
Available online 25 May 2012

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

A B S T R A C T

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.

Crown Copyright © 2012 Published by Elsevier B.V. All rights reserved.

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 $m \times n$ 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 [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

* Corresponding author. Tel.: +886 3 8634031; fax: +886 3 634010.
E-mail addresses: sjyen@mail.ndhu.edu.tw (S.-J. Yen), jungkuei@gmail.com (J.-K. Yang), stone@npu.edu.tw (K.-Y. Kao), tnyang@faculty.pccu.edu.tw (T.-N. Yang).

0950-7051/$ - see front matter Crown Copyright © 2012 Published by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2012.05.002

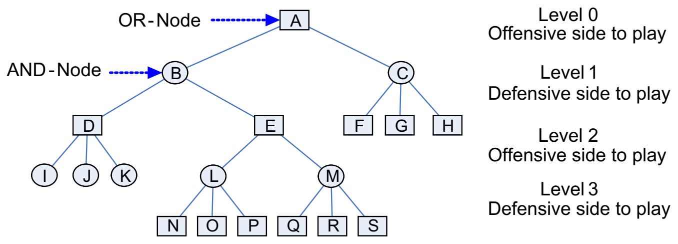


Fig. 1. An AND/OR search tree. The rectangle represents an OR-Node (Offensive side), and the circle represents an AND-Node (Defensive side).

extremely large. The concept of multistage search is to control the timing of the generating different candidate moves. In any game position, candidate moves are classified to various types, and are generated in multistage search based on the types. In Connect6, those types are *double-threat*, *single-threat* and *non-threat*. Numerous differences exist among the types. For example, in the case of double-threat moves, because Defender requires two stones to block threats, it has fewer branching factors. Thus, search of double-threat is faster than that of other types.

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.

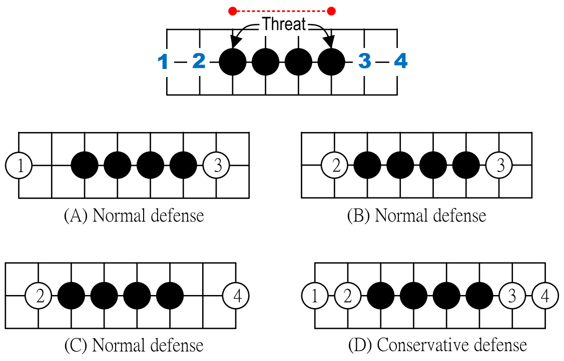


Fig. 2. The double-threat pattern and its blocking stones. (A), (B), and (C) refer to the normal defense, while (D) refers to the conservative defense.

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

Table 1
Rules which PNS uses to select the most-proving node.

Rules for AND-node	Rules for OR-node
$pn(n) = \sum_{s \in successor(n)} pn(s)$	$pn(n) = \min_{s \in successor(n)} \{pn(s)\}$
$dn(n) = \min_{s \in successor(n)} \{dn(s)\}$	$dn(n) = \sum_{s \in successor(n)} dn(s)$

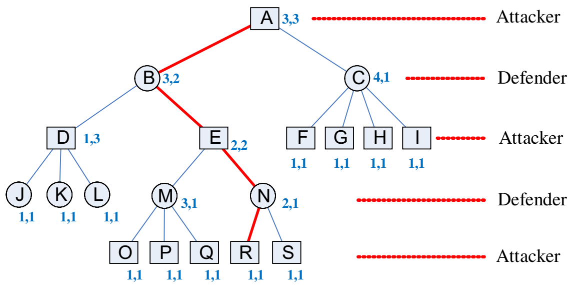


Fig. 3. PNS tree. The numbered pair comprises the proof (left) and disproof (right) numbers.

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* (I 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 of cells are represented by $I_{(x,y)}$. Fig. 4A shows the board coordinates of cells in a 9×9 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.

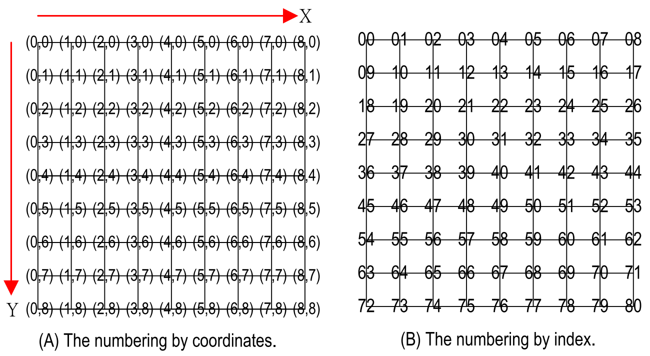


Fig. 4. Board coordinates and indexes of cells in a 9×9 board.

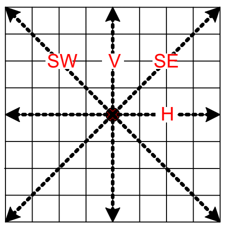


Fig. 5. The four directions on a board.

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:

$$\begin{cases} n + x - y, & \text{if } (x - y) + n < Min\{m, n\} \\ Min\{m, n\} & \text{if } Min\{m, n\} \leqslant (x - y) + n \leqslant Max\{m, n\} \\ m - (x - y), & \text{if } Max\{m, n\} < (x - y) + n \end{cases}$$

Slash east Lines:

$$\begin{cases} x+y+1, & \text{if } (x+y) < Min\{m,n\} - 1 \\ Min\{m,n\} & \text{if } Min\{m,n\} - 1 \leqslant (x+y) \leqslant Max\{m,n\} - 1 \\ Min\{m,n\} - ((x+y) - Max\{m,n\} + 1), & \text{if } Max\{m,n\} - 1 < (x+y) \end{cases}$$

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, “000111100” and “0111100”, have the same encoding value, 60, but the two shapes are different to 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 is calculated by the information of its Connections. 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 $2^{20} - 2^6 - 1$.

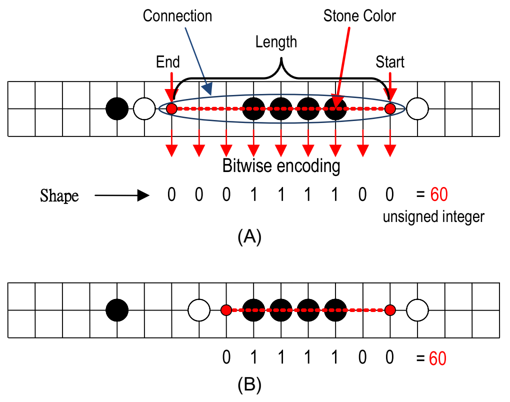
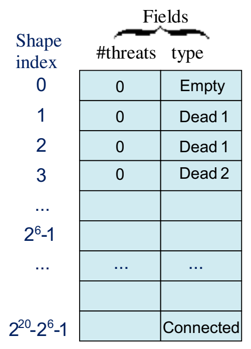


Fig. 6. The shape code of the shape in the Connection.



Shape index	#threats	type
0	0	Empty
1	0	Dead 1
2	0	Dead 1
3	0	Dead 2
…		
2^6-1		
…	…	…
2^{20}-2^6-1		Connected

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.

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 2
The Line index and mapping functions of a cell.

Direction	length	Line index	Line index of cell (x, y)
Vertical	m	From 0 to $m - 1$	x
Slash west	$m + n - 2k + 1$	From m to $2m + n - 2k$	$m + n + x - y - k$
Horizontal	n	From $2m + n - 2k + 1$ to $2(m + n - k)$	$2m + n - 2k + y + 1$
Slash east	$m + n - 2k + 1$	From $2(m + n - k) + 1$ to $3m + 3n - 4k + 1$	$2(m + n - k) + x + y + 1$

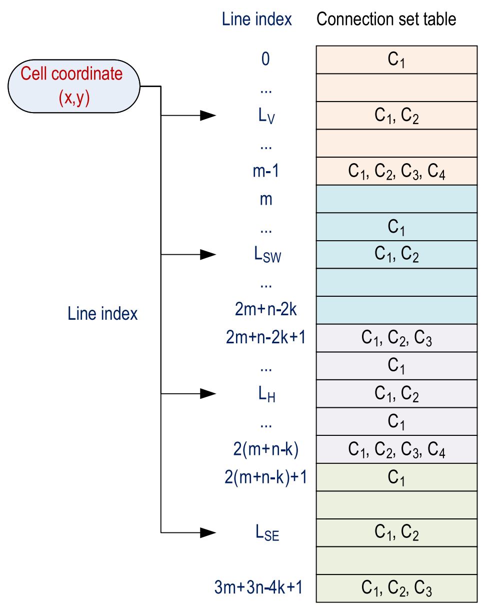


Fig. 8. Illustration of the Connection set table and the Line index.

knowledge base, three bitwise operation algorithms are proposed to improve three basic Connect6 functions: modifying Connections after playing a stone, checking whether the game is over, and 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

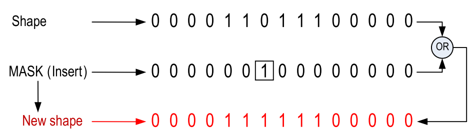


Fig. 9. Modifying a Connection for an updated cell with the same color. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

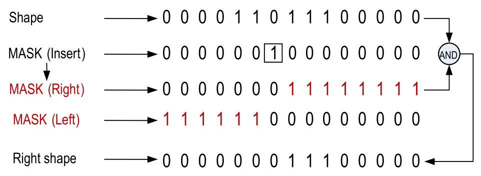


Fig. 10. Modifying a Connection for an updated cell with different color. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

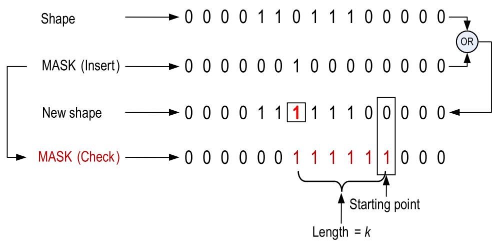


Fig. 11. Checking whether the game is over.

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, from the starting point to the terminal-cell of the Connection, we make a bitmask whose encoding length of “1” is k and the other cells are encoded as “0” as showed in Fig. 11.

The checking must cooperate with a bitwise operator “AND”. The return value is calculated by the AND operation on the shape and the bitmask. If the return value is equal to the value of bitmask, the Connection has a consecutive k stones in this segment. The algorithm takes turns to use this inspection method to finish the whole checking process, and the number of checking is k times in a Connection at most. The algorithm is as follows:

```
Step 1:    Prepare the MASK (Check) to do checking.
Step 2:    if ((new shape & MASK (Check))==MASK (Check))
             The game is over;
           else
             MASK (Check) ≪ = 1;
```

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

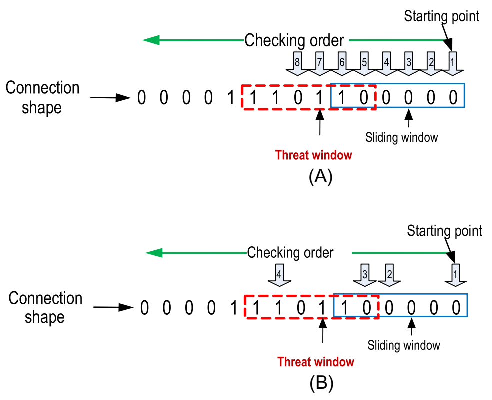


Fig. 12. The concept for counting the number of threats in a shape.

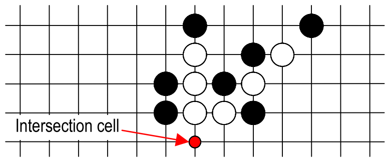


Fig. 13. Intersection cell of two Connections. White can use a single stone to make two threats according to the two dead-3 Connections.

sum of stones checked in a sliding window. The number of jumping equals to the number of stones to form a threat window minus the sum of stones in a sliding window and minus one. Take Fig. 12B as an example. In the round one of sliding window checking processes, the starting point is the first cell as in Fig. 12B. In the first sliding window, there is only one stone; therefore, the second checking point is the fourth cell. The reason is that it is impossible to meet a threat window inside two rounds of checking. Therefore, rounds two and three are omitted in Fig. 12A.

Secondly, if the inspection meets a threat window, we go onto check behind the farthest empty cell inside the Threat Window. In Fig. 12B, the inspection in round three meets the threat window, so the next checking point is that of the round four as in this figure. In Wu's algorithm, when sliding window becomes threat window, it marks all the empty squares in the window. It means that it does not jump over the threat window.

5. Generating threat move

As mentioned in [30–32], threats are the key to play Connect6 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.

Table 3
Patterns, Threat-Cells, and Threat-Pairs.

Pattern	The pattern after adding one stone	Set of used Threat-Cell	The pattern after adding two stones	Set of used Threat-Pair
Live-3	Live-4	T2-Cell set	Live-5	T2-Pair set
	Dead-4	T1-Cell set	Dead-5	T1-Pair set
Dead-3	Dead-4	T1-Cell set	Live-5	T2-Pair set
			Dead-5	T1-Pair set
Live-2			Live-4	T2-Pair set
			Dead-4	T1-Pair set
Lead-2			Dead-4	T1-Pair set

- 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.
- Indirect-Attack-Connection: The Connection is neither a Threat-Connection nor a 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.

The general threat move generating method is to check the patterns of the Attack-Connections in a game position. For example, given a[1]live-2 Attack-Connection, Attacker can generate a live-4 Threat-Connection. When search all the threat moves on the board, if multiple Attack-Connections exist, it is necessary to consider whether an *intersection cell* 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 *T1-Cell*. If a Threat-Cell can form two threats, the cell is labeled *T2-Cell*. *Tn-Cell set* is the set of Tn-Cell.

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 *T1-Pair*. If the number of threats formed by a Threat-Pair equals two, the Threat-Pair is labeled the *T2-Pair*. *Tn-Pair set* is the set of *Tn*-Pairs.

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,

[1] Live-2 is a Connect6 pattern, and so as live-n and dead-n, where n is between 1 and 5 [30].

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 win-
            ning move and thus ending the search.
        (e) Examine the intersection cell of this T1-Cell set with
            the formerly generated T1-Pair 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 individ-
        ual 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-Connec-
            tion, 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 S_1 and S_2, for any two different cells $c_1 \in S_1$ and $c_2 \in S_2$, Attacker can win by playing on c_1 and c_2.*

Proof. Because the T2-Cell can forms two threats using a single stone, place stones on c_1 and c_2 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 S_1 and T2-Cell set S_2, if there is an intersection cell $c_x$of the two sets, Attacker can win by playing on the c_x.*

Proof. The cell $c_2 \in S_2$ can generate the double-threat pattern, and $c_1 \in S_1$ can the generate single-threat pattern. Therefore, c_1 and c_2 can generate three threats. However, if c_x is empty after Defender plays the two stones, Defender may block the three threats by playing on c_x and another stone. □

Property 3. *Given the two T1-Cell sets S_1 and S_2, for any two different cells $c_1 \in S_1$ and $c_2 \in S_2$, Attacker can make a double-threat pattern by playing on the c_1 and c_2.*

Property 4. *Given the T2-Cell set S_c and T2-Pair set S_p, for any cell $c_1 \in S_c$ and any cell pair $(c_{p1}, c_{p2}) \in S_p$, if c_1 is equal to either c_{p1} or c_{p2}, Attacker can win by playing on the c_{p1} and c_{p2}.*

Property 5. *In Property 4, if S_p is a T1-Pair set, Attacker can also win by playing on the c_{p1} and c_{p2}.*

Property 6. *In Property 4, if S_c is a T1-Cell set, Attacker can also win by playing on the c_{p1} and c_{p2}.*

Property 7. *In Property 4, if S_c is a T1-Cell set and S_p is a T1-Pair set, Attacker can make a double-threat pattern by playing on the c_{p1} and c_{p2}.*

Property 8. *It is impossible for the T2-Pair or T1-Pair sets share the same intersection Pair.*

Proof. 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 sets 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, as the two special dead-4 in Fig. 14.

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.

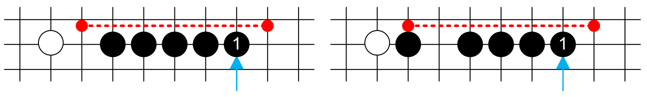


Fig. 14. One stone can transform the dead-4 into a live-5.

Table 4
The kinds of moves in different Multistage Searches.

Strategy	Stage		
	First stage	Second stage	Third stage
1-Stage	Double-threat Single-threat Non-threat		
Type-I 2-stage	Double-threat	Single-threat Non-threat	
Type-II 2-stage	Double-threat Single-threat	Non-threat	
3-Stage	Double-threat	Single-threat	Non-threat

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^2, 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. 15 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 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. 16, 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 numbers 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.

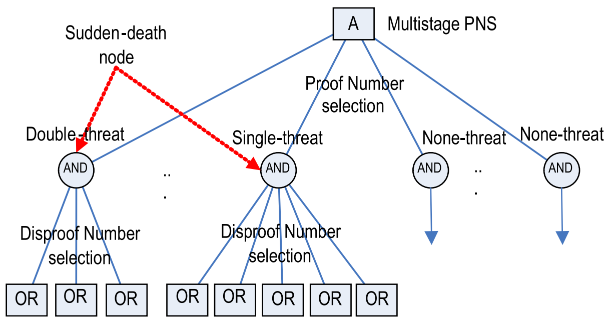


Fig. 15. Multistage Proof Number Search architecture.

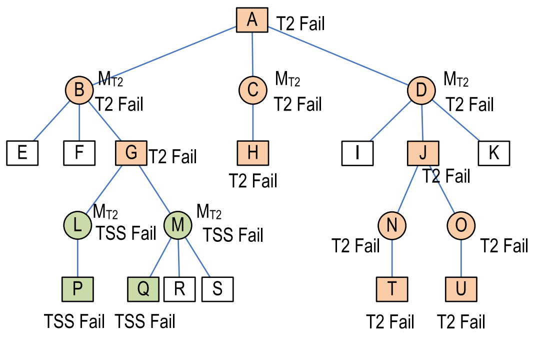


Fig. 16. First stage – double-threat search.

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 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

Table 5
The efficiency of the knowledge base system.

	Type-II Multistage PNS Total time (24 puzzles of 2008 series)
Use the system	474.367 (s)
Does not use the system	4512.006 (s)

Table 6
The search strategy of different algorithms.

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

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 7
The result for the puzzles with T2 solution.

Puzzles	Algorithms					
	DFS with depth limit to 14	BFS	Standard PNS	Type-I 2-stage PNS	Type-II 2-stage PNS	Type-II 2-stage MCTS [34]
2007-Q4-2-5	3.859 (51,458)	6.218 (65,959)	X	0.015 (238)	0.015 (238)	0.062 (238)
2007-Q4-2-6	0.031(638)	0.015 (517)	0.031 (197)	0.062 (1061)	0.062(1061)	0.062(1061)
2007-Q4-3-1	X	0.296 (4312)	0.001 (221)	0.001 (61)	0.001 (61)	0.001 (61)
2007-Q4-3-4	X	0.687 (9058)	0.015 (500)	0.015 (517)	0.015 (517)	0.031 (517)
2007-Q4-3-5	X	6.000 (60,901)	0.015 (154)	0.001 (131)	0.001 (131)	0.001 (131)
2007-Q4-3-6	0.156 (2480)	0.001 (104)	0.001 (23)	0.001 (22)	0.001 (22)	0.015 (22)
2007-Q4-4-4	0.015 (134)	0.001 (224)	0.001 (119)	0.001 (43)	0.001 (43)	0.001 (43)
2007-Q4-4-6	X	8.046 (70,102)	X	0.171 (3024)	0.421 (3681)	1.046 (4911)
2007-Q4-5-1	0.046 (892)	0.001 (72)	18.156 (147,298)	0.001 (144)	0.031 (144)	0.015 (144)
2007-Q4-5-2	X	0.001 (238)	0.001 (58)	0.001 (31)	0.001 (31)	0.015 (31)
2007-Q4-5-3	0.015 (334)	0.078 (982)	0.001 (42)	0.001 (38)	0.001 (38)	0.015 (38)
2007-Q4-5-4	0.001 (56)	0.001 (247)	0.001 (55)	0.001 (30)	0.001 (30)	0.001 (30)
2008-Q1-1-1	0.219 (4785)	0.016 (202)	0.001 (78)	0.016 (84)	0.001 (84)	0.001 (84)
2008-Q1-1-2	0.391 (8062)	11.469 (70,263)	X	0.031 (554)	0.031 (554)	0.031 (554)
2008-Q1-2-1	0.984 (23,071)	0.078 (884)	0.001 (97)	0.001 (89)	0.001 (89)	0.001 (89)
2008-Q1-2-2	2.906 (70,184)	0.953 (10,003)	0.094 (1311)	2.921 (4475)	0.265 (4475)	0.406 (4475)
2008-Q1-3-1	4.281 (103,946)	0.078 (705)	0.031 (610)	0.047 (563)	0.047 (563)	0.031 (563)
2008-Q1-3-2	5.250 (120,907)	0.031 (581)	12.890 (105,130)	0.001 (27)	0.016 (27)	0.001 (27)
2008-Q2-1-1	1.203 (26,558)	1.672 (15,734)	X	0.359 (54)	0.438 (54)	0.001 (54)
2008-Q2-1-2	1.719 (43,917)	3.062 (15,661)	0.453 (67)	0.172 (50)	0.188 (50)	0.001 (50)
2008-Q2-2-1	0.046 (914)	0.047 (487)	0.313 (48)	0.219 (56)	0.266 (56)	0.016 (56)
2008-Q2-2-2	0.609 (13,320)	0.250 (2263)	0.219 (46)	0.188 (29)	0.172 (29)	0.001 (29)
2008-Q3-1-1	9.813 (233,264)	0.016 (168)	6.156 (1610)	0.234 (1200)	0.359 (1200)	0.031 (208)
2008-Q3-1-2	0.001 (26)	0.031 (202)	0.234 (54)	0.172 (18)	0.172 (18)	0.001 (18)

Table 8
The result for the puzzles with TSS solution.

Puzzles	Algorithms					
	DFS with depth limit to 14	BFS	Standard PNS	Type-I 2-stage PNS	Type-II 2-stage PNS	Type-II 2-stage MCTS [34]
2007-Q4-1-1	X	X	X	X	X	X
2007-Q4-1-2	X	18.500 (310,280)	1.359 (24,551)	1.625 (28,225)	**0.265 (4203)**	0.390 (3384)
2007-Q4-1-3	X	X	9.500 (93,583)	19.453 (230,023)	**3.937 (52,168)**	78.031 (781,178)
2007-Q4-1-4	X	X	X	X	**48.406 (539,170)**	X
2007-Q4-2-1	X	X	X	X	X	**12.078 (102,098)**
2007-Q4-2-2	X	X	X	30.171 (355,780)	X	**5.828 (54,794)**
2007-Q4-2-3	X	X	X	X	38.687 (359,600)	**0.812 (9163)**
2007-Q4-2-4	X	36.562 (613,438)	0.640 (14,453)	7.625 (83,954)	**0.609 (9467)**	0.968 (12,752)
2007-Q4-3-2	X	63.500 (929,075)	X	0.484 (10,911)	**0.187 (1967)**	1.437 (10,608)
2007-Q4-3-3	X	0.921 (15,966)	X	6.125 (90,060)	**0.546 (15,855)**	4.468 (99,971)
2007-Q4-4-1	X	149.062 (2,518,898)	X	1.390 (30,094)	7.093 (89,261)	**0.609 (2154)**
2007-Q4-4-2	X	X	X	413.296 (4,973,584)	58.859 (640,402)	**5.156 (40,317)**
2007-Q4-4-3	X	X	X	X	X	X
2007-Q4-4-5	X	X	149.812 (1,491,658)	171.640 (1,963,234)	**2.281 (16,379)**	6.375 (25,369)
2007-Q4-5-5	X	X	11.484 (114,332)	X	X	**12.281 (129,631)**
2008-Q1-1-4	X	X	X	3.469 (62,209)	**2.953 (32,299)**	3.047 (32,767)
2008-Q1-1-5	X	X	X	X	X	X
2008-Q1-2-3	X	X	X	X	X	**173.938 (1,224,641)**
2008-Q1-2-4	6.015 (100,011)	0.001 (152)	0.016 (197)	0.015 (154)	**0.001 (154)**	**0.001 (154)**
2008-Q1-2-5	X	246.828 (3,916,209)	660 (5,013,875)	0.172 (3297)	**0.203 (3684)**	1.267 (14,538)
2008-Q1-3-3	X	99.421 (1,362,512)	8.671 (77,554)	324.515 (2,226,606)	324.266 (2,589,807)	**178.875 (1,697,174)**
2008-Q1-3-4	X	X	X	X	**48.750 (329,571)**	86.063 (399,731)
2008-Q1-3-5	X	X	X	X	X	**1912.343 (16,786,491)**
2008-Q2-1-3	X	7.703 (129,573)	X	1.016 (16,717)	**0.375 (5786)**	2.734 (20,458)
2008-Q2-1-4	X	12.281 (199,448)	0.250 (3407)	0.375 (4488)	**0.140 (1175)**	3.063 (11,941)
2008-Q2-1-5	X	X	X	128.047 (1,556,262)	**11.937 (154,639)**	X
2008-Q2-2-3	X	X	51.781 (512,988)	0.375 (6689)	**4.000 (18,935)**	X
2008-Q2-2-4	X	5.468 (171,519)	X	0.250 (4751)	**0.078 (1810)**	0.141 (2217)
2008-Q2-2-5	X	X	X	X	X	X
2008-Q3-1-3	X	14.765 (384,205)	0.172 (3258)	22.687 (143,071)	**0.500 (8258)**	0.563 (4528)
2008-Q3-1-5	X	X	X	109.594 (1,341,419)	89.266 (901,372)	**8.797 (81,888)**

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 Connect6 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

[1] H. Akiyama, K. Komiya, Y. Kotani, Nested Monte-Carlo search with simulation reduction, Knowledge-Based Systems (2011), http://dx.doi.org/10.1016/j.knosys.2011.11.015.
[2] L.V. Allis, Searching for solutions in games and artificial intelligence, Ph.D. Thesis, University of Limburg, Maastricht, The Netherlands, 1994.
[3] L.V. Allis, H.J. van den Herik, M.P.H. Huntjens, Go-Moku solved by new search techniques, Computational Intelligence 12 (1) (1996) 7–23.
[4] L.V. Allis, M. van der Meulen, H.J. van den Herik, Proof-number search, Artificial Intelligence 66 (1) (1994) 91–124.
[5] D.M. Breuker, J. Uiterwijk, H.J. van den Herik, The PN2-search algorithm', in: H.J. van den Herik, B. Monien (Eds.), Advances in Computer Games, IKAT, Universiteit Maastricht, Maastricht, The Netherlands, 2001, pp. 115–132.
[6] K.-H. Chen, Dynamic randomization and domain knowledge in Monte-Carlo tree search for Go knowledge-based systems, Knowledge-Based Systems (2011), http://dx.doi.org/10.1016/j.knosys.2011.08.007.
[7] R. Coulom, Efficient selectivity and backup operators in Monte-Carlo tree search, in: H.J. van den Herik, P. Ciancarini, H.J. Donkers (Eds.), Proceedings of the 5th International Conference on Computer and Games, Lecture Notes in Computer Science (LNCS 4630), 2007, pp. 72–83.
[8] S. Gelly, D. Silver, Monte-Carlo tree search and rapid action value estimation in computer Go, Artificial Intelligence 175 (11) (2011) 1856–1875.
[9] R. Grimbergen, Using bitboards for move generation in Shogi, ICGA Journal 30 (1) (2007) 25–34.
[10] E.A. Heinz, How dark thought plays chess, ICCA Journal 20 (3) (1997) 166–176.
[11] H.J. van den Herik, M.H.M. Winands, Proof-number search and its variants, Oppositional Concepts in Computational Intelligence (2008) 91–118.
[12] R. Hyatt, Rotated bitmaps, a new twist on an old idea, ICCA Journal 22 (4) (1999) 213–222.
[13] A. Kishimoto, Y. Kotani, Parallel AND/OR tree search based on proof and disproof numbers, in: Proceedings of the 5th Games Programming Workshop, IPSJ Symposium Series, vol. 99(14), 1999, pp. 24–30.

[14] A. Kishimoto, M. Müller, Search versus knowledge for solving life and death problems in Go, in: Twentieth National Conference on Artificial Intelligence (AAAI-05), 2005, pp. 1374–1379.
[15] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, (Eds.), Proceedings of the 17th European Conference on Machine Learning (ECML), 2006, pp. 282–293.
[16] C.-S. Lee, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-J. Yen, M.-H. Wang, Current frontiers in computer Go, IEEE Transactions on Computational Intelligence and AI in Games 2 (4) (2010) 229–238.
[17] P.-H. Lin, I-C. Wu, NCTU6 wins in the man–machine Connect6 championship 2009, ICGA Journal 32 (4) (2009) 230–232.
[18] H.-H. Lin, D.-J. Sun, I-C. Wu, S.-J. Yen, TAAI computer game tournament report in 2010, ICGA Journal 34 (1) (2011) 51–54.
[19] P. San Segundo, R. Galan, D. Rodriguez-Losada, F. Matia, A. Jimenez, Efficient search using bitboard models, in: Proceedings XVIII International Conference on Conference on Tools for AI, Washington, 2006, pp. 132–138.
[20] Taiwan Connect6 Association, Connect6 homepage, <http://www.connect6.org/>.
[21] A. Nagai, Df-pn algorithm for searching AND/OR trees and its applications, PhD thesis, University of Tokyo, Japan, 2002.
[22] J. Pawlewicz, L. Lew, Improving depth-first PN-search: 1+ε trick, 5th International Conference on Computers and Games, in: H.J. van den Herik, P. Ciancarini, H.H.L.M. Donkers (Eds.), Lecture Notes in Computer Science (LNCS 4630), Computers and Games, Springer, Heidelberg, 2007, pp. 160–170.
[23] F. Reul, New architectures in computer chess, Ph.D. thesis, Tilburg University, Tilburg, The Netherlands, 1990.
[24] J.T. Saito, M.H.M. Winands, H.J. van den Herik, Randomized parallel proof-number search. Advances in computer games conference (ACG'12), in: H. Jaap van den Herik, Pieter Spronck, (Eds.), Lecture Notes in Computer Science (LNCS 6048), Palacio del Condestable, Pamplona, Spain, 2010, pp. 75–87.
[25] M.P.D. Schadd, M.H.M. Winands, M.J.W. Tak, J.W.H.M. Uiterwijk, Single-player Monte-Carlo tree search for SameGame, Knowledge-Based Systems (2011), http://dx.doi.org/10.1016/j.knosys.2011.08.008.
[26] J. Schaeffer, N. Burch, Y.N. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, S. Sutphen, Checkers is solved, Science 5844 (317) (2007) 1518–1552.
[27] M. Seo, H. Iida, J.W.H.M. Uiterwijk, The PN*-search algorithm: application to tsumeshogi, Artificial Intelligence 129 (1-2) (2001) 253–277.
[28] S. Tannous, Avoiding rotated bitboards with direct lookup, ICGA Journal 30 (2) (2007) 85–91.
[29] M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, PDS-PN: a new proof-number search algorithm: application to Lines of action. Computers and Games 2002, in: J. Schaeffer, M. Müller and Y. Björnsson (Eds.), Lecture Notes in Computer Science (LNCS 2883), Computers and Games, Springer, Heidelberg, 2003, pp. 61–74.
[30] I-C. Wu, D.-Y. Huang, H.-C. Chang, Connect6, ICGA Journal 28 (4) (2005) 235–242.
[31] I-C. Wu, D.-Y. Huang, A new family of *k*-in-a-row games, the 11th Advances in Computer Games Conference (ACG 2005), in: H. Jaap van den Herik, Shun-Chin Hsu, Tsan-sheng Hsu, H.H.L.M. Donkers (Eds.), Proceedings of the 11th Computers and Games, Lecture Notes in Computer Science (LNCS 4250), 2006, pp. 180–194.
[32] I-C. Wu, P.-H. Lin, Relevance-zone-oriented proof search for Connect6, IEEE Transactions on Computational Intelligence and AI in Games 2 (3) (2010) 191–207.
[33] I-C. Wu, P.-H. Lin, S.-J. Yen, Morethanfive wins Connect6 tournament, ICGA Journal 33 (3) (2010) 179–180.
[34] I-C. Wu, H.-H. Lin, P.-H. Lin, D.-J. Sun, Y.-C. Chan, B.-T. Chen, Job-level proof number search for Connect6, in: J. van den Herik, H. Iida, A. Plaat (Eds.), Proceeding of the Computers and Games 2010, Lecture Notes in Computer Science (LNCS 6515), 2011, pp. 11–22.
[35] C.-M. Xu, Z.-M. Ma, X.-H. Xu, A method to construct knowledge table-base in *k*-in-a-row games, in: Proceedings of the 2009 ACM Symposium on Applied Computing, 2009, pp. 929–933.
[36] S.-J. Yen, J.-K. Yang, The bitboard design and bitwise computing in Connect6, in: Proceedings of the 14th Game Programming Workshop, 2009, pp. 95–98.
[37] S.-J. Yen, T.-C. Su, I-C. Wu, The TCGA 2011 computer-games tournament, ICGA Journal 34 (2) (2011) 108–110.
[38] S.-J. Yen, J.-K. Yang, Two-stage Monte Carlo tree search for Connect6, IEEE Transactions on Computational Intelligence and AI in Games 3 (2) (2011) 100–118.
[39] S.-J. Yen, T.-C. Su, I-C. Wu, The TCGA 2011 computer-games tournament, ICGA Journal 34 (2) (2011) 108–110.
[40] S.-J. Yen, C.-W. Chou, C.-S. Lee, H. Doghmen, O. Teytaud, The IEEE SSCI 2011 human vs. computer-Go competition, ICGA Journal 34 (2) (2011) 106–107.