Two-Stage Monte Carlo Tree Search for Connect6

Shi-Jim Yen, *Member, IEEE*, and Jung-Kuei Yang

***Abstract*—Recently, Monte Carlo tree search (MCTS) has become a well-known game search method, and has been successfully applied to many games. This method performs well in solving search trees with numerous branches, such as Go, Havannah, etc. Connect6 is a game involving a search tree with numerous branches, and it is also one of the sudden-death games. This paper thus proposes a new MCTS variant related to Connect6, called two-stage MCTS. The first stage focuses on threat space search (TSS), which is designed to solve the sudden-death problem. For the double-threat TSS in Connect6, this study proposes an algorithm called iterative threat space search (ITSS) which combines normal TSS with conservative threat space search (CTSS). The second stage uses MCTS to estimate the game-theoretic value of the initial position. This stage aims at finding the most promising move. The experimental result shows that two-stage MCTS is considerably more efficient than traditional MCTS on those positions with TSS solution in Connect6. Furthermore, according to Connect6 heuristic knowledge, this paper uses relevance-zone search to accelerate identifying winning and losing moves.**

***Index Terms*—Board games, Connect6, conservative TSS, iterative TSS, Monte Carlo tree search, threat space search.**

I. Introduction

SEARCHING is both a method of solving problems and a means for programs to display their intelligence. When facing complex problems, computers must explore a vast number of states, which requires enormous computational time. Two means of tackling difficult problems exist in such situations. The first approach involves applying heuristic knowledge of the relevant field to decrease the search states. This approach saves considerable time on problem solving. Currently, heuristic knowledge plays a significant role in branch elimination. Only effective evaluation can correctly evaluate different game states.

The second approach involves selecting an efficient search algorithm. An effective search method can correctly guide search orientation and increase search efficiency. This can avoid unnecessary time wasting and focus the search on the optimal state space, significantly improving search performance.

Manuscript received April 15, 2010; revised July 30, 2010; accepted March 12, 2011. Date of publication April 05, 2011; date of current version June 15, 2011. This work was supported in part by the National Science Council of the R.O.C. (Taiwan) under contracts NSC 96-2628-E-259-020-MY3 and 98-2221-E-259-021-MY3.

S.-J. Yen is with the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan (e-mail: sjyen@mail.ndhu.edu.tw).

J.-K. Yang is with the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan and also with the Department of Applied Foreign Languages, Lan Yang Institute of Technology, I Lan, Taiwan (e-mail: jungkuei@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2011.2134097

Recently, Monte Carlo tree search (MCTS) has become an extremely popular game search method, and it has been applied to numerous game searches. MCTS-based programs for which good evaluation functions are hard to build, for example Go, have succeeded in [4]–[7], [11], and [12]. Moreover, MCTS is also used in other games, such as Amazons [14], Backgammon [18], lines of action (LOA) [19], Havannah [8], and Hex [3].

MCTS is based on the idea of sampling enormous branches by playout from the leaf node. Not only does MCTS sample from the branches after evaluating the position, but it also corrects mistakes in the upper sampling position by developing the correspondent branches of the search tree.

A. Connect6

Since Wu [20] investigated online board games like k-in-a-row or Connect (m, n, k, p, q) in 2005, Connect (19, 19, 6, 2, 1) or Connect6, derived from Gomoku, has been a very popular research topic [13], [21], [23]–[26], [28]. Connect6 is a fair and highly complex game with simple rules. Although the game itself is not new, its features offer a research direction.

The rules of Connect6 are very simple. Connect6 is a game for two players (black and white). Black and white take turns placing stones on a 19×19 board. Except for the first move, which is limited to placing just one stone, in subsequent moves players are allowed to place two stones simultaneously. Normally, black moves first. Because the rules do not allow player to pass, the game states change in a stable manner. Since stones cannot be killed, cell[1] states remain constant following stone placement, and thus the game state never repeats during a single game.

Because the rules of Connect6 require all moves after the first one to involve placing two stones, the search tree has an enormous number of branches. Take 19×19 board for example. Some 64 620($360 \times 359/2$) possibilities exist for the second move. As the branches are so numerous, it becomes difficult or impossible to search all positions.

Allis [2] proved that the first mover in Gomoku can always win. In the case of Connect6, each side always has one more stone on the board than their opponent. Connect6 thus is fairer than Gomoku because the first player in Gomoku has either one more stone than their opponent or the same number of stones. Additionally, thus far nobody has demonstrated that the advantage in Connect6 lies with any one side, so based on current research the game is fairer than traditional Gomoku.

B. Problem Statement and Research Questions

This paper discusses how the MCTS can be applied to Connect6, possible improvements and how strongly it can play. Hence, the research aims are as follows:

[1]The intersections to place stones on the board is called cell in this paper.

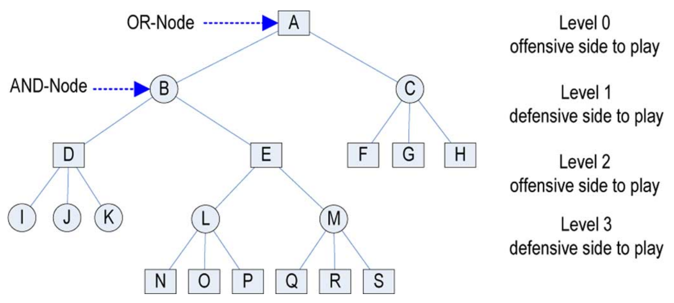
OR-Node
AND-Node
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
Level 0 offensive side to play
Level 1 defensive side to play
Level 2 offensive side to play
Level 3 defensive side to play

Fig. 1. AND/OR search tree. Rectangles represent an OR-Node, and circles represent an AND-Node.

1) an introduction to threat space search (TSS) was used in k-in-a-row games and related subjects;
2) develop a search structure that allows MCTS to be used to play Connect6;
3) discuss the heuristic knowledge that can be used to improve MCTS according to the features of Connect6;
4) analyze the accuracy and performance of the new structure developed by this paper for MCTS in Connect6.

The remainder of this paper is organized as follows. Section II discusses the TSS, which is a sudden-death feature incorporated into k-in-a-row games. Section III then proposed an algorithm for double-threat TSS (like VCDT as mentioned in [26]) in Connect6, which is termed the iterative threat space search. Subsequently, Section IV introduces MCTS implementation to Connect6. This section provides a new structure for MCTS, called two-stage MCTS. Section V introduces a heuristic knowledge that can be used in Connect6. Section VI then explains the experimental method and results. Finally, Section VII presents conclusions.

II. Background and Related Work

A. Search in Connect6

1) AND/OR Tree: AND/OR trees can be used to describe problems involving Connect6. In Connect6 search, the player who is the search target is the offensive side, and the opposing player is the defensive side. The search begins on the offensive side, which represents the initial search position, and ends when one side wins or a draw occurs.

Fig. 1 is an AND/OR search tree of Connect6. The root node is the position after the defensive side move, and is the initial searching position, which is termed level 0 (or root node) of the search tree. Based on the position of level 0, level 1 shows the moves of the offensive side, like nodes B or C, and level 1 of the search tree. Similarly, level 2 shows the moves of the defensive side based on the move of the offensive side in level 1. The search tree is formed in this manner.

The position after the move of the defensive side is an OR-Node. Among the children of this OR-Node, whenever a node is identified that demonstrates a win for the offensive side, it is proved that the offensive side achieves this victory under this OR-Node, like nodes B or C in the figure. Whenever a node (B or C) is proved victorious, the offensive side wins under the position of node A.

The position after the move of the offensive side is an AND-Node. For the offensive side, regardless of the defensive efforts of the defensive side, the offensive side must win because only then can it be seen as a winner. For example, nodes D and E in Fig. 1 are defensive moves of the defensive side under the position of node B. we have to prove that, whether in node D or E, the position where the offensive side wins can be searched. Victory for the offensive side can then be shifted to node B.

2) Candidate Moves: Forming candidate moves of Connect6 is a very difficult decision because, except for the first move, every move involves two stones. For a given position, the only candidate moves are those involving various combinations of empty cells. Thus, the amount of candidate moves at every level is very large, which will exert a strong contradictory influence on searching.

Moves that comply with the game rules are called *legal moves*. Legal moves are the basic requirements that generate candidate moves in a game searching. However, when forming moves, moves formed by considering winning or losing conditions are termed *rational moves*.

Taking Connect6 for example, if there is any threat from one side, the other side must make blocking moves, or lose the game. Moves made under these conditions are called rational moves.

It is easier to make legal moves with the knowledge of which cells on the board are empty. Rational moves are those generated based on consideration of threats in Connect6. The candidate moves presented in this paper must be based on the consideration of threats on some position, and thus correspond with the requirement that moves be rational.

B. Threat Space Search

TSS is the most common search method in Connect-k games [1], [2], [9], [17], [23]. TSS controls the candidate moves by using the situation in which Defender[2] must block the *threats* that occur after the move of Attacker.

According to the definition of the threats used by Wu [20] "*one player, say W, cannot connect six. B is said to have t threats, if and only if W needs to place t stones to prevent B from winning in the next move of B.*" This paper assumes that when one of the players makes a threat, their opponent will make a blocking move.

Therefore, the TSS features require Attacker to make a threat whenever it makes a move. For Connect6, the threat can be single or double, and this paper calls the move that can make one or two threats, *Threat-Move*. According to this concept, Attacker must select one among numerous candidate moves to fulfill this need, and this paper calls this kind of search TSS (like VCST as mentioned in [26]).

Attacker can belong to the offensive or defensive sides of the search tree. Thus, the TSS begins when one player makes threats, with that player being called Attacker. TSS aims to generate the number of threats that are bigger than the number of legal stones to play and Defender cannot block, resulting in a

[2]TSS search can be used in the offensive or defensive sides of the game tree search. To distinguish the offensive and defensive sides of the TSS-subtree and whole search tree, this paper labels the offensive side Attacker and the other side Defender in TSS.

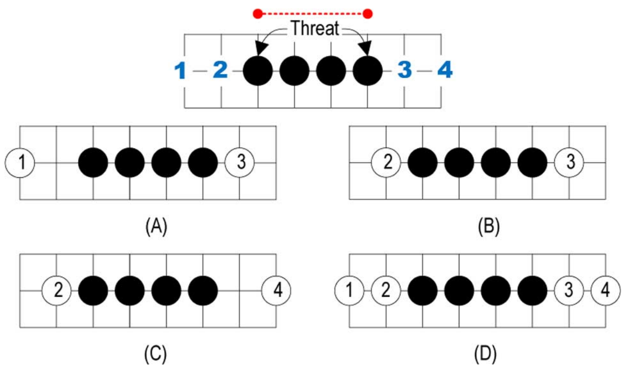

Fig. 2. Double-threat Connection (Live 4) and its defensive moves. (A), (B), and (C) denote the normal defense, while (D) denotes the conservative defense. (A) Normal defense, (B) normal defense, (C) normal defense, and (D) conservative defense.

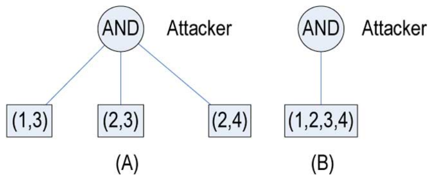

Fig. 3. Search tree of double-threat TSS. (A) denotes normal defense, and (B) represents conservative defense. In this figure, Attacker lies on the offensive side of search tree. Attacker of double-threat TSS can lie on either offensive side or defensive side of the search tree. (A) Normal defense and (B) conservative defense.

victory for Attacker, and identifying the terminal state for Attacker win. In this case, the $Value$[3] of this node is logically *True* if Attacker is on the offensive side of the AND/OR tree. This method can significantly decrease the searching states, and quickly identifies a possible winning move for the present position.

In this paper, when a position has a solution, it means that Attacker follows some winning strategy to achieve the final winning position (like triple-or-more-threat move in [26]) in all Defender blocking moves. For example, Attacker uses the strategy of continuous single-threat-or-more moves under a position to find the final winning position, and the position is said to have a TSS solution. Under a certain position, one or more solutions may exist for Attacker, but the goal of TSS is to find one solution.

1) Double-Threat TSS: Connection is a most commonly used information when searching in Connect-k games. The double-threat TSS begins after a player finds double-threat moves. Currently, Threat-Move generation uses *Connection Pattern* (or *Pattern*) to determine the Threat-Move, and the same approach is used to obtain the related defense moves. For information on *Connection Pattern*, please refer to [20], [23]; for information on saving and calculating Connection, please refer to [15], [27], [28].

When Attacker makes a double-threat move, rationality dictates that Defender must block the threats. Fig. 2 illustrates an example of a Connection involving two threats, together with different defensive moves. Three defenses exist: (A), (B), and

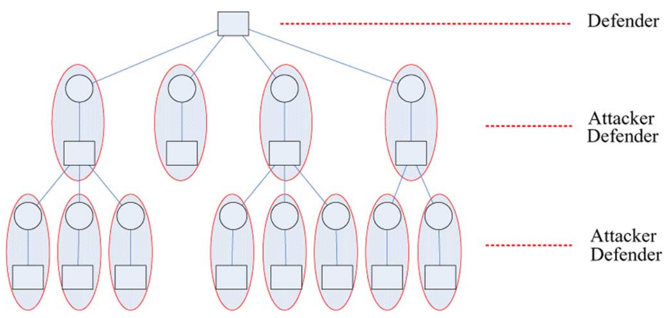

Fig. 4. Search tree structure of CTSS. The child of every Attacker is just one node. Attacker of CTSS is the offensive side of the search tree in this figure.

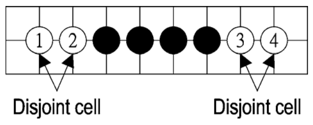

Fig. 5. Example of conservative defense.

(C). This kind of defense is termed *normal defense*[4], and the set of all normal defense moves is termed the *defense set*. The number of defense set for all kinds of blocking two threats can be assumed from the observation of Connect6.

2) Property 1: In Connect6, for double-threat moves, the defense set contains a maximum of four defensive moves.

In this paper, double-threat TSS with normal defense denotes the search method in which Defender blocks the threats of Attacker using legal moves, and Attacker responds by applying deeper double-threat TSS in response to individual defensive moves. As the search tree shows in Fig. 3(a), when Attacker threatens Defender, three different defensive moves exist, (A), (B), and (C) in Fig. 2, all of which are the children of Attacker's node, as shown in Fig. 3(a). The state space to be searched is larger in this approach because every defensive move must be searched individually, increasing the complexity of dealing with these nodes in the search tree.

In double-threat TSS, if Attacker uses the strategy of continuous double-threat-or-more moves in a position to find the final winning position in all Defender moves, the position is said to have a double-threat solution (or T2 solution).

3) Conservative TSS: Conservative defense, introduced by Wu [20], is a method Defender uses to block double-threat moves. A double-threat TSS with this kind of defense is called a conservative threat space search (CTSS). This approach is based on having Defender place stones on all the cells in normal defense moves. Fig. 2(d) shows the cells on which CTSS places stones, and Fig. 3(b) shows the defensive node of search tree created by conservative defense.

Because the method of playing stones used by conservative defense involves playing stones on all the cells in the defense set, the depth of the search tree turns to a half height from the perspective of the search tree. This occurs because all Defender and Attacker nodes turn into one. Fig. 4 shows the structure of the CTSS tree. Because every Attacker node has only one child,

[3]The $Value$ of a node in an AND/OR tree is as follows: $\text{Value} = \{\text{Win}, \text{Fail}, \text{Unknown}\}$

[4]Using legal stones to defend against threats is called normal defense. Such moves are normal defense moves.

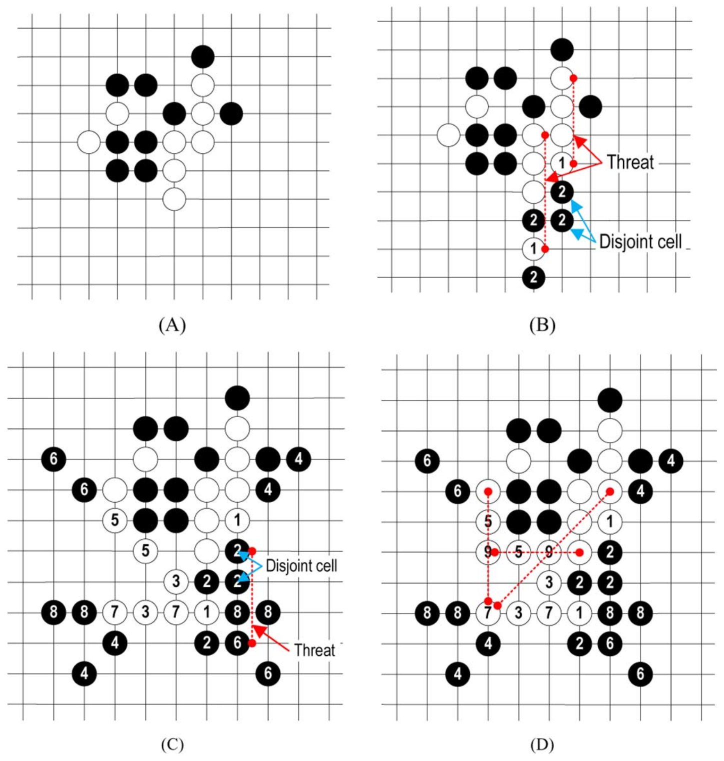

Fig. 6. Defender's threat caused by conservative defense (numbers in the figure represent the cells in each searching level). (A) The initial position of searching. (B) The double-threat move of Attacker and the blocking moves of Defender. (C) The position when black stones are played on the cells numbered 2 in CTSS. (D) The position of searching end in CTSS, the white wins.

Attacker and Defender nodes can be combined in Attacker node, like the nodes in the ellipses of Fig. 4.

Because the depth of the search tree is reduced to half of the height, the search states can decrease significantly, increasing the search speed. Consequently, this paper obtains the following property.

4) Property 2: In the AND/OR tree for the two-player game, Attacker and Defender nodes are combined into an OR-Node if Attacker has just one child. Therefore, the height of the search tree is halved, and the AND/OR tree is transformed into an OR tree.

Using conservative defense to decrease the branching factors and the depth of the search tree accelerates the search speed. However, this approach is excessively defensive and ignores lots of the state space when searching.

This approach can be considered favorable for searching by Defender: in each defense, the number of Defender stones increases faster than the number of Attacker stones. Take the Live 4 Connection for example; Defender can increase by a maximum of four stones at a time, which naturally puts Attacker at a disadvantage, and impedes the finding of a solution.

In double-threat TSS, if Attacker finds the T2 solution based on the conservative defense of Defender, the solution that is found by using CTSS in a position is said to have a CTSS solution. Thus, the following property is obtained.

5) Property 3: When playing the CTSS in a Connect6 position, a CTSS solution that can be obtained under a position is definitely the correct answer; otherwise, no CTSS solution exists for the position.

Anyhow, CTSS has the advantage of rapid search speed and the disadvantage of higher error rate. CTSS thus is a high-risk search method.

C. Search Goal

Considering the above discussion and the characteristics of Connect6, the search goals in this game are summarized below.

1) If the game-theoretic value [9] of the initial position has been determined, that value must be identified. The game-theoretic value of Connect6 is the TSS solution. The $Value$ of the node in the AND/OR tree is as follows: $Value = \{\mathrm{Win}, \mathrm{Fail}, \mathrm{Unknown}\}$.
2) If no TSS solution exists in the initial position, the most promising move is obtained from the first level.

Achieving these objectives requires considering numerous issues. First, this paper discusses the Iterative TSS of Connect6.

III. The Iterative Threat Space Search

This section proposes a new search structure for double-threat TSS, termed as the iterative threat space search (iterative TSS or ITSS).

A. Disjoint Relation

Like the discussion on double-threat TSS, the existence of two threats after Attacker plays a move requires Defender to make different defensive moves. Fig. 5 shows that when white faces two threats, white can respond with the following three

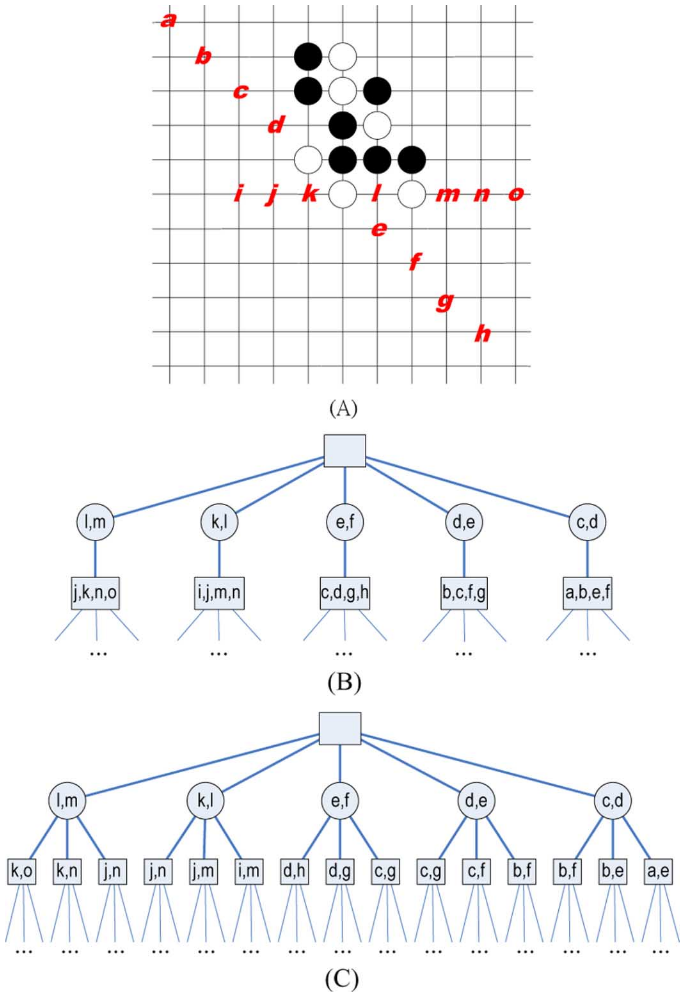

Fig. 7. Example of using CTSS cannot obtain the solution because of the disjoint relation at level 1. (A) is the initial position, (B) is the search tree that uses a conservative defense against Attacker moves in level 1, and (C) is the search tree that uses the normal defense against Attacker moves in level 1.

defensive moves, (1, 3), (2, 3), and (2, 4). Regardless of circumstances, Defender will make one of three defensive moves. Therefore, the three defensive moves can be said to possess a *disjoint relation*.

Definition 1: In double-threat TSS with normal defense, *disjoint relation* indicates the relationship among the normal defensive moves of Defender.

Moves with a disjoint relation must not simultaneously appear in defensive moves. Thus, this problem must be solved when using conservative defense. Two methods exist for solving this problem of disjoint relation. First, this paper discusses the influence of disjoint relation caused by CTSS. The most obvious influence is the problem of Attacker in facing nonexistent threats.

Fig. 6 illustrates CTSS used by white. When white stones are placed on the cells numbered 1, two threats are generated by two Dead 4, and because Defender employs conservative defense, the stone is placed on the cells numbered 2; when searching level 8, Defender occupies the cells numbered 8 in (C). At that time, Attacker faces an unexpected threat, like that marked in the figure.

Definition 2: In CTSS, all the cells that are conservative defenses of Defender that do not exist simultaneously in normal defense are called *disjoint cells*.

When searching level 2, black has two *disjoint cells* because of its conservative defense, like the disjoint cells in (B) and (C) of Fig. 6. Thus, when searching level 9, if white considers blocking this threat, this search will yield no CTSS solution, and will result in mistakes. The search solution for this example is showed in Fig. 6(d).

To solve this problem, disjoint cells are adopted in conservative defense. Fig. 5 shows this conservative defense move (cells 1, 2, 3, 4) when white is about to block the threats of black. Regardless of how Defender blocks the threats, cells, $\{1, 2\}$ and $\{3, 4\}$ cannot exist simultaneously. In this situation, cells $\{1, 2\}$ and $\{3, 4\}$ are considered disjoint cells. When searching leads Defender to confront a threat, we can examine whether disjoint cells exist in the Threat-Connection to determine the existence of the threat.

B. Iterative TSS

Rather than using conservative defense moves, the second method uses normal defense for double-threat moves. This approach can solve the problem of disjoint relation. This occurs because when using normal defense, the disjoint relation naturally does not exist. However, one has to give up the advantage of CTSS quick searching.

Fig. 7 illustrates an example of conservative defense with no solution, and normal defense with solution. Five double-threat moves exist for white in the initial position. Regardless of what move is made, if Defender adopts a conservative defense, Attacker cannot find the solution, as shown by Fig. 7(b). Instead of using conservative defense, this paper uses normal defense, and the example shown yields a solution.

However, this paper predicts that the use of normal defense in searching will rapidly expand the searching space. Additionally, since Attacker move is an AND-Node, to prove that the solution can be found under an AND-Node, it must prove that all of the children of the AND-Node must obtain solutions. As shown in Fig. 7(c), proving that (e, f) can obtain a solution requires proving that moves, (d, h), (d, g), (c, g) can first find solutions.

To solve this problem, this paper uses Iterative TSS to avoid disjoint relations. This method is illustrated using the example in Fig. 7(a).

On the first iteration, ITSS uses conservative defense to deal with the double-threat moves, and no solution is searched for white. On the second iteration, ITSS uses normal defense in response to the double-threat moves of white in the first iteration. This paper only discusses the move, (e, f) in Fig. 7(a), and no solution is obtained in other double-threat moves involved in the first iteration. Fig. 8(a) shows the relative position in Fig. 7(a). White places on the cells are numbered 1 in the figure [e and f in Fig. 7(a)].

ITSS uses normal defense to deal with the double-threat move, and black places on (c, g), (d, g), and (d, h), respectively. Regardless of move, solutions can be obtained using CTSS, like (B), (C), and (D) in Fig. 8.

Thus, a solution exists for white in this position, involving white playing a move on the cells numbered 1. Fig. 9 shows the

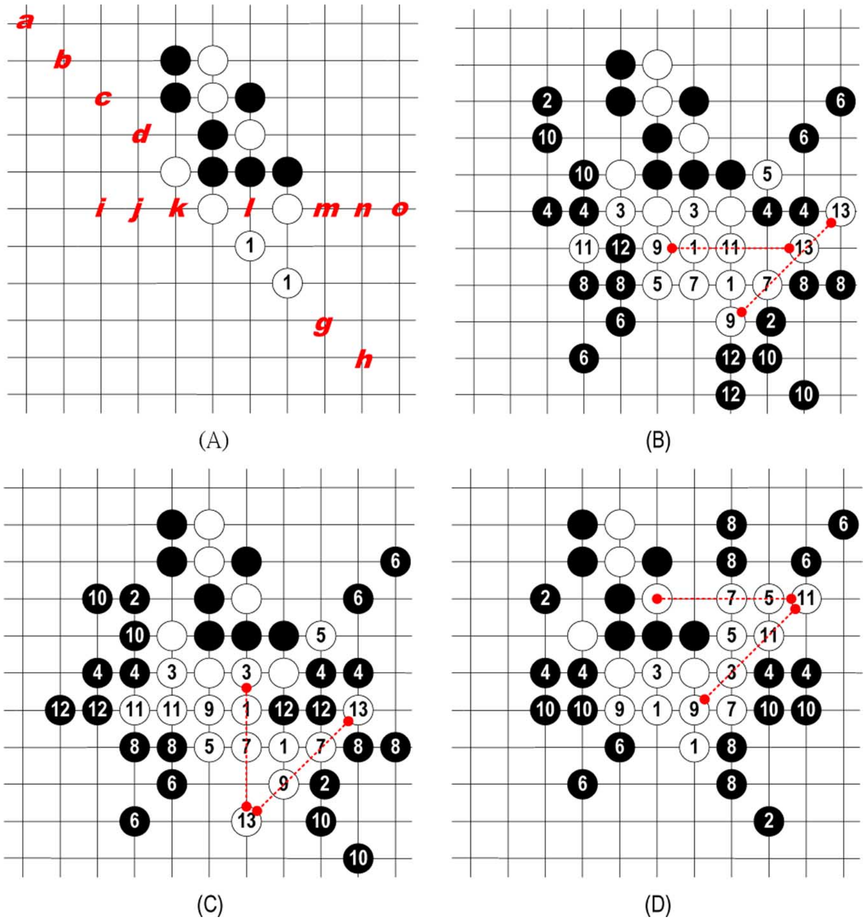

Fig. 8. Example of ITSS finding the double-threat solution.

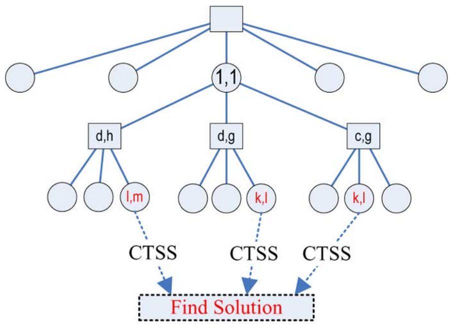

Fig. 9. Structure of the search tree for the ITSS solution. White makes a move on the cells numbered 1 in Fig. 8(a).

structure of the search tree for this solution. If Attacker makes a move on the cells numbered 1 in Fig. 8(a), the correspondent solution is obtained.

C. The Search Architecture of ITSS

In the first iteration of searching, this paper uses CTSS to rapidly search all the double-threat moves. In this iteration, the search ends if any CTSS solution exists; otherwise, the search proceeds in its next iteration.

The second iteration uses normal defense to deal with the double-threat moves generated from the first iteration. After performing the defensive moves, new double-threat moves are made based on the defensive moves. Simultaneously, CTSS is repeated on these double-threat moves. Because Defender uses normal defense in the second iteration, the problem of disjoint relation of moves, which occurs in the conservative defense of Defender, is excluded from the search result of the double-threat moves in the first iteration.

Fig. 10 illustrates the search architecture of ITSS. Node A is the initial position of the board. First, ITSS performs CTSS on the double-threat moves, nodes B, C, and D. If one node can obtain a solution using CTSS, the CTSS-subtree can be preserved and the search process can be completed in that iteration. Otherwise, the branch built by CTSS is deleted. Because no possible Defender moves exist in the conservative defense, the branch built by CTSS is deleted.

This approach can maintain the efficiency of CTSS. As for the position where the double-threat solution can be obtained using CTSS, benefits still exist for quickly obtaining the solution, such as those shown in the first half of Fig. 10.

When the solution for nodes, B, C, and D cannot be obtained in the first iteration, ITSS forms normal defense moves, such as nodes E, F, G, H, I, J, K, and L, based on the double-threat moves of B, C, and D in the second iteration, and continues making double-threat moves of Attacker based on every defensive move. Currently, ITSS can use the same approach as in the first iteration, performing CTSS on these double-threat moves, like nodes, M, N, O, P, Q, and R in Fig. 10.

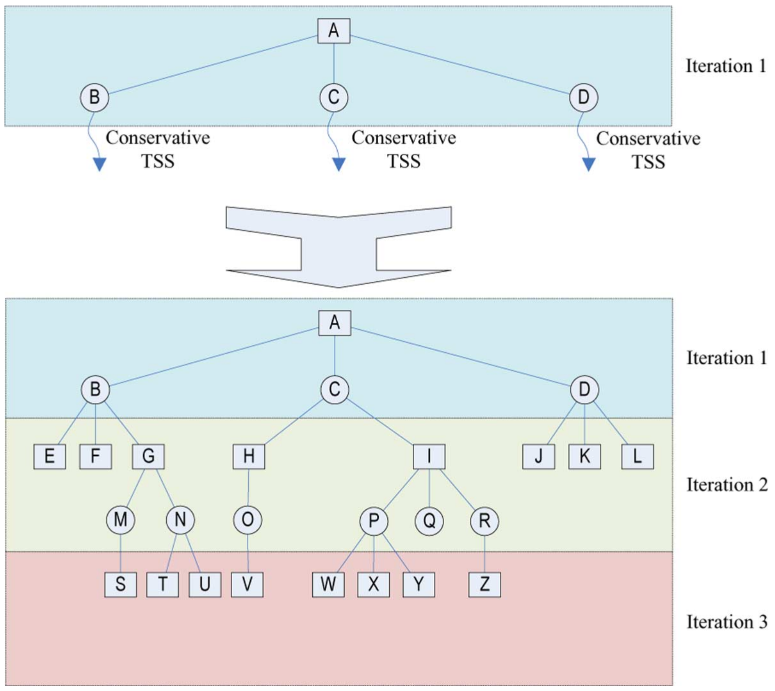

Fig. 10. Development of the search tree in ITSS.

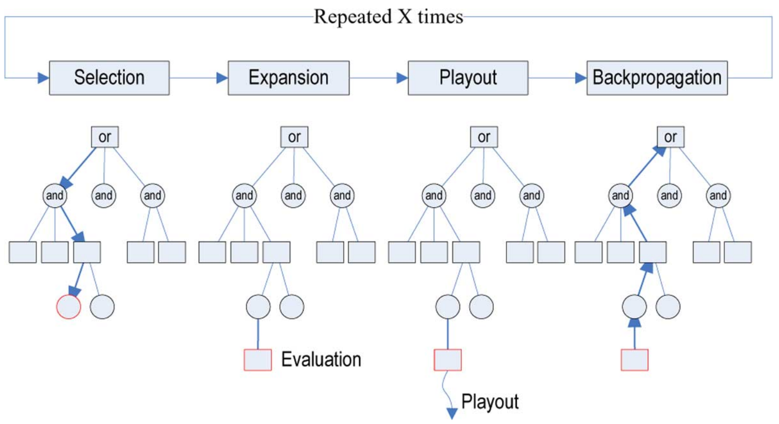

Fig. 11. Outline of the MCTS algorithm edited from [4].

When conducting searches in the second iteration, since the double-threat moves use normal defense in the second iteration, no problem of disjoint relation exists for those double-threat moves in level 1, and the problem of being unable to correctly identify the solution attributed to the disjoint relation of conservative in the first iteration can naturally be solved. Thus, if no CTSS solution exists, development of the next iteration can continue until either a solution is found or the absence of any solution is confirmed.

D. Conclusion for ITSS

TSS is a common searching method for Connect-k games. Meanwhile, CTSS is a much more efficient searching method for Connect6. However, the problem of disjoint relation occurring in CTSS is an important issue. Therefore, how to take advantage of the quick searching of CTSS, how to efficiently solve the problem of disjoint relation, and how to rapidly and accurately perform TSS, this paper proposes the ITSS method. The ITSS method can efficiently perform double-threat TSS for Connect6.

ITSS is a search architecture for double-threat TSS, and can be combined with any search method to develop the search tree. Hence, Section IV explains both how to perform ITSS, and how to combine it with MCTS. The Section IV proposes a MCTS search structure for Connect6, and applies it to perform the search for Connect6.

IV. MCTS Implementation in Kavalan

This section illustrates how to combine the MCTS with the ITSS method mentioned in the previous section. Furthermore, this section demonstrates how to design the two-stage MCTS algorithm for Connect6.

A. The Basic Algorithm of MCTS

First, this paper discusses the basic search architecture of MCTS. MCTS [4], [5], [11] are divided into four steps: *selec-*

TABLE I
THE TYPES OF TWO-STAGE MCTS

Types	First stage	Second stage
Type I	Double-threat moves	Single-threat moves Non-threat moves
Type II	Double-threat moves Single-threat moves	Non-threat moves

tion, *expansion*, *play random game* (or *playout*), and *back-propagation*. The four steps are repeated according to the settled times until the search time is used up. Fig. 11 shows the outline of MCTS, and illustrates the four simulation steps.

MCTS is a best-first search [5], [19], which uses playout to predict candidate moves, and simultaneously builds the search tree, from top to bottom, to improve the precision of the prediction in the upper position. Hence, in the simulation of every round, MCTS uses playout to predict the leaf node, and corrects the ancestor PV[5] value from the leaf node based on the result of the playout to improve the search quality.

1) two-Stage Search: Basic MCTS does not search in stages, but rather takes the candidate moves as those with equal importance, and distinguishes the significance of different moves based on the playout results. However, since Connect6 is a sudden-death game, one side tends to immediately lose if it neglects the threats possessed by the opponent. Therefore, this paper divides the development of candidate moves into two stages. The core idea of stages is that the focus is on solving the sudden-death problem during the first stage, and on searching for the most promising move during the second stage. This form of MCTS is labeled two-Stage MCTS.

This paper uses an AND/OR tree to develop the search tree to fit the situation of Connect6. As described in "Section II-C," for determining the game-theoretic value of the initial position, the AND/OR tree is an appropriate means of doing this.

2) Types of Two-Stage MCTS in Connect6: Connect6 involves three kinds of moves, including double-threat, single-threat, and nonthreat moves. The use of two-stage MCTS to develop candidate moves thus can be divided into two types, as listed in Table I. The difference between the two types involves the stage in which single-threat moves are generated.

Selection of two-stage MCTS is a strategic decision. In the initial position, in the event of double-threat or single-threat solutions, using different types will exert different effects on searching efficiency. This paper employs an experimental method to compare the two types.

If, in initial position, neither double-threat or single-threat solutions exist, using two-stage search and focusing on TSS may negatively impact the search for other candidate moves. Thus, when playing games, it is necessary to make a suitable distribution according to usable resources.

B. The Four Strategic Steps of two-stage MCTS

As for the four steps of MCTS, this paper details the strategy used in two-stage MCTS step-by-step form below.

1) Selection Strategy: Regarding selection strategy, this paper uses *UCT Selection* because this approach is easy to use and has performed well in numerous studies [5], [6], [11], [13]. This approach is developed from the multiarmed bandit problem [10], and it considers the balance between exploitation and exploration. This approach not only selects the best node based on the information obtained, but also selects less visited nodes. In Formula 1, this approach chooses nodes from its children, as follows:

$$k \in \arg\max_{i\in S}\left(V_i + C \times \sqrt{\frac{\ln n_p}{n_i}}\right),\ \text{where } V_i = \frac{PV_i}{n_i}. \quad (1)$$

In two-stage MCTS, each node p represents a given position of a game. Let k be the chosen node from the current node p, i be the child node of p, and S be the set of nodes that are the children of node p.

A node contains two pieces of information: PV_i and n_i. PV_i represents a value which is the result of playout and evaluation for node i and will be discussed in Subsection 4. n_i denotes the number of visits for node i, and its value will be increased by 1 if it is a node in the simulation path. V_i is the PV rate, and represents a value like the win rate in MCTS, but its value may be negative (opponent wins), and is calculated as described in Formula 1.

Furthermore, C is the exploration factor, and controls the balance between exploitation and exploration. Two-stage MCTS uses different selection strategies for different types of candidate moves.

2) Type I of Two-Stage MCTS: During the first stage, to consider the possible solution of double-threat moves, Type I of two-stage MCTS equally develops double-threat moves. This design aims to find whether T2 solution exists in a position, and the experimental results indicate that BFS is more efficient than DFS. This paper thus modified Formula 1. When two-stage MCTS develops the T2-subtree of double-threat TSS, the development of the T2-subtree resembles that of BFS. In T2-subtree, this paper selects a node from its children using Formula 2

$$k \in \arg\max_{i\in S}\left(V_i + \frac{n_p}{n_i}\right). \quad (2)$$

During the second stage, to offer more choices to the node which is not T2 Fail[6], this paper adds a value in the computation of UCT selection, which is called Heuristic value (Hv). The aim of the H_V is to direct the search according to the heuristic knowledge, and it increases the number of search times for the T2-subtree. This approach is *UCT selection enhancement*, and is a widely applied concept [5], [11].

Taking Fig. 12 as an example, when node A is T2 Fail, it develops other candidate moves, such as nodes E and F. Evaluating node E involves double-threat moves for the defensive side. Therefore, this node becomes the root of T2-subtree for the defensive side, and thus is assigned an Hv value.

The Hv value is a heuristic value that increases the times of selecting the T2-subtree. If a position has finished the first stage search, and the T2 solution cannot be identified, the candidate moves developed after the first stage must search the T2 solution

[5]PV is the value of back-propagation, which is the result of playout and evaluation for the leaf node in every simulation.

[6]T2 Fail means the solution of double-threat TSS does not exist, and TSS Fail means the solution of double-threat and single-threat TSS does not exist.

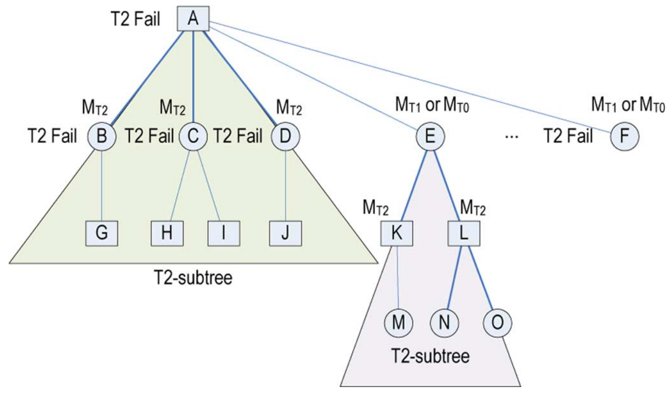

Fig. 12. Search tree and its T2-subtree in two-stage MCTS. M_{T2} represents double-threat move, and M_{T1} represents single-threat move.

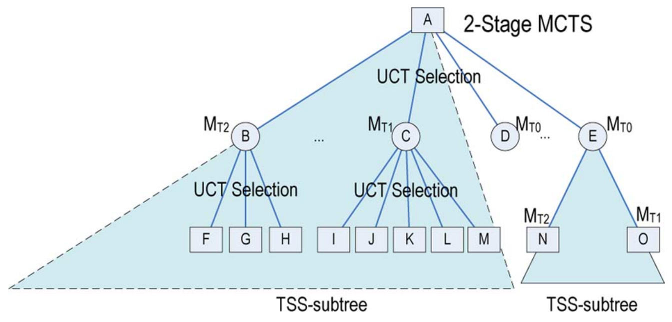

Fig. 13. Search tree and its TSS-subtree of two-stage MCTS.

of the defensive side. The method of node selection from the children of that position resembles that laid out in Formula 3

$$k \in \arg\max_{i \in S} \left(V_i + C \times \sqrt{\frac{\ln n_p}{n_i}} + H_v \right). \tag{3}$$

To search T2-subtree and the balance between other nodes without T2-subtree, the Hv value of the node is reduced by 1 after each simulation. The main advantage of this method is that it gradually reduces the effect of Hv value.

3) Type II of Two-Stage MCTS: The difference between Types II and I is that the single-threat moves of Type II are generated during the first stage. This kind of search mode forms double-threat and single-threat moves during first stage. Fig. 13 shows the search tree of the Type II of two-stage MCTS.

Type II of two-stage MCTS cannot immediately show the difference between double-threat and single-threat moves at the beginning of the first stage. Thus, this paper uses the Hv value method to give an Hv value for double-threat moves after position evaluation. This approach can distinguish moves with different threats via UCT Selection.

As for Type I, this Hv value must be gradually decreased during back-propagation.

4) Expansion Strategy: The concept of the two-stage MCTS is used to control the timing of the generation of different candidate moves. In any position, candidate moves generated by one side may have various types, and these moves are generated in stages according to game features.

5) Node Evaluation: Node evaluation determines the position of a node and generates candidate moves accordingly. In two-stage MCTS, in position evaluation, this paper uses CTSS to seek double-threat solutions. CTSS is used mainly because it is highly efficient. While CTSS will waste some calculating time to search, the associated benefits outweigh the calculating time. The main reason for the benefits is shown in Property 2, and can be demonstrated from the results of the experiment in Section VI.

Moreover, because the search result of CTSS is a determined value, two-stage MCTS does not need to waste any simulation resources on these determined positions. Using CTSS for node evaluation can rapidly reduce the state space search and the number of simulations required.

6) Generating First-Stage Candidate Moves: In the proposed Connect6 search method, the first stage focuses the search on Threat-Moves. Therefore, after the node evolution, candidate moves depend on the strategy developed in earlier stages. This paper divides two-stage MCTS into two different types based on the generation of single-threat moves. In first stage, Type I of two-stage MCTS generates only double-threat moves, but Type II generates double-threat and single-threat moves.

In the proposed search architecture, CTSS is always used for double-threat moves in a two-stage MCTS. Following this development strategy, the development of double-threat moves is based on ITSS. Restated, two-stage MCTS uses ITSS to search for double-threat moves.

Some positions having numerous Threat-Moves may become a heavy resource burden when searching for all possible Threat-Moves. Therefore, the search should be based on information gained to determine whether it should continue to focus the search on Threat-Moves when resources are limited. This paper proposes a method for determining whether it should continue to search for the Threat-Moves under a position based on its simulation times and PV rate.

This strategic approach is needed though the search results adversely affect accuracy if two-stage MCTS gives up searching for Threat-Moves as the PV rate is low. However, this cannot help being done like that under the limited resources. The purpose of search is finding promising moves in level 1 of the search tree, so this strategy is only used in the root node. If the first-stage PV rate of the root node is low, two-stage MCTS should give up searching for Threat-Moves and change the search strategy to defense. The proposed strategy for selecting simulation times and PV rate is described in Section VI-B.

7) Generating Second-Stage Candidate Moves: When making candidate moves during the second stage, a gradual generation method is used. When evaluating positions, a heuristic function calculates and ranks the scores for all empty cells in a position. Therefore, two-stage MCTS generates candidate moves based on the scores for empty cells. These empty cells ordered by scores are called ordered-by-scores cells (or *probing cells*). Therefore, the probing cells is the probing order for all empty cells in a position, and it is also established in node evaluation. The two-stage MCTS forms candidate moves based on the probing cells during the second stage.

If two-stage MCTS generates candidate moves based on all empty cells during the second stage, it results in an excessively large number of candidate moves. Therefore, a gradual generation method is used. Whenever two-stage MCTS generates second-stage candidate moves, it uses a fixed number of probing cells to generate candidate moves at a time.

The number of probing cells selected is a strategic consideration. If the number is too large, it will result in excessive candidate moves, which may decrease search depth. However, too few probing cells may ignore certain important cells in the beginning.

Besides, to prevent excessively rapid search tree development, and avoid wasting excessive time on search tree construction, two-stage MCTS establishes a leaf node in every simulation.

8) Play Random Game (Playout) Strategy: MCTS uses *playout* to forecast the possible condition of leaf node. This is a strategic consideration. When selecting moves in playout, if one simply randomly selects moves to place stones on empty cells, the prediction is inadequate. Therefore, the *playout* method of selecting moves should be identical to the expansion strategy of the search tree in the simulation.

The strategy used in this step is making Threat-Moves the primary consideration, since if Threat-Moves exist in a position, the likelihood of obtaining the TSS solution is higher. Therefore, Threat-Moves are the first choice.

This paper limits the depth of playout based on the feature of Connect6. This strategy supposes that the probability of a draw increases with reducing the number of cells. Furthermore, if a CTSS solution can be obtained when evaluating a position, this position has no need to perform playout in prediction because the $Value$ of this position is determined.

9) Back-Propagation Strategy: Back-propagation describes the mechanism whereby after every simulation, the result of the evaluation or playout of the leaf node is propagated back to the ancestor of the relevant node. This mechanism affects the selection in the next simulation. Because this paper not only uses playout to predict the leaf node, but also uses CTSS to search for double-threat moves, it includes the reporting-back of both situations.

The first situation involves reporting-back when CTSS finds the T2 solution. This form of reporting-back can correct the ancestor *Value* above a node, and correct it according to every node type (AND-Node or OR-Node).

Besides, when one position is determined, that position is assigned a larger value of playout PV. According to [29], the number this paper used is that if offensive side wins, PV gain 10; if defensive side wins, PV lose 5.

The second situation is reporting-back after playout. The way we use that is if offensive side wins, PV gain 1; if it is a draw, PV earn 0; if defensive side wins, PV lose 1.

C. The Search Architecture of two-stage MCTS

This section describes the search architecture of two-stage MCTS. Two-stage MCTS generates candidate moves in two stages. The division of candidate moves is a strategic decision, and is made based on the features of the game. Regardless of stage strategy adopted, the difference of two-stage MCTS lies in the mechanism that generates the candidate moves and the stage-transition condition.

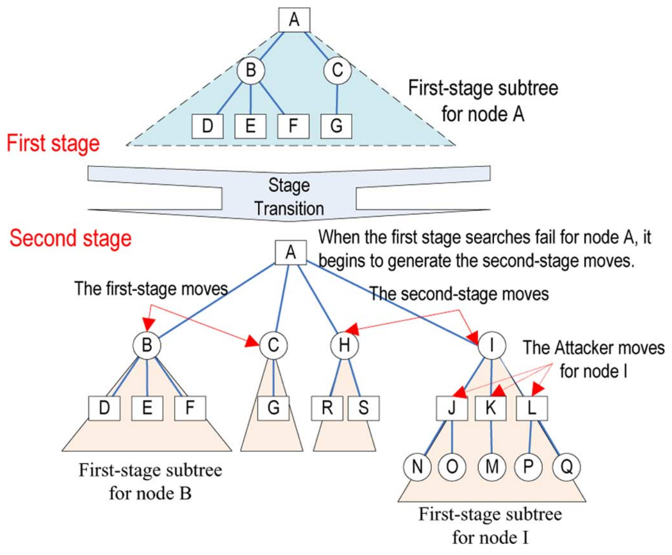
A
B
C
D
E
F
G
First-stage subtree for node A
First stage
Stage Transition
Second stage
A
When the first stage searches fail for node A, it begins to generate the second-stage moves.
The first-stage moves
The second-stage moves
B
C
H
I
The Attacker moves for node I
D
E
F
G
R
S
J
K
L
N
O
M
P
Q
First-stage subtree for node B
First-stage subtree for node I

Fig. 14. Search architecture of two-stage MCTS.

1) The Development of Two-Stage Search: Fig. 14 illustrates the development of the search tree for two-stage search. When seeking a goal from an initial position, failure to identify a goal means this position has no such goal. Simultaneously, changing our target to the successors of the initial position examines whether this goal exists for the successors, and developing other candidate moves. When new candidate moves are generated, it is necessary to examine whether this goal exists under these new generated nodes.

Fig. 14 shows that when the goal of node A search fails, it is necessary to examine whether this goal exists under nodes B and C. Furthermore, when creating nodes H and I, it is necessary to examine whether this goal exists under nodes H and I. Taking the two-stage MCTS of Connect6 as an example, the goal of Type I is a T2 solution, while that of Type II is a TSS solution. The development of two-stage search proceeds in this way.

Following this developing mode, if the goal of the offensive side cannot be identified, the development of the candidate moves of the offensive side will consider whether the goal exists in the defensive side in two-stage MCTS. This method can be accelerated to understand whether the offensive side will encounter resistance from the defensive side in the candidate moves of offensive side for fear that the defensive side wins.

2) Stage Transition: The above illustration demonstrates that the search tree contains the first-stage subtree (T2-subtree for Type I; TSS-subtree for Type II, see Figs. 12 and 13). The second-stage search tree, shown in Fig. 14, represents the inclusion relation between the first-stage subtree and the search tree.

Type I of two-stage MCTS takes the double-threat solution as its search target during the first stage. Thus, in the T2-subtree, if the root of T2-subtree is T2 Fail, the stage transition is initiated. Meanwhile, Type II takes the double-threat and single-threat solutions as its search target during the first stage. Thus, when the root of the TSS-subtree is TSS Fail, the stage transition is initiated.

```
MonteCarloTreeSearch() {
  while (MCTSRoot.Value == Unknown && ResourceAvailable()) {
    Initialization();              // Initialize simulation state
    PlaySimulation(MCTSRoot);
    nsimulations++;
  }
}
```

Fig. 15. Two-stage MCTS algorithm.

Attacker of first-stage subtree can be the offensive or defensive side of the search tree. Therefore, assessing search failure in the first stage depends on the subtree not the search tree.

D. The Proposed Algorithm

This subsection describes the two-stage MCTS algorithm. It includes two main parts: two-stage MCTS algorithm and play-simulation algorithm.

1) Two-Stage MCTS Algorithm: Fig. 15 shows the MCTS algorithm, which comprises three parts: the initial simulation state, the simulation process, and the accumulation of simulation times. The algorithm ends when it is proved that the offensive side wins or loses; alternatively the search ends on reaching the user set terminative condition.

Because this paper divides the search into two stages, the MCTS algorithm implements the second stage after the end of the first one. The $Value$ of a node in an AND/OR tree is as follows: $Value = \{\mathrm{Win}, \mathrm{Fail}, \mathrm{Unknown}\}$.

Consequently, three situations can exist when the algorithm ends. First, the $Value$ of root node is Win. This means that the offensive side finds its solution. Second, the $Value$ of root node is $Fail$. This means that the defensive side finds its solution, and offensive side cannot defend this solution. In other cases, the $Value$ of root node is $Unknown$. This means that the game-theoretic value of the initial position is unknown.

2) Play-Simulation Algorithm: The four steps involved in MCTS are performed in the *Play-Simulation* function, as detailed in Fig. 16.

The *Play-Simulation* algorithm comprises three parts: evaluation, first-stage search, and second-stage search. The algorithm only allows each node to be evaluated once. During evaluation, generated threats are checked by moves based on the positions. Appropriate defense is implemented in response to each threat, and candidate moves are formed.

Besides, in the evaluating state, this paper adds the CTSS check for double-threat moves. This method can accelerate the examination of the game-theoretic value of a node (position). If a solution of CTSS exists under this node, the $Value$ of ancestor must be updated from this node; otherwise, playout must be predicted for this node.

V. Heuristic Knowledge—Relevance-Zone Search

To demonstrate how an OR-Node becomes a $Fail$ position or an AND-Node becomes a Win position in Connect6, the concept of *relevance zones* [17], [20], [23], [26] is used to develop a search tree in two-stage MCTS. The search method mentioned

```
PlaySimulation(TNode myNode) {
 HandleFirstStageSubtree();  // Judge the root of first-stage subtree

 if (myNode.Visits == 0) {
   Evaluate(myNode);
   HandleFirstStageFail();  // Handle the search Fail of first stage
   if (myNode.Value == Unknown)
     BackWins=Playout(myNode);
   else {
     EvaluateResult = true;
     If(myNode.Value == Win ) BackWins=10 else BackWins=-5;
   }
 }
 else {
   if (Not myNode.FirstStageFail){
     // ------ First stage – Threat-Moves Search ------
     if(ThreatMove.Count != 0)
       Next = Create Threat-Moves
     else
       Next = Select Threat-Moves;  // Selection strategy
   }
   else {
     // ------ Second stage – promising move search ------
     if(CandidateMoves.Count != 0)
       Next = Create Candidate Moves;
      else
        Next = Select Candidate Move;  // Selection strategy
    }
   PlaySimulation(Next);  // Recursive selection
   if(EvaluateResult) { UpdateNodeValue(myNode); }
 }
 // ------ Backpropagation strategy ------
 myNode.Visits++;
 if(myNode == offensive side)
   myNode.PV  += BackWins;
 else
   myNode.PV -= BackWins;
}
```

Fig. 16. Play-Simulation algorithm.

in the section developed independently of [26], is similar to that in Section III in [26].

A. Relevance Zone

In Connect6, threats are the key to prove winning positions. Wu and Lin [26] proposed relevance-zone-oriented proof (RZOP) search, a new threat-based proof search method. The RZOP search is a powerful method of proving winning positions. For RZOP search, Wu and Lin presented a novel general method for efficiently constructing and promoting relevance zones in different orders. Relevance zone can substantially reduce the search space; therefore, this section introduces relevance-zone search in two-stage MCTS.

Fig. 17 is an example of relevance-zone search. When white plays a move on (A, B) in Fig. 17(a), black has the CTSS solution shown in Fig. 17(b). When one side has the CTSS solution, it is labeled Attacker in relevance-zone search while the other side is labeled Defender. If Attacker finds a solution via CTSS, Defender must place stones to prevent this CTSS solution from replaying. Otherwise, Defender loses the game.

A method of constructing the relevance zone for Connect6 is described in [26]. The Wu and Lin theory offered a sounder theoretical basis for constructing the relevance zone for a position in Connect6. The key point of constructing the relevance zone is

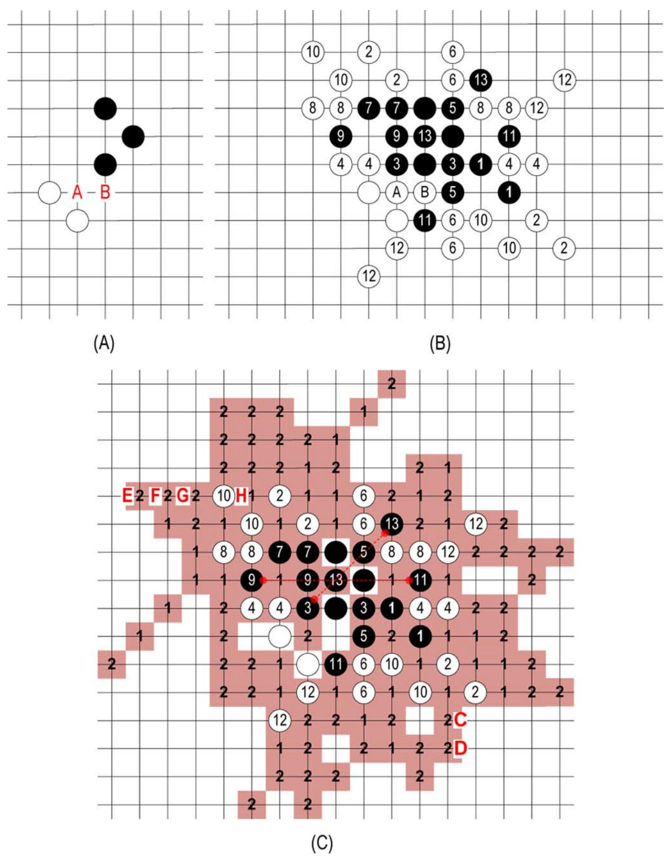

Fig. 17. Item (A) is a Connect6 Joseki obtained as described in [16], (B) is the black CTSS solution, and (C) is the relevance zone based on the CTSS solution for black. (A) The initial position. (B) A CTSS solution. The marked numbers represent the order of stones placement for black and white. (C) The relevance zone of the black CTSS solution comprises the area of gray cells.

forming a counter-threat segment or an inversion that prevents Attacker from replaying.

Fig. 17(c) shows that the relevance zone $\Psi = \langle Z_1, Z_2 \rangle$.[7] Z_1 and Z_2 are the set of empty cells in the initial position. The Z_1 is those cells in which one stone can form a counter threat segment or an inversion, and Z_2 is those cells in which two stones can form a counter threat segment or an inversion. Note that since Z_1 and Z_2 are incremental, $Z_1 \subseteq Z_2$.

Because relevance-zone search is embedded into two-stage MCTS, the relevance zone in Fig. 17(c) is based on the initial position and the CTSS solution for black. In Fig. 17(c), construction of the relevance zone does not consider whether white plays the move on (A, B). Therefore, if white makes a null move, black can surely get this CTSS solution and construct this relevance zone based on the CTSS solution.

In Fig. 17(c), the relevance zone derived in the CTSS solution can be used for Defender. According to the initial position, if Defender does not place any stone in the relevance zone $\Psi = \langle Z_1, Z_2 \rangle$, Attacker can win based on this CTSS solution.

1) Property 4: Assume that Defender constructs the relevance zone $\Psi = \langle Z_1, Z_2 \rangle$ based on both the CTSS solution and the initial position of Defender. If Attacker finds a CTSS solution, the cells in the relevance zone Z_1 can clearly prevent the CTSS solution from replaying.

Property 4 shows that if a CTSS solution exists on one side in the situation where Defender has two stones to defend the CTSS solution, the opposing side has two possible defenses:

1. place one stone in the relevance zone Z_1;
2. make a Defender threat-move before Attacker establishes the final winning position.

Making a Defender threat-move may not be done inside the relevance zone Z_1. Although cells C and D in Fig. 17(c) are not included in Z_1, they can form a counter threat segment. Relevance-zone search is performed if Attacker presents two threats, presents one threat, or presents no threat.

B. The Design of Relevance-Zone Search

The design of the relevance-zone search in this paper is based on a CTSS solution. When one side has a CTSS solution, the other side runs a relevance-zone search to generate candidate moves. Relevance-zone search is performed depending on the number of threats that Attacker has; therefore, the initial position of relevance-zone search is determined in one of three situations. Explanations for these three situations are as follows.

1) Case 1: Attacker has Two Threats: Attacker presents two threats, so Defender must place two stones to block the threats. Thus, possible defensive moves are checked to determine if Defender can defend against the CTSS solution of Attacker. Case 1 of relevance-zone search is as follows.

Steps 1: Find the relevance zone $\Psi = \langle Z_1, Z_2 \rangle$ based on the new CTSS solution.

Steps 2: Judge whether the other defensive moves can defend against the CTSS solution based on the relevance zone Ψ generated in Steps 1.

The judgment of Steps 2 can be made based on the Property 4. In Fig. 18(a), white already makes two threats, so Defender (black) considers the cells where it can place stones, namely A, B, C, and D. The normal defense moves are (A, C), (A, D), (B, C), and (B, D). Defender considers the situation where black makes the move (A, C), and white obtains the CTSS solution, as shown in Fig. 18(b). From relevance zone Ψ, cell B does not occupy the relevance zone Z_1, and does not form threats with black stones. Defender thus judges that defensive move (B, C) cannot defend against the CTSS solution. By using this approach, black can quickly determine whether to defend from this position. Consequently, the only problem is searching moves (A, D) and (B, D) to determine whether white has a CTSS solution.

2) Case 2: Attacker Has Only One Threat: Because one threat already exists in this case of relevance-zone search, one stone is to block one threat. Only one stone can currently defend against the CTSS solution. Therefore, this case of relevance-zone search is based on the blocking cells to check whether the cell can defend the CTSS solution respectively.

When constructing the relevance zone, the position must include the blocking cell in this case, and it is sufficient to construct relevance zone Z_1. For Fig. 19(a), black plays a move on (A, B), and white finds a CTSS solution in Fig. 19(b); therefore, black has to determine whether blocking cell A is indefensible.

[7]In this paper, the relevance zone is described using the notation in [26].

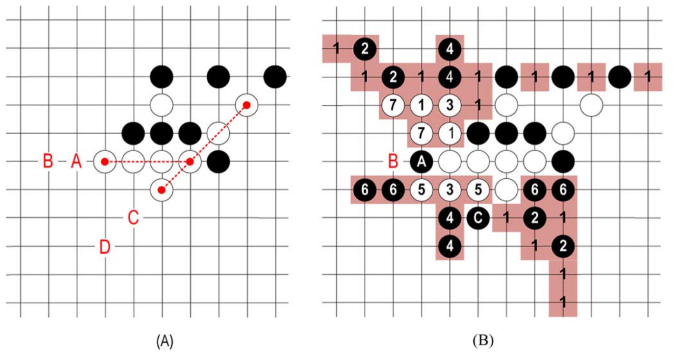
Fig. 18. Example of relevance-zone search where Attacker has two threats, and relevance zone Z_2 is omitted in (B). (A) Initial position. (B) White finds the CTSS solution, and the gray cells represent relevance zone Z1 based on (A).

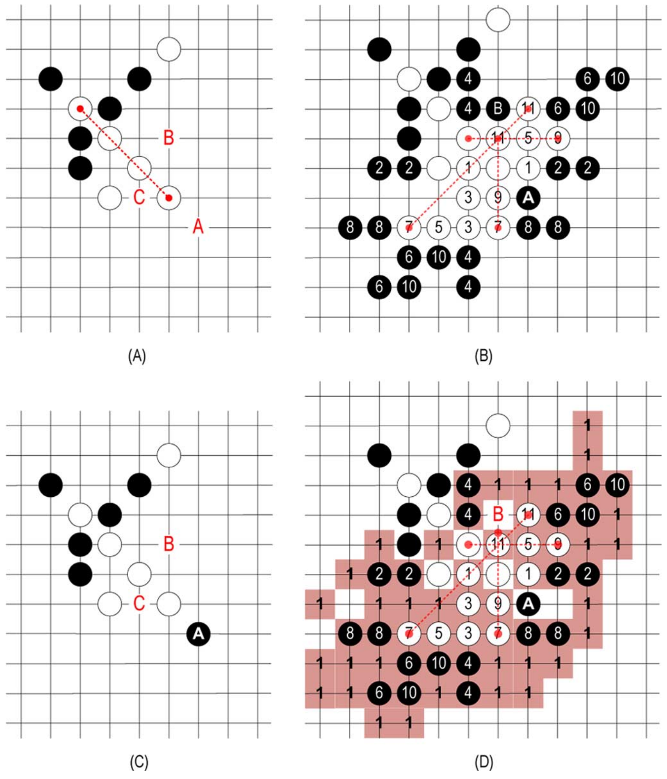
Fig. 19. Example of relevance-zone search where Attacker has one threat. (A) The initial position, (B) a CTSS solution based on black plays a move on cells (A, B) from the initial position, (C) the initial position of black places one stone on cell A, and (D) the relevance zone Z1 constructed based on (C) and the CTSS solution of (B).

If Defender is indefensible in a position, Attacker can find a solution regardless of how Defender moves.

To prove that blocking cell A is indefensible, the position in which black places one stone on cell A must be the initial position. Fig. 19(d) shows the relevance zone Z_1 constructed based on this initial position. That is to say, black has one stone fixed on cell A, and relevance-zone search must verify whether the cell A is indefensible.

In Fig. 19(d), the relevance zone Z_1 excludes the cell B because Fig. 19(b) has proved that cell B cannot defend this CTSS solution. Therefore, the only way to defend this CTSS solution is to play another stone in the relevance zone Z_1 of Fig. 19(d).

Fig. 20(a) shows another CTSS solution for white when black plays one stone on cell A and another stone on cell C, which is in the relevance zone Z_1 of Fig. 19(d). In Fig. 20(a), the area of gray cells indicates the relevance zone Z_1' constructed based on the position of Fig. 19(c) and the new CTSS solution. Fig. 20(b) is the subset computed for the intersection of relevance zone Z_1 (see Fig. 19(d)) and Z_1' (see Fig. 20(a)). Clearly, the cells in this subset can prevent both CTSS solutions from replaying, and the cells outside this subset must have at least one of the two CTSS solutions to win by replaying.

Therefore, if no stone is placed in the subset of Fig. 20(b), the move is indefensible for the two CTSS solutions. In this case of relevance-zone search, the judgment of an indefensible move has to take care of the relaxed critical defense. The move for blocking single threat in relaxed critical defense is unsuitable for this judgment. Fig. 21 shows an example of relaxed critical defense. White has one threat in this Connection Pattern, and black can block it by placing a single stone on cell A, but black can also block it by placing two stones on cells B and C.

Fig. 20 illustrates the situation where Defender has one stone to defend a CTSS solution. The basis for judging relevance-zone search failure is the number of cells in the subset generated by calculating the intersection for a series of relevance zones Z_1. If the number of cells in the subset equals zero, relevance-zone searches fail if Defender has one stone to defend a CTSS solution.

Thus, when the subset contains at least one cell, candidate moves can be generated based on those cells. Case 2 of the relevance-zone search is as follows.

Steps 1: Find the relevance zone $\Psi = \langle Z_1 \rangle$ based on the new CTSS solution given above.

Steps 2: Calculate the intersection based on the new relevance zone Z_1 and other previous relevance zone Z_1', and judge whether the number of cells in the subset is equal to zero. If the number of cells in the subset is not equal to zero, go to Steps 3. If it equals zero, relevance-zone search fails under the blocking cell. If all relevance-zone searches fail under all blocking cells, Defender fails.

Steps 3: Define nodes as indefensible if they have already been produced, but cannot be defended.

Steps 4: Form possible defensive moves based on the cells blocking one threat and the cells in the subset formed by calculating the intersection.

3) Case 3: Attacker has no Threat: Because Attacker presents no threat, the two Defender stones can be used to defend a CTSS solution. This case is more complex because the candidate moves to defend a CTSS solution are more than the Case 2.

Fig. 17 is an example of Case 3. If Attacker finds a CTSS solution, it can be sure that Defender must place one of the two stones on $\Psi = \langle Z_1, Z_2 \rangle$ of Fig. 17(c); otherwise, Defender loses the game [26]. In this paper, the relevance-zone search in Case 3 has a heuristic design, which can be divided into three parts.

4) Case 3–1: Attacker Finds the First CTSS Solution: If Attacker finds the first CTSS solution, it constructs the relevance zone $\Psi = \langle Z_1, Z_2 \rangle$ and checks whether the nodes already produced are indefensible. A move is indefensible in Case 3-1 if:

- neither of its two stones is in Z_2;

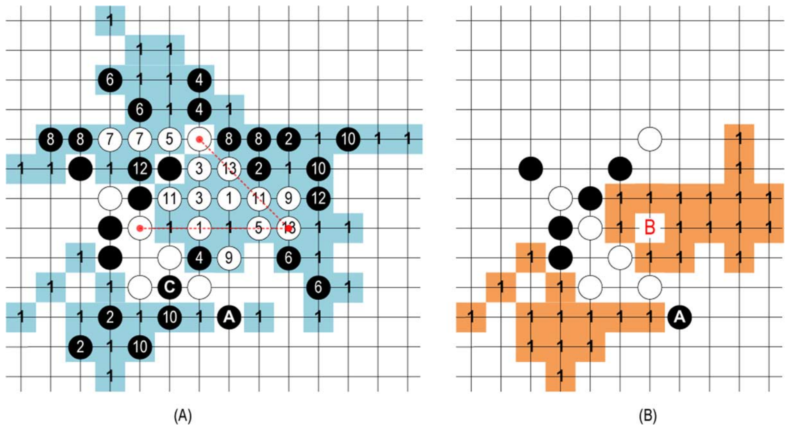

Fig. 20. Example of intersection computed for relevance zones. (A) The Z_1' constructed based on the Fig. 19(c). (B) The subset of relevance zone, which is the intersection of Figs. 19(d) and 20(a).

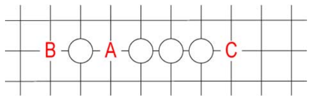

Fig. 21. Example of the relaxed critical defense for single threat.

- one stone is in Z_2, but not in Z_1, and the other is outside Z_2;
- the move cannot form the threat before Attacker establishes the winning position.

According to Property 4, the relevance zone Z_1 is a critical relevance zone. If Defender places one stone in Z_1, it can prevent the CTSS solution from replaying. The proposed strategy in this case is to defend the relevance zone Z_1 in the first place. The steps for Case 3–1 of the relevance-zone search are as follows.

Steps 1: Find the relevance zone $\Psi = \langle Z_1, Z_2 \rangle$ based on the new CTSS solution.

Steps 2: Define nodes as indefensible if they are already produced and cannot be defended.

Steps 3: Generate candidate moves based on the relevance zone Z_1.

The aim of generating candidate moves in this part is to verify whether cells in relevance zone Z_1 are indefensible. Therefore, candidate moves are generated for every two cells in relevance zone Z_1. In Fig. 22, relevance zone Z_1 has 73 cells, so it can generate 37 candidate moves. If the number of cells in Z_1 is odd, it randomly selects a cell in Z_2 but not in Z_1 to form a move. In the next part, it must prove that every cell in Z_1 is indefensible.

5) Case 3-2: Verify Whether the Cells in Relevance Zone Z_1 Can Defend Black CTSS Solution: When Attacker finds the other CTSS solution, it starts to check whether the two cells of the move are indefensible. The procedure is the same as in Case 2 of the relevance-zone search because the two cells can be considered the blocking cells. In Fig. 23(a), white plays a move on cells A and B, which are in the relevance zone Z_1 of Fig. 22, and black finds a new CTSS solution. In this situation, relevance-zone search starts to verify whether cells A and B are indefensible. Therefore, the relevance zones are constructed based on white playing a stone on cell A and B as in (B) and (C), respectively, in Fig. 23. Case 3–2 of the relevance-zone search is same as Case 2.

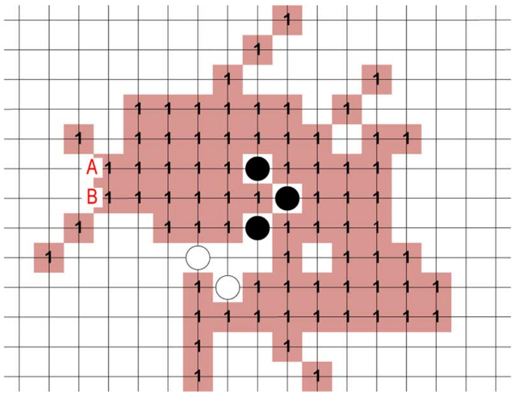

Fig. 22. Relevance zone Z_1 obtained as in Fig. 17(c).

6) Case 3-3: Verify the Cells That are in Z_2 but not in Z_1: When all cells in the critical relevance zone have proven to fail in defending Attacker CTSS solution in part two, relevance-zone search starts to verify whether cells that are in Z_2 but not in Z_1 are indefensible. Candidate moves generated in this part are based on cells that can form threat-move.

In Fig. 17(c), cells E, F, and G are qualified, but cell H is not qualified because cell H was proven indefensible in part two. The defensive moves formed by these three qualified cells are (E, F), (E, G), and (F, G), respectively. If all the defensive moves formed by cells that are in Z_2 but not in Z_1 fail, Case 3 of the relevance-zone search fails.

C. Conclusion in Relevance-Zone Search

Relevance-zone search can be used not only when the offensive side defends a CTSS solution of the defensive side, but also when the offensive side searches for promising moves in the search tree. If the offensive side makes a move, a CTSS solution exists for the associated position. Meanwhile, relevance-zone search can be performed on the offensive side to judge whether the defensive side can defend against the CTSS solution. Such a search can significantly assist the offensive side in seeking useful moves and can improve overall search efficiency.

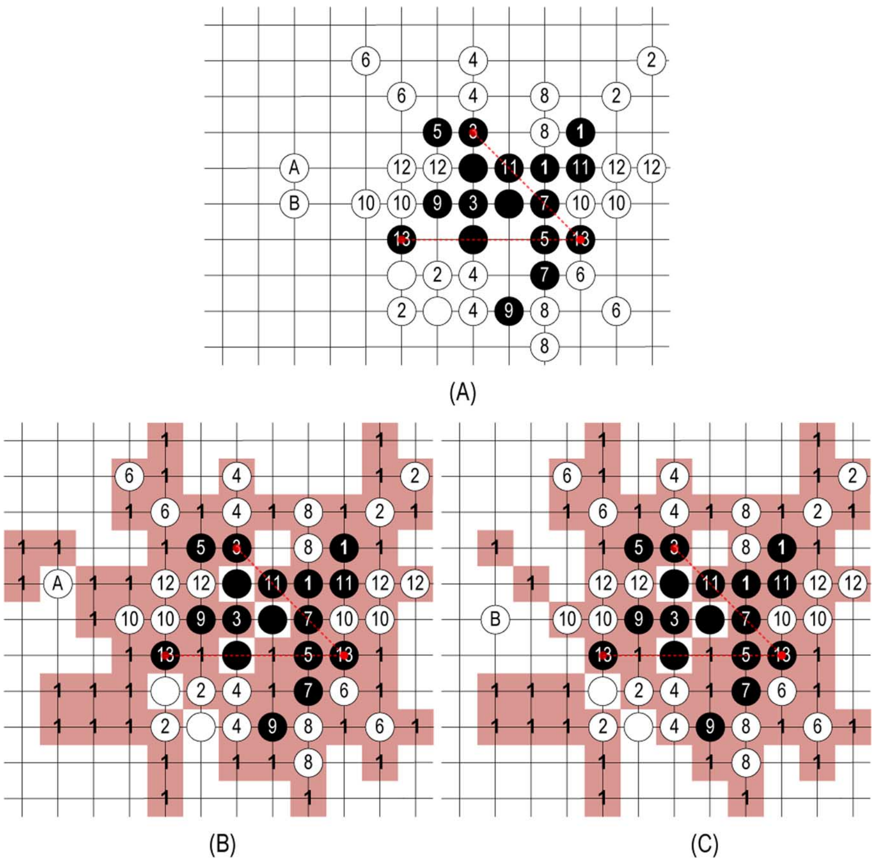

Fig. 23. Example of relevance-zone search where Attacker has no threat. (A) The black CTSS solution based on white plays a move on cells A and B. (B) The Z_1 based on white plays a stone on cell A and the CTSS solution of (A). (C) The Z_1 based on white plays a stone on cell B and the CTSS solution of (A).

TABLE II
RESULTS OF KAVALAN IN THE LAST TOURNAMENTS

Tournaments	Participants	Kavalan's rank
ICGA 2010 (Kanazawa)	8	2
ICGA 2009 (Pamplona)	6	5
ICGA 2008 (Beijing)	10	5
ICGA 2007 (Amsterdam)	4	3

VI. EXPERIMENTS

This section explains the experimental method and results. This paper designs two types of experiments. The first experiment involves comparing searching algorithms. This experiment tests accuracy and efficiency. The second experiment focuses on performance.

This paper developed an AI player capable of playing the game Connect6, named Kavalan. Kavalan has participated in the Computer Olympiad, Connect6, four times, and Table II lists the results [10]. The two-stage MCTS was developed after the tournament in 2009, and it won silver in 2010.

The experiments were performed on 2.0 GHz with 2GB of memory running Windows XP.

A. Comparison of the Searching Algorithm for the TSS Solution

The first experiment aims to analyze the accuracy and efficiency for positions with TSS solution in two-stage MCTS. This paper gathers the latest year puzzles from a Taiwanese Connect6 website [16], and obtains 30 puzzles for use as an experimental test benchmark for the algorithm.

To examine the ability of two-stage MCTS to search TSS solution, this paper divides these puzzles into the T2 and TSS solutions. For the 30 puzzles, 12 questions belong to T2 solutions, and 18 belong to TSS solutions.

For these puzzles with T2 solution, only the use of CTSS can obtain solutions. Therefore, this paper considered five examples of double-threat solutions that could not be obtained by CTSS from the initial position. The five examples are ITSS-Example-1 to ITSS-Example-5, and Appendix A presents these examples in Figs. 25–29, respectively. Therefore, Figs. 25–29 are the puzzles in which CTSS cannot find the double-threat solution from the initial position, but the solution can be found by ITSS.

Algorithm Setup: The maximum probing positions of BFS and DFS are limited to 600 000. Basic DFS has no depth limit, but the outcome is bad for these puzzles with T2 solutions. Therefore, this paper tests the results of limiting the search depth of DFS from six to 19. Table III lists the outcome of DFS, including the best test result. Besides, the search of DFS and BFS is limited to double-threat moves, and excludes other candidate moves.

The simulation times of traditional MCTS, two-stage MCTS (Type I), and two-stage MCTS (Type II) are limited to 60 000. For these different types of MCTS, this paper uses six controlling variables to control MCTS. The control variables of two-stage MCTS are shown in Table III.

TABLE III
THE CONTROL VARIABLE AND ITS VALUE

Control Variable		Value
The number of probing Cells		20
Exploration Coefficient of UCT Selection		1.2
Heuristic Value to double-threat moves		200
Heuristic Value to single-threat moves		300
The depth of playout (from initial position)		30
Back-Propagation	Determined node – Attacker Win	10
	Determined node – Attacker Fail	-5
	Playout – Attacker Win	1
	Playout – Attacker Fail	-1

TABLE IV
SEARCH ALGORITHM COMPARISON FOR THE T2 SOLUTION

Algorithms / Puzzles	DFS with depth limit to 14	BFS	Traditional MCTS	2-Stage MCTS (Type I)	2-Stage MCTS (Type II)
2008-Q1-1-1	0.219 (4785)	0.016 (202)	0.188 (85)	0.001 (84)	0.001 (84)
2008-Q1-1-2	0.391 (8062)	11.469 (70263)	X	0.031 (554)	0.031 (554)
2008-Q1-2-1	0.984 (23071)	0.078 (884)	0.031 (66)	0.016(89)	0.001 (89)
2008-Q1-2-2	2.906 (70184)	0.953 (10003)	0.078 (1321)	0.391 (4475)	0.406 (4475)
2008-Q1-3-1	4.281 (103946)	0.078 (705)	X	0.031 (563)	0.031 (563)
2008-Q1-3-2	5.250 (120907)	0.031 (581)	11.656 (72242)	0.016 (27)	0.001 (27)
2008-Q2-1-1	1.203 (26558)	1.672 (15734)	0.375 (952)	0.001 (54)	0.001 (54)
2008-Q2-1-2	1.719 (43917)	3.062 (15661)	0.047 (101)	0.001 (50)	0.001 (50)
2008-Q2-2-1	0.046 (914)	0.047 (487)	0.046 (109)	0.001 (56)	0.016 (56)
2008-Q2-2-2	0.609 (13320)	0.250 (2263)	0.016 (189)	0.016 (29)	0.001 (29)
2008-Q3-1-1	9.813 (233264)	0.016 (168)	0.078 (1209)	0.031 (208)	0.031 (208)
2008-Q3-1-2	0.001 (26)	0.031 (202)	0.078 (86)	0.001 (18)	0.001 (18)
ITSS-Example-1	1.813 (43669)	0.609 (6420)	X	0.031 (353)	0.016 (353)
ITSS-Example-2	0.547 (12921)	2.109 (23154)	1.453 (4482)	1.000 (5286)	0.625 (6243)
ITSS-Example-3	0.578 (13942)	11.234 (110643)	23.734 (180571)	5.375 (41866)	12.563 (76088)
ITSS-Example-4	0.828 (19010)	0.094 (1094)	X	0.250 (724)	0.406 (754)
ITSS-Example-5	X	80.156 (593170)	22.766 (125121)	0.719 (8954)	1.750 (9412)
Total time	X	101.905	X	7.912	15.882

The playout step begins when MCTS enters a position that has not yet been identified as a determined position in the search tree. Elaborating an efficient playout strategy is a difficult issue. In Kavalan, the number of moves per playout is limited because the purpose of playout is estimating the given position. For Connect6, the TSS solution under a position may exceed one, so finding the nearest solution is helpful. Therefore, if playout requires excessive computing resources to estimate positions, performance is adversely affected.

1) Double-Threat TSS: To understand the efficiency of the two-stage MCTS search algorithm, this paper compares it with the traditional MCTS and the brute-force methods, BFS and DFS. Table IV lists the results with the unit of search time is seconds. The number under the search time (inside the parenthesis) is the number of node expansions, including the CTSS search positions for double-threat moves.

At the beginning of developing MCTS, this paper does not use H_V value to distinguish between threat and nonthreat moves. The results demonstrate that solutions are unavailable for most puzzles. This paper thus finds that it is difficult to find sudden-death property of Connect6 in a position if simply using playout to control search tree development.

The difference between the two types of MCTS during the first stage is that type I only generates double-threat moves while type II simultaneously generates both double-threat and single-threat moves.

When the T2 solution occupies the initial position, Type I searches fewer positions while Type II searches more. However, if double-threat moves are assigned a H_V value and given regular search times during the first stage, it is possible to improve the efficiency when using Type II to search for those puzzles with T2 solution.

The experimental results demonstrate that the extra calculation time associated with using CTSS to assess double-threat moves does not compromise the searching efficiency, but rather significantly improves it.

Although two-stage MCTS does not offer the fastest searching in all these testing puzzles, the total time clearly demonstrates that regardless of type of two-stage MCTS, MCTS has superior search efficiency to BFS and DFS. Furthermore, ITSS-Example-5 clearly demonstrates that ITSS can demonstrate greater search efficiency in the case of more difficult T2 solutions.

The experimental results show that new Kavalan using ITSS can solve 100% puzzles with T2 solution.

2) Single-Threat TSS: Among the 18 puzzles of TSS solutions, this paper excludes two, 2008-Q1–1–3 and 2008-Q3–1–4, because they lack TSS solution. Table V lists the results.

Searching single-threat solution is a more difficult question. Because Type I of two-stage MCTS does not generate single-threat first in candidate moves, it is hard to search for the TSS solution based on the experimental results. Type II of two-stage MCTS generates single-threat moves during the first stage. Thus, it can focus its search on all threat moves.

The experimental results show that relevance-zone search is helpful in searching for single-threat moves. The new Kavalan using Type II of two-stage MCTS with relevance-zone search can solve 75% (12/16) of puzzles with TSS solutions.

3) The Efficient Analysis in MCTS: The experimental result shows that the two-stage MCTS works in Connect6. A node expansion involves the search of CTSS for examining double-threat moves and the playout for nondetermined moves. From the Table V, the average time for nodes expansion is about 8300 nodes/s whether the relevance-zone search is used or not for proving some positions.

B. Performance Analysis in Kavalan

Threat Space Search is the most common search method in Connect6 as described in [17], [20], [23], and [26], and relevance zones are used to accelerate the proof for some posi-

TABLE V
SEARCH ALGORITHM COMPARISON FOR THE TSS SOLUTION

Algorithms / Puzzles	2-Stage MCTS (Type I)		2-Stage MCTS (Type II)	
	Not used	Relevance-zone search	Not used	Relevance-zone search
2008-Q1-1-4	X	397.438 (225273)	X	3.047 (32767)
2008-Q1-1-5	X	X	X	X
2008-Q1-2-3	X	X	X	173.938 (1224641)
2008-Q1-2-4	0.031 (152)	0.016 (154)	0.016 (152)	0.001 (154)
2008-Q1-2-5	1453.968 (13624848)	3.453 (43360)	40.719 (531499)	1.267 (14538)
2008-Q1-3-3	278.593 (952409)	246.063 (841201)	194.906 (1767462)	178.875 (1697174)
2008-Q1-3-4	X	X	101.344 (581786)	86.063 (399731)
2008-Q1-3-5	X	X	X	1912.343 (16786491)
2008-Q2-1-3	X	X	2.968 (30543)	2.734 (20458)
2008-Q2-1-4	13.812 (120557)	54.656 (298524)	9.703 (113115)	3.063 (11941)
2008-Q2-1-5	X	X	X	X
2008-Q2-2-3	X	X	X	X
2008-Q2-2-4	X	X	4.562 (71348)	0.141 (2217)
2008-Q2-2-5	X	X	X	X
2008-Q3-1-3	25.703 (174044)	14.266 (96601)	15.593 (294480)	0.563 (4528)
2008-Q3-1-5	X	X	25.703 (406548)	8.797 (81888)

tions. This paper also uses these two methods to conduct Connect6 search. The MCTS architecture is also used to develop the search tree by integrating TSS and by proving the relevance zones. The proposed search architecture used to fit the sudden-death property of Connect6 is called two-stage MCTS.

When using the concept of stages to generate candidate moves, the search tree can be developed by using various searching algorithms such as proof number search, PNS or Monte Carlo Tree Search, MCTS. Here, MCTS is used because it has proven effective in numerous game searches.

In Connect6, threats are the key to proving winning positions; therefore, threat-based searches such as TSS or relevance zones are aimed at proving winning positions. Although MCTS is not aimed at proving winning positions, it is effective for finding the most promising move.

Fig. 24 is a search tree for two-stage MCTS. The positions H, V, and W are proved as winning positions. Therefore, the offensive side chooses to play move C based on the search tree because move C gets more PV value in back-propagation as described in Section IV. In two-stage MCTS, simulation times for move C are bigger if the resources are unlimited.

In contest, because the resources are limited, whether the search should focus on threat-moves must be determined on the simulation times and the PV rate of the root node. The prediction of PV rate for the root node may be incorrect in the initial MCTS simulations because the prediction is based mostly on playout results. The prediction accuracy for the root node increases with the development of the search tree if the structure of the developed tree is correct.

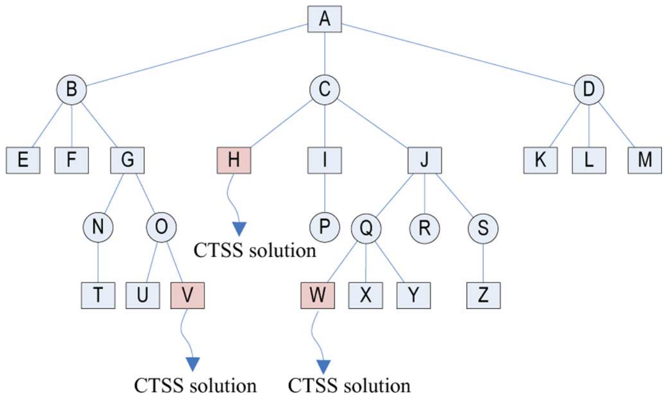

Fig. 24. Search tree for two-stage MCTS.

From observation, when the simulation times for a position exceed 6000, the PV rate is reliable. Therefore, if the simulation times for the root node exceed 6000 and the PV rate for the offensive side is less than -0.5, the searching strategy changes from attacking to defending. This strategy is only adopted in contest because the time is limited.

Based on the two-stage MCTS techniques described in this paper, the new Kavalan is significantly stronger than Kavalan 2009, and can beat it about 95% of the time. A total of 40 contests were tested, and players move alternately starting with black. Kavalan 2009 uses only double-threat moves in threat-based search, and lacks the ability to search for single-threat solutions.

Furthermore, this paper tested the playing strength of the proposed two-stage MCTS against the most recent version of X6 (v1.4.1.d1), at level 14. X6 is the first rank of the 12th Computer Olympiad, Connect6 in 2007, and it was developed by Liou and Yen [22]. The result of the contests is that new Kavalan won seven of 10 games. Clearly, two-stage MCTS significantly improved the search performance of Kavalan.

VII. CONCLUSION AND FUTURE WORK

A. Conclusion

This paper presents a new search architecture for Monte Carlo Tree Search in Connect6, and calls it two-stage MCTS. According to the experimental results, the proposed two-stage MCTS search structure has the following three advantages.

- The search efficiency of TSS solution
  This paper proposed two-stage MCTS search structure for Connect6 based on its feature of sudden-death. Two-stage MCTS significantly outperforms traditional MCTS in the case of positions for which TSS solutions exist.
  The proposed structure uses stages to develop candidate moves. According to the experimental results, the search efficiency significantly exceeds that of traditional MCTS for positions for which TSS solutions exist.
  According to the experimental results, new Kavalan can solve 100% puzzles with double-threat solutions, and can solve 75% (12/16) of puzzles with single-threat solution for the test puzzles presented here.
- A new search structure for double-threat TSS

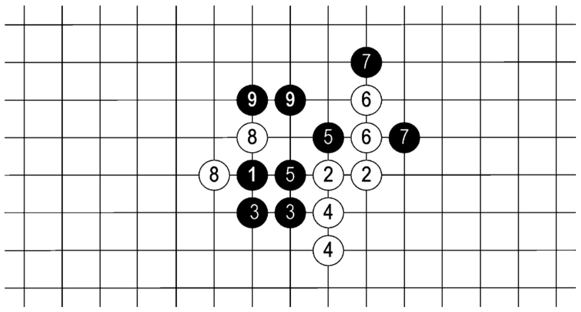
Fig. 25. ITSS-Example-1: White to play and win.

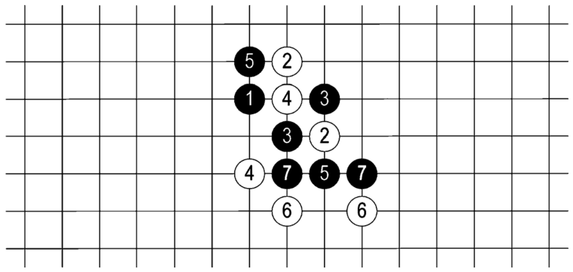
Fig. 26. ITSS-Example-2: White to play and win.

For the position for which double-threat solution exists, the ITSS search structure provided in this paper is clearly more efficient than BFS, DFS, or traditional MCTS.

According to the two-stage MCTS search structure, if the solution can be obtained via CTSS under the initial position, it can also be obtained through CTSS with an equivalent search time.

Because ITSS develops T2-subtree via iteration, it can avoid extending all the search branches and wasting calculation time. Though using CTSS to evaluate double-threat moves involves repetitive calculations, the time costs of this additional calculation are outweighed by the other time savings.

- Relevance-zone search

  To accelerate the demonstration of a defensive side (OR-Node) fails versus offensive side (AND-Node) wins, this paper uses the relevance-zone search.

  According to heuristic knowledge of Connect6, this paper proposes using relevance-zone search to accelerate the demonstration of whether offensive side wins or defensive side fails. The experiment proves that this approach can avoid unnecessary searches of extensive state space, reducing time spent on the demonstration.

B. Future Work

This paper divides candidate moves using a two-stage scheme. Because three kinds of candidate moves exist for Connect6, candidate moves can be divided into three-stages. Whether three-stages can improve search efficiency is worth discussing.

Furthermore, ITSS is the search architecture for double-threat TSS, and can be combined with numerous search methods to develop the search tree. This paper only used MCTS to develop ITSS. Whether PNS can improve the efficiency of T2 or TSS solutions deserves exploration.

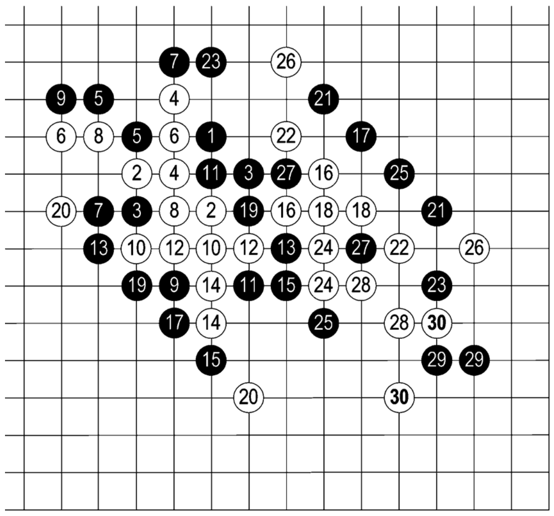
Fig. 27. ITSS-Example-3: Black to play and win.

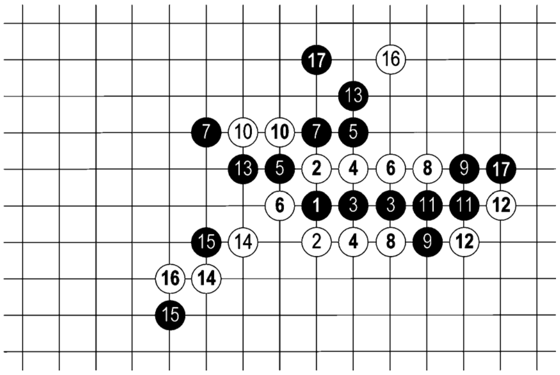
Fig. 28. ITSS-Example-4: White to play and win.

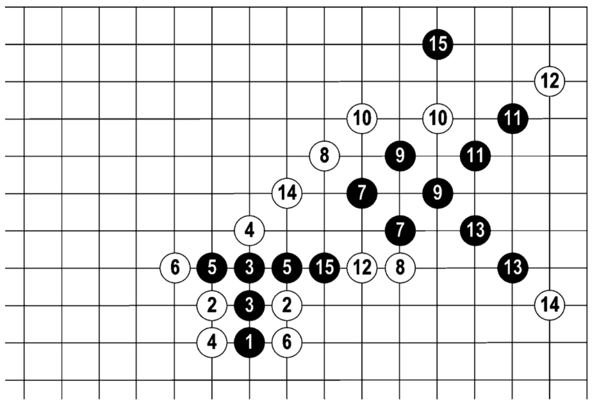
Fig. 29. ITSS-Example-5: White to play and win.

APPENDIX
ITSS EXAMPLES

The figures above are the ITSS test puzzles in the experiments. Therefore, from Fig. 25 to Fig. 29 are the puzzles in which CTSS cannot find the T2 solution, but the T2 solution can be found by ITSS discussed in Section III. The numbers in figure represent the order of moves.

References

[1] L. V. Allis, "Searching for Solutions in Games and Artificial Intelligence," Ph.D. dissertation, Univ. of Limburg, Maastricht, The Netherlands, 1994.

[2] L. V. Allis, H. J. van den Herik, and M. P. H. Huntjens, "Go-Moku solved by new search techniques," *Comput. Intell.*, vol. 12, pp. 7–23, 1996.

[3] B. Arneson, R. Hayward, and P. Henderson, "WolveWins Hex tournament," *ICGA J.*, vol. 32, no. 1, pp. 49–53, Mar. 2009.

[4] B. Bouzy and G. M. J.-B. Chaslot, "Monte-Carlo go reinforcement learning experiments," in *Proc. IEEE 2006 Symp. Comput. Intell. Games*, Reno, NV, 2006, pp. 187–194.

[5] G. Chaslot, M. Winands, J. van den Herik, J. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," *New Math. Natural Comput.*, vol. 4, no. 3, pp. 343–352, 2008.

[6] R. Coulom, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Proc. 5th Comput. Games Conf. (CG 2006)*, Berlin, Germany, 2007.

[7] R. Coulom, H. van den Herik, J. W. H. M. Uiterwijk, M.H.M. Winands, and M. P. D. Schadd, Eds., "Computing elo ratings of move patterns in the game of go," in *Proc. Comput. Games Workshop 2007 (CGW 2007)*, Amsterdam, The Netherlands, 2007, pp. 113–124.

[8] J.-D Fossel, "Monte-Carlo Tree Search Applied to the Game of Havannah," B.Sc. thesis, Univ. of Limburg, Maastricht, The Netherlands, 2010.

[9] H. J. van den Herik, J. W. H. M. Uiterwijk, and J. V. Rijswijck, "Games solved: Now and in the future," *Artif. Intell.*, vol. 134, no. (1–2), pp. 277–311, 2002.

[10] International Computer Games Association (ICGA) ICGA website [Online]. Available: http://www.grappa.univ-lille3.fr/icga/

[11] L. Kocsis and C. Szepesvari, J. Füurnkranz, T. Scheffer, and M. Spiliopoulou, Eds., "Bandit based Monte-Carlo planning," in *Proc. Mach. Learn.: ECML 2006*, Berlin, Germany, 2006, pp. 282–293.

[12] C. S. Lee, M. H. Wang, C. Chaslot, J. B. Hoock, A. Rimmel, O. Teytaud, S. R. Tsai, S. C. Hsu, and T. P. Hong, "The computational intelligence of MoGo revealed in Taiwan's computer Go tournaments," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 1, pp. 73–89, Mar. 2009.

[13] P.-H. Lin and I.-C. Wu, "NCTU6 wins in the man-machine Connect6 championship 2009," *ICGA J.*, vol. 32, no. 4, pp. 230–232, Dec. 2009.

[14] R. J. Lorentz, H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, Eds., "Amazons discover monte-carlo," in *Proc. Comput. Games (CG 2008)*, 2008, 5131 of Lecture Notes Comput. Sci. (LNCS), pp. 13–24.

[15] P. S. San, R. Galan, D. Rodriguez-Losada, F. Matia, and A. Jimenez, "Efficient search using bitboard models," in *Proc. XVIII Int. Conf. Tools AI*, Washington, DC, 2006, pp. 132–138.

[16] Taiwan Connect6 Association, Connect6 homepage [Online]. Available: http://www.connect6.org/

[17] T. Thomsen, "Lambda-search in game trees—With application to go," *ICGA J.*, vol. 23, no. 4, pp. 203–217, 2000.

[18] F. Van Lishout, G. Chaslot, and J. W. H. M. Uiterwijk, "Monte-Carlo tree search in backgammon," in *Comput. Games Workshop*, Amsterdam, The Netherlands, 2007, pp. 175–184.

[19] M. Winands, Y. Björnsson, and J.-T. Saito, "Monte-Carlo tree search solver," in *Proc. 6th Int. Comput. Games Conf. (CG'08)*, Beijng, China, Sep. 2008, pp. 25–36.

[20] I.-C. Wu, D.-Y. Huang, and H.-C. Chang, "Connect6," *ICGA J.*, vol. 28, no. 4, pp. 234–241, 2005.

[21] I.-C. Wu and S.-J. Yen, "NCTU6 wins Connect6 tournament," *ICGA J.*, vol. 29, no. 3, pp. 157–158, Sep. 2006.

[22] I.-C. Wu and S.-J. Yen, "X6 wins Connect6 tournament," *ICGA J.*, vol. 30, no. 2, pp. 116–117, Jun. 2007.

[23] I.-C. Wu and D.-Y. Huang, "A new family of k-in-a-row games," in *Proc. 11th Adv. Comput. Games Conf.*, Taipei, Taiwan, 2008.

[24] I.-C. Wu and P.-H. Lin, "NCTU6-Lite wins Connect6 tournament," *ICGA J.*, vol. 31, no. 4, pp. 240–243, 2008.

[25] I.-C. Wu, C.-P. Chen, P.-H. Lin, K.-C. Huang, L.-P. Chen, D.-J. Sun, Y.-C. Chan, and H.-Y. Tsou, "A volunteer-computing-based grid environment for Connect6 applications," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE-09)*, Vancouver, BC, Canada, Aug. 29–31, 2009.

[26] I.-C. Wu and P.-H. Lin, "Relevance-Zone-Oriented proof search for Connect6," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 3, Sep. 2010.

[27] C.-M. Xu, Z.-M. Ma, and X.-H. Xu, "A method to construct knowledge table-base in k-in-a-row games," in *Proc. 2009 ACM Symp. Appl. Comput.*, Honolulu, HI, 2009, pp. 929–933.

[28] S.-J. Yen and J.-K. Yang, "The bitboard design and bitwise computing in Connect6," in *Proc. 14th Game Program. Workshop (GPW-09)*, Kanagawa, Japan, Nov. 13–15, 2009, pp. 95–98.

[29] P. Zhang and K. Chen, "Monte-Carlo go tactic search," *New Math. Natural Comput. J.*, vol. 4, no. 3, pp. 359–367, Nov. 2008.

Shi-Jim Yen received the B.Sc. degree in computer science and information engineering from Tamkang University, Taipei City, Taiwan, in 1991, and the M.Sc. degree in electrical engineering from National Central University, Jhongli City, Taiwan, in 1993. He also received the Ph.D. degree in computer science and information engineering from National Taiwan University, in 1999.

He is currently an Associate Professor in the Department of Computer Science and Information Engineering at the National Dong Hwa University, Hualien, Taiwan. He has specialized in artificial intelligence and computer games. In these areas, he has published over 50 papers in international journals or conference proceedings. He is a 6-dan Go player. He served as a Workshop Chair on 5th International Conference on Grid and Pervasive Computing in 2010, and a Workshop Chair of 2010 International Taiwanese Association for Artificial Intelligence (TAAI) Conference. He serves as a Workshop Cochair of 2011 IEEE International Conference on Fuzzy Systems. He is the Chair of the IEEE Computational Intelligence Society (CIS) Emergent Technologies Technical Committee (ETTC) Task Force on Emerging Technologies for Computer Go in 2010.

Jung-Kuei Yang received the B.Sc. degree in industrial engineering and the M.Sc. degree in information management from Dayeh University, Dacun, Taiwan, in 1994 and 1996, respectively. He is currently working towards the Ph.D. degree in the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan.

He is also an instructor in the Department of Applied Foreign Languages, Lan Yang Institute of Technology, I Lan, Taiwan. He is the current Chief Designer of the Connect6 program Kavalan. His research interests include artificial intelligence and computer games.