Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2016, Article ID 5216861, 14 pages
http://dx.doi.org/10.1155/2016/5216861

Research Article

Turn-Based War Chess Model and Its Search Algorithm per Turn

Hai Nan,[1,2] Bin Fang,[1] Guixin Wang,[2] Weibin Yang,[3] Emily Sarah Carruthers,[4] and Yi Liu[5]

[1]*College of Computer Science, Chongqing University, Chongqing 400044, China*
[2]*Department of Software Engineering, Chongqing Institute of Engineering, Chongqing 400056, China*
[3]*College of Automation, Chongqing University, Chongqing 400044, China*
[4]*College of International Education, Chongqing University, Chongqing 400044, China*
[5]*PetroChina Chongqing Marketing Jiangnan Company, Chongqing 400060, China*

Correspondence should be addressed to Bin Fang; fb@cqu.edu.cn

Received 20 August 2015; Revised 25 December 2015; Accepted 10 January 2016

Academic Editor: Michela Mortara

War chess gaming has so far received insufficient attention but is a significant component of turn-based strategy games (TBS) and is studied in this paper. First, a common game model is proposed through various existing war chess types. Based on the model, we propose a theory frame involving combinational optimization on the one hand and game tree search on the other hand. We also discuss a key problem, namely, that the number of the branching factors of each turn in the game tree is huge. Then, we propose two algorithms for searching in one turn to solve the problem: (1) enumeration by order; (2) enumeration by recursion. The main difference between these two is the permutation method used: the former uses the dictionary sequence method, while the latter uses the recursive permutation method. Finally, we prove that both of these algorithms are optimal, and we analyze the difference between their efficiencies. An important factor is the total time taken for the unit to expand until it achieves its reachable position. The factor, which is the total number of expansions that each unit makes in its reachable position, is set. The conclusion proposed is in terms of this factor: Enumeration by recursion is better than enumeration by order in all situations.

1. Introduction

Artificial intelligence (AI) is one of the most important research fields in computer science, and its related algorithms, technologies, and research results are being widely used in various industries, such as military, psychological, intelligent machines and business intelligence. Computer games, known as “artificial intelligence’s drosophila,” are an important part of artificial intelligence research. With the increasing development of computer hardware and research methods, artificial intelligence research in traditional board games has seen some preliminary results. Alus et al. [1] have proven that Go-Moku’s AI, provided it moves first, is bound to win against any (optimal) opponent by the use of threat-space search and proof-number search. The Monte Carlo Tree Search (MCTS) method, based on UCT (UCB for tree search), has improved the strength of 9×9 Go, close to the level of a professional Kudan [2].

Computer game based on artificial intelligence is a sort of deterministic turn-based zero-sum game, containing certain information. Man-machine games can be classified into two categories: two-player game and multiplayer game, according to the number of game players. Most traditional chesses, such as the game of Go and Chess, belong to the two-player game category, to which α-β search based on min-max search and its enhancement algorithms such as Fail-Soft α-β [3], Aspiration Search [4], Null Move Pruning [4], Principal Variation Search [5], and MTD(f) [5] are usually applied. On the contrary, Multiplayer Checkers, Hearts, Sergeant Major, and so forth belong to the multiplayer game category [6] which runs according to a fixed order of actions, with participants fighting each other and competing to be the sole winner of the game. Its search algorithm involves Max^n search [7], Paranoid [6], and so forth. The α-β search previously mentioned based on min-max search is a special case based on the Max^n search and shadow pruning algorithms [7]. Man-machine

(a) Wargaming

(b) SLG game "Battle Commander"

Figure 1: Wargaming and SLG game.

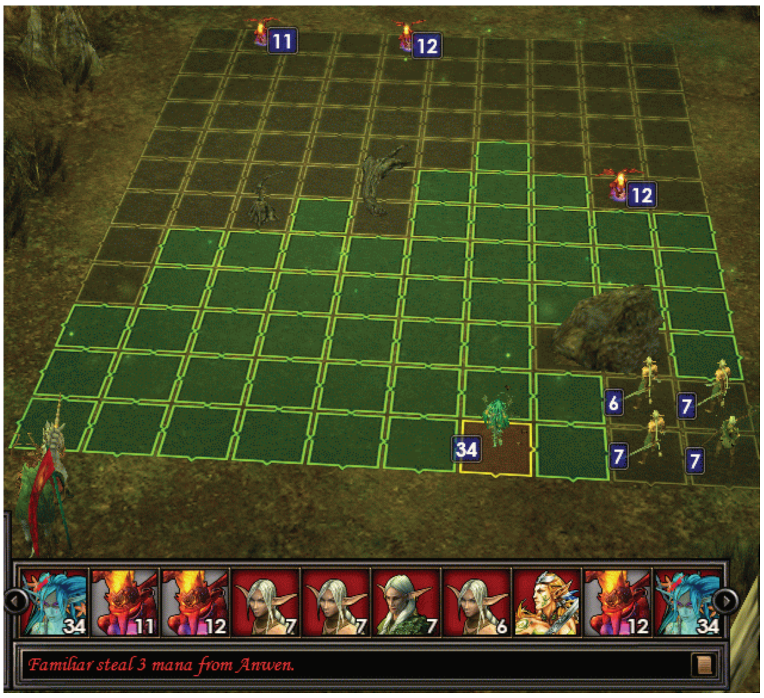


Figure 2: SLG game "Heroes of Might & Magic."

games can also be classified into two categories: classic board games and new board games, according to the game content. Classic board games involve Go, chess, backgammon, checkers, and so forth, which are widespread and have a long history. While other board games such as Hex [8], Lines of Action [9], and Scotland Yard [10] are ancient games, with the rapid development of modern board games and mobile client applications they have been accepted by more and more players until their prevalence is comparable to that of the classic board game. The machine game algorithms of the board games listed above are all based on α-β search and their enhancement algorithms. The MCTS algorithm has developed rapidly in recent years, being used increasingly in these board games and getting increasingly satisfactory results [8–10].

However, not all board games can be solved with the existing algorithms. Turn-based strategy games (TBS), as well as turn-based battle simulation games (SLG) (hereinafter collectively referred to as turn-based strategy games), originated from the wargames [11] that swept the world in the mid-19th century (Figure 1(a) shows an example of a wargame). With the introduction of computer technology, this new type of game, turn-based strategy game, has flourished (Figure 1(b) shows a famous TBS game called "Battle Commander," and Figure 2 shows the popular SLG game "Heroes of Might & Magic"). Now, TBS games have become the second most famous type of game after RPGs (role-playing games). With the blossoming of mobile games, TBS games will have greater potential for development in the areas of touch-screen operation, lightweight, fragmented time, and so on. The content of a TBS game generally comprises two levels: strategic coordination and tactical battle control. The latter level, whose rules are similar to those of board games, for example, moving pieces on the board, beating a specified enemy target for victory, and turn-based orders, is called the turn-based war chess game (TBW). The artificial intelligence in TBW is an important component of TBS games. The AI of modern TBS games is generally not so intelligent, of which the fundamental reason is that the AI in its local battle (TBW) is not so intelligent. How to improve the TBW's artificial intelligence, thus improving the vitality of the entire TBS game industry, is an urgent problem that until now has been overlooked.

Currently, the study of artificial intelligence in turn-based strategy games is mainly aimed at its macro aspect, and the research object is primarily the overall macro logistics, such as the overall planning of resources, construction, production, and other policy elements. The main research contents involve planning, uncertainty decisions, spatial reasoning, resource management, cooperation, and self-adaptation. However, studies on artificial intelligence for a specific type of combat in TBS are scarce, and the attention paid to researching the TBW units' moves, attacks, and presentation of the game round transformation, whose AI is precisely the worst of all parts of the AI in a large number of TBS games, is not enough. At present, the research related to TBW's behavior involves spatial reasoning techniques. Bergsma and Spronck [12] divided the AI of TBS (NINTENDO's Advanced Wars) into tactical and strategic modules. The tactical module essentially has to decide where to move units and what to attack. It accomplishes this by computing influence maps, assigning a value to each map tile to indicate the desirability for a unit to move towards the tile. This value is computed by an artificial neural network. However, they unfortunately do not provide any detail on how such a mechanism would work. Paskaradevan and Denzinger [13] presented a shout-ahead architecture based on two rule sets, one making decisions without communicated intentions and one with these intentions. Reinforcement learning is used to learn rule weights (that influence decision making), while evolutionary learning is used to evolve good rule sets. Meanwhile, based on the architecture, Wiens et al. [14] presented improvements that add knowledge about terrain to the learning and that also evaluate unit behaviors on several scenario maps to learn more general rules. However, both approaches are essentially based on rules for the artificial intelligence, resulting in a lack of flexibility of intelligent behaviors, a lack of generality as they depend on a game's custom settings, and, moreover, a lack of reasoning for more than one future turn, similar to common chess games.

At present, research on TBW's AI from the perspective of the multiround chess game method is scarce because a TBW's player needs to operate all his pieces during each round, which is an essential difference with other ordinary chess games. Thus, the number of situations generated by permutation grows explosively such that, from this perspective, the TBW's AI can hardly be solved during regular playtime by the game approach described previously.

This paper attempts to study TBW's AI from the perspective of the chess game method. This is because the TBW's rules have many similarities with other chess games, and the decision made every turn in a TBW can be made wisely as in other chess games. In this paper, we propose two enumeration methods in a single round: dictionary sequence enumeration and recursive enumeration, which is the fundamental problem in our new framework. The improvement in TBW's AI can not only bring more challenges to game players but also bear a new series of game elements, such as smart AI teammates, which will provide players with a new gaming experience.

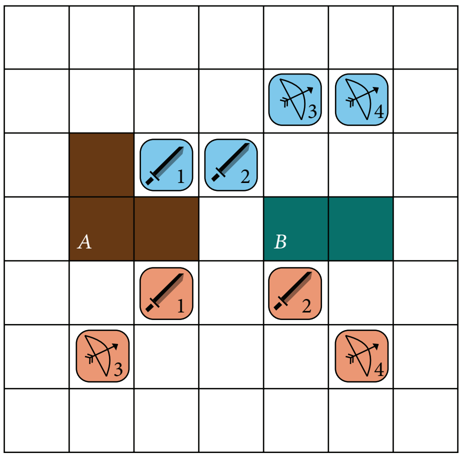


Figure 3: An example of TBW: Here are four red units and four blue units belonging to two players, respectively, on a square board. The units are divided into sword men and archers. The number written in the bottom right of each unit is the unit index. White tilts mean their terrain can be entered. However, ochre ones marked by "A" illustrate hilly areas no unit can enter. The dark green tilts marked by letter "B" illustrate lakes or rivers, which also cannot be entered, but archers can remotely attack the other side of the tilts.

A TBW game is essentially the compound of combinational optimization laterally and game tree search vertically (Section 3.2), which can be regarded as a programming problem of multiagent collaboration in stages and can be seen as a tree search problem with huge branching factor. Thus, the expansion and development of the traditional systems hidden behind TBW games will make the research more meaningful than the game itself.

This paper first summarizes the general game model for TBW and illustrates its key feature, that is, that the branching factor is huge in comparison with traditional chess games. Then, it puts forward two types of search algorithms for a single round from different research angles: the dictionary sequence enumeration method (Algorithm 2) and the recursive enumeration method (Algorithm 5). Ensuring invariability of the number of branches, Algorithm 5 has less extension operation of pieces than Algorithm 2 under a variety of conditions. The experiments also confirmed this conclusion.

2. Game Module of Turn-Based War Chess

2.1. Rules. TBW is played on a square, hexagonal, or octagonal tile-based map. Each tile is a composite that can consist of terrain types such as rivers, forests, and mountains or built up areas such as bridges, castles, and villages (Figure 3). Each tile imposes a movement cost on the units that enter them. This movement cost is based on both the type of terrain and the

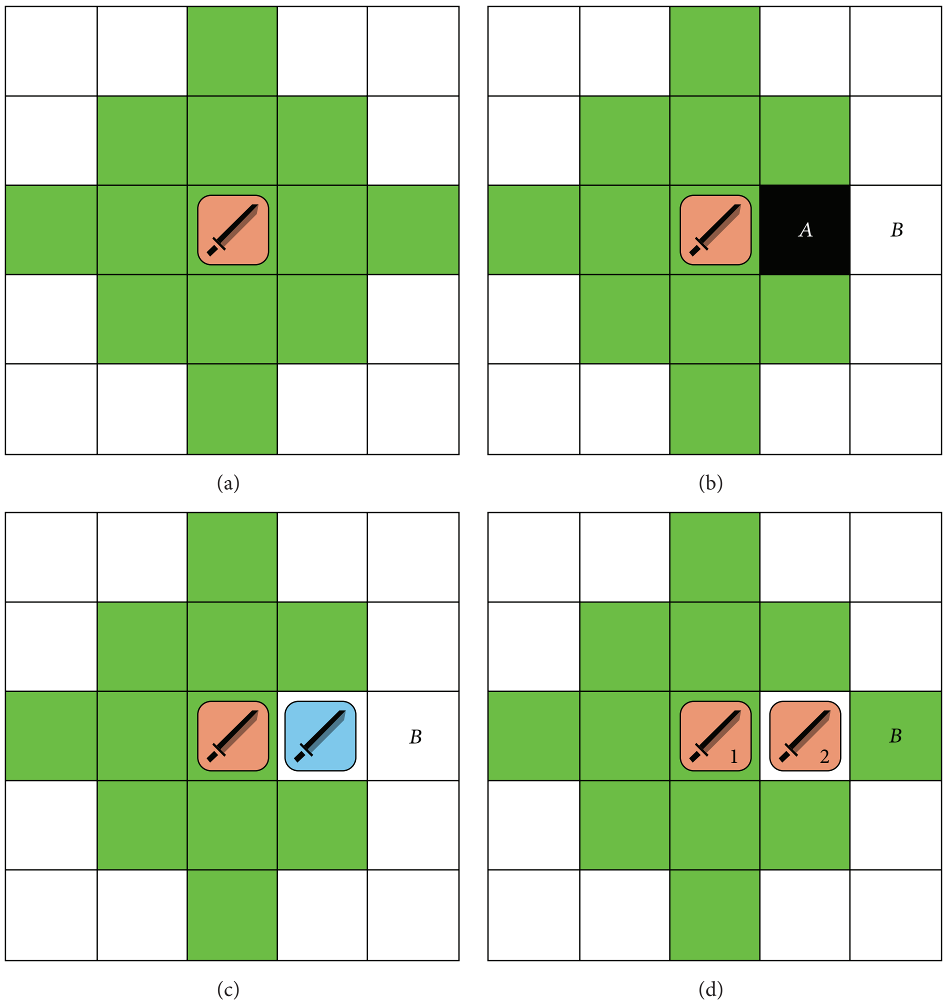


Figure 4: Green tilts illustrate the movement range of a swordsman, whose movement point is 2. The movement cost of each tilt is 1. (a) No obstacle. (b) Tilt *A* is an obstacle and thus tilt *B* is out of the movement range. (c) The swordsman cannot pass the enemy to reach tilt *B*. (d) The swordsman can pass units of the same side to reach tilt *B*.

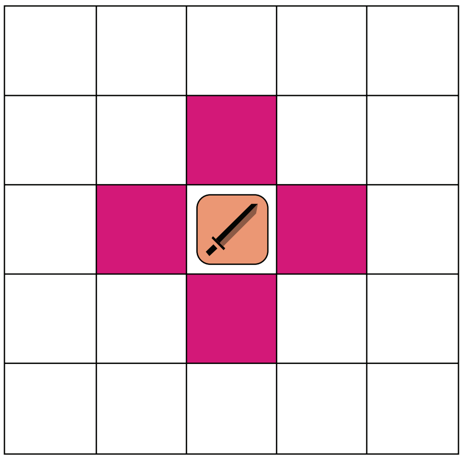
type of unit. Each tile is occupied by only one unit at the same time.

Each player in a TWB game controls an army consisting of many units. All units can either move or attack an enemy unit. Each unit has an allotted number of movement points that it uses to move across the tiles. Because different tiles have different movement costs, the distance that a unit can travel often varies. All of the tiles the unit can travel to compose a union of them called movement range (Figure 4), including the tile occupied by the unit itself. The movement range can generally be calculated by some algorithm such as breadth first search [18].

In addition to the movement point, each unit has its own health point (Hp) and attack power (ATK), which are numerical values and are various among the different units. Like movement range, a unit's attack range is another union of tiles to which the unit can attack from its current tile (Figure 5). Commonly, a unit's attack range is determined by its attack technique. Melee units, such as swordsmen, generally only attack adjacent units, and thus their attack range looks like that shown in Figure 5(a). Ranged attacking units, such as archers, can attack enemies as far as two or more tiles away (Figure 5(b)). Special units' attack range is also a special one. If a unit attacks another unit, it forfeits all of its movement points and cannot take any further actions that turn; therefore, if a unit needs to be moved to a different tile, it must perform the move action prior to performing an attack action. A unit also has the option not to take any attack action after its movement or even not to take any action and stay on its current tile.

Each unit attacked by its enemy must deduct its Hp by the attacking unit's ATK, which indicates the damage. When a unit's Hp is deducted to or below 0, this indicates that it is dead and must be removed from the board immediately. The tilt it occupied becomes empty and can be reached by other following units.

A game of TBW consists of a sequence of turns. On each turn, every player gets their own turn to perform all of the actions for each of their side's units. This is unlike ordinary board games, such as chess, where turns are only for selecting


(a) Swordsmen's attack range

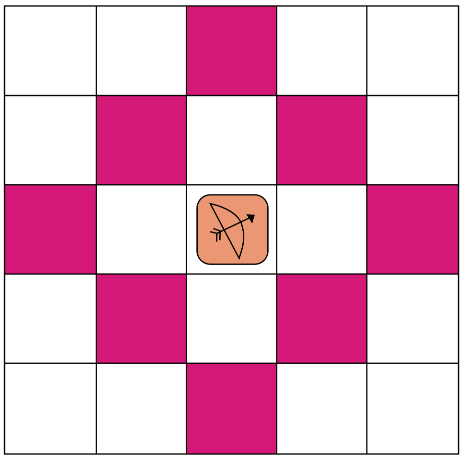

(b) Archer's attack range

FIGURE 5: Units' attack range.

a pawn to move. The opposing side does not get to perform its actions until the current side has finished. A player wins the game if all of the units or the leader units of the other player have died.

2.2. Setup and Notation. TBW is composed of the board and pieces (units). The board is considered as an undirected graph $G(V, E)$, where V is the set of vertices (tilts) and E is the set of edges that connect the neighboring tilts. Units are divided into two parties A (*Alex's*) and B (*Billie's*) according to which player they belong to. The sizes of the two parties are denoted as n_A and n_B, respectively, and the indexes of the units in the two parties are $1, 2, \ldots, n_A$ and $1, 2, \ldots, n_B$, respectively. Let $n = n_A + n_B$ be the total number of units. An assignment $M : [1, n] \rightarrow V$ places the units in unique tilts: $\forall i, j \in [1, n]$, $j \neq i : M(i) \in V, M(j) \in V, M(i) \neq M(j)$. For each unit, there is a movement range $R \subseteq V$, where $r = |R|$, and an attack range R_{atk}, where $r_{\text{atk}} = |R_{\text{atk}}|$.

Let Ord^C be a sequence of elements in set C such that Ord_i^C is the ith element of this sequence. We denote $n = |C|$, and thus $Ord_i^C \in C$ and $\forall i, j \in [1, n], j \neq i : Ord_i^C \neq Ord_j^C$.

Let Q^C be a set of all sequences of the elements in set C such that $Q^C = \{Ord^C\}$. Thus, $|Q^C| = P_n^n$.

Without loss of generality, let Ord^A be an action sequence of units in *Alex's* turn such that Ord_i^A expresses the index of the unit doing the ith action, where $i \in [1, n_A]$.

2.3. Game Tree Search. We try to use game tree search theory to research the AI of TBW. Game tree search is the most popular model for researching common chess games. In the game tree (Figure 6), nodes express states of the game board. Branches derived from nodes express selections of the move method. The root node is the current state, and the leaf nodes are end states whose depths are specifically expanded from the root. Both sides take turns. Even layer nodes belong to the current player (squares), while odd layer nodes belong to the other side (circles). If the leaf node is not able to give a win-lose-draw final state, an evaluation on a leaf node is needed to select the expected better method from the current state; this is the function of game tree search. Game tree search is based on min-max search, which is used to find the best outcome for the player and the best path leading to this outcome (Principal Variation) and, eventually, to find the corresponding move method in the root state (Root Move), that is, the best move for the player's turn [19].

It is not difficult to see that the evaluation and search algorithm are the most important parts of the game tree. For TBW, the evaluation factor of the state generally involves powers, positions, spaces, and motilities of units. The most common algorithms of game tree search are Alpha-Beta search [20] and Monte Carlo Tree Search [21], which can also be, although not directly, applied to TBW's search. This is because the branching factor of the search tree for TBW is huge and the common algorithms applied to TBW's search cause a timeout.

3. Features and Complexity Analysis

3.1. Complexity Analysis. A game of TBW consists of a sequence of turns. During each turn, every player gets their own turn to perform all of the actions for each of their side's units, which is the most important feature of TBW. The sequence of actions is vital. This is because the units cannot be overlapped; moreover, a different sequence of actions will also have a different state when a unit of another side is eliminated (Figure 7). Thus, during each side's turn, all of the plans of actions for its units are calculated by a permutation method. The amount of plans is estimated from both the worst and best situations (e.g., in the case of *Alex's* turn).

Step 1. Determine the sequence of actions: the total number is $P_{n_A}^{n_A} = n_A!$.

Step 2. Calculate the number of all plans of action in a specified action sequence.

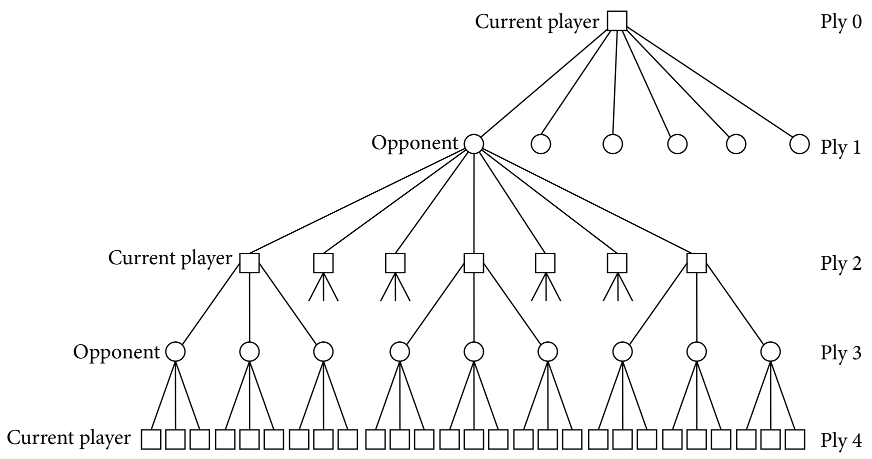


FIGURE 6: Game tree expanding to ply 4.

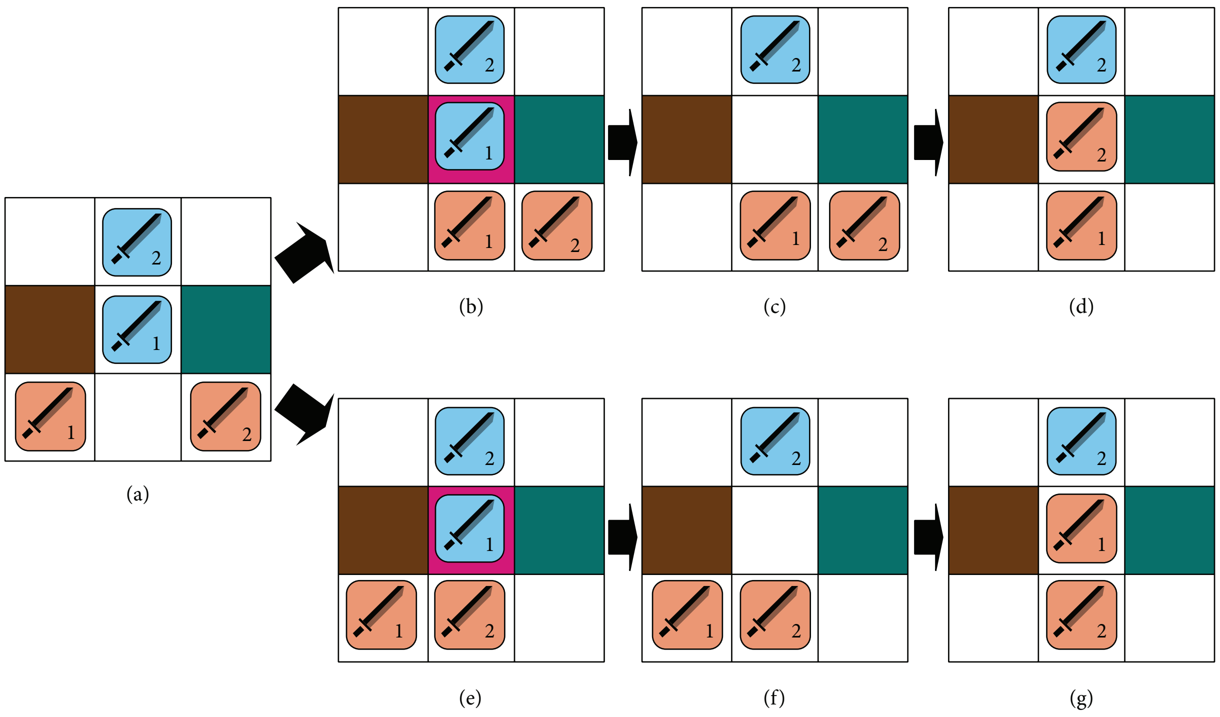


FIGURE 7: Effect of actions sequence. (a) The initial state of red side's turn. (b–d) Red swordsman number 1 acts and eliminates blue swordsman number 1, followed by red swordsman number 2. (e–g) Red swordsman number 2 acts and eliminates blue swordsman number 1, followed by red swordsman number 1.

Let R_i be the movement range of unit number i such that $r_i = |R_i|$. For simplicity, we assume that $r_1 = r_2 = \cdots = r_{n^A} = r$. In the worst case, the movement ranges of all of *Alex*'s units are independent without overlapping each other; that is, $\forall i, j \in [1, n_A]$, $j \neq i : R_i \cap R_j = \varnothing$. Moreover, in the attack phase, the amount of enemies that fall into each of *Alex*'s units reaches maximum. For example, on a four-connected board, a melee unit has at most four adjacent tilts around it, which are full of enemies. Then, the number of attack plans is at most five (including a plan not to attack any enemy), that is, $r_{\text{atk}} + 1$. According to the multiplication principle, the number of states expanding under a specified actions sequence is

$$\left[r\left(r_{\text{atk}} + 1\right)\right]^{n_A}. \tag{1}$$

According to Step 1, the number of actions sequences is $P_{n_A}^{n_A} = n_A!$ and thus, in the worst situation, the number of plans is

$$S_{\text{worst}} = \left[r\left(r_{\text{atk}} + 1\right)\right]^{n_A} \times n_A!. \tag{2}$$

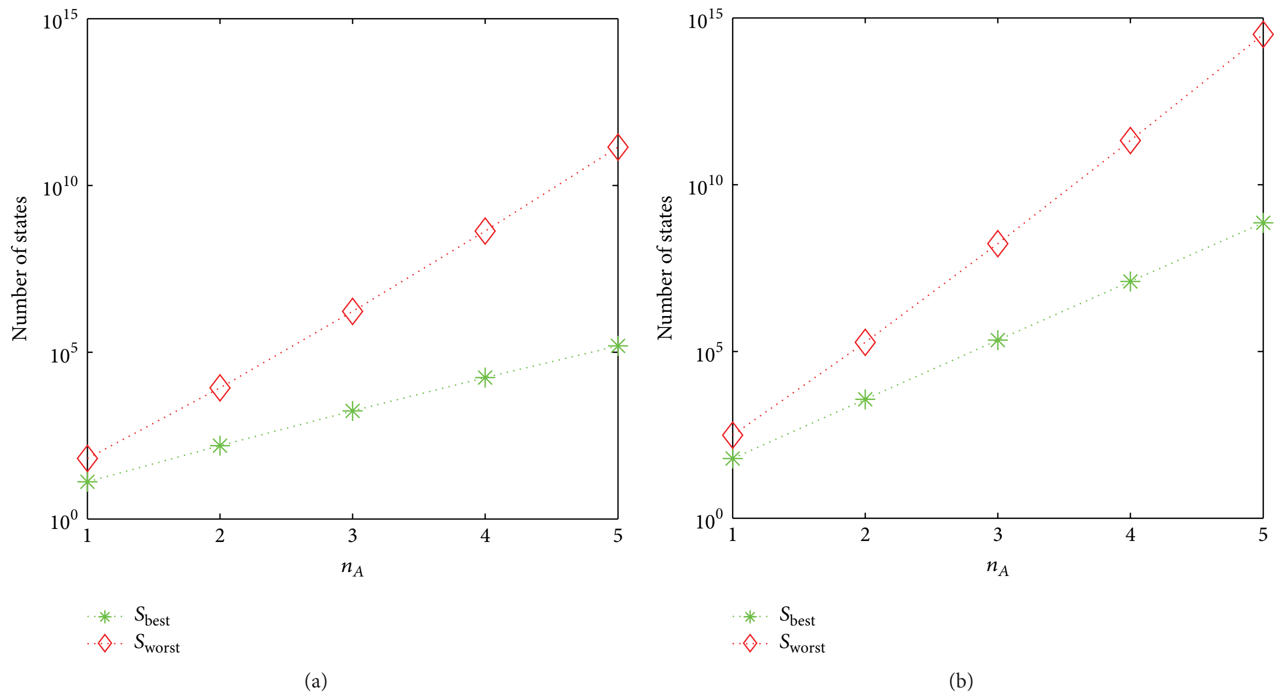


Figure 8: The growth trend of states S following n_A under different movement points. (a) Movement point: 2, movement cost: 1; (b) movement point: 5, movement cost: 1.

In the best situation, the movement ranges of all units overlap completely such that $R_1 = R_2 = \cdots = R_{n^A} = R$. Moreover, there are no enemies in the attack range of every unit. Thus, the amount of states can be calculated by the arrangement number $P_r^{n_A}$ such that we can select n_A from r positions to make all of the arrangements of the units. Therefore, the number of plans in the best situation is

$$S_{\text{best}} = \frac{r!}{(r - n_A)!}. \quad (3)$$

Above all, the total number of plans under all action sequences, denoted by S, is

$$S_{\text{best}} < S \leq S_{\text{worst}}. \quad (4)$$

In the following examples, we calculate the actual values of the total plans S. For "Fire Emblem," a typical ordinary TBW game, both sides have five units, and in the open battlefield, the movement range of each unit can reach at most 61 tilts (in that map, each tilt is adjacent to four other tilts, the movement point is 5, the movement cost of each tilt is 1, and there is no obstacle). Thus, $S_{\text{best}} \approx$ 710 million and $S_{\text{worst}} \approx$ 317 trillion. Assuming that the average computing time for searching a plan is 200 nanoseconds, searching all plans for one side's turn will then take from 2.4 minutes to approximately two years. Note that in the formula n_A is a key factor such that as it increases, the number of plans will dramatically expand (Figure 8). For a large-scale TBW, such as "Battle Commander," whose units may amount to no less than a dozen or dozens, the search will be more difficult.

Table 1: Branching factors comparison between TBW game and other board games.

	Branching factor	Comment
Chess [15]	≈30	Maximum branching factor is 40.
Go [16]	≈100	
Amazons [17]	≈1500	There are 2176 branches in the first turn.
TBW	710 million~317 trillion	Suppose that the movement point is 5, the movement cost is 1, and the amount of units is 5 for each side.

3.2. Features and Comparison. Compared with TBW games, other board games (such as chess, checkers, etc.) only require selecting a unit to perform an action in a single round, which not only results in fewer single-round action plans but also makes the number of plans linear with increasing numbers of units (for the chess type played by adding pieces, such as Go and Go-Moku, the number of plans is linear with increasing amounts of empty grids on the board). The number of single-round action plans corresponds to the size of the game tree branching factor. Table 1 shows a comparison between TBW games and some other ordinary board games that have more branching factors. A large branching factor and a rapidly expanding number of units are the key features by which the TWB games are distinguished from other board games.

A TBW game is essentially the compound of combinational optimization laterally and game tree search vertically

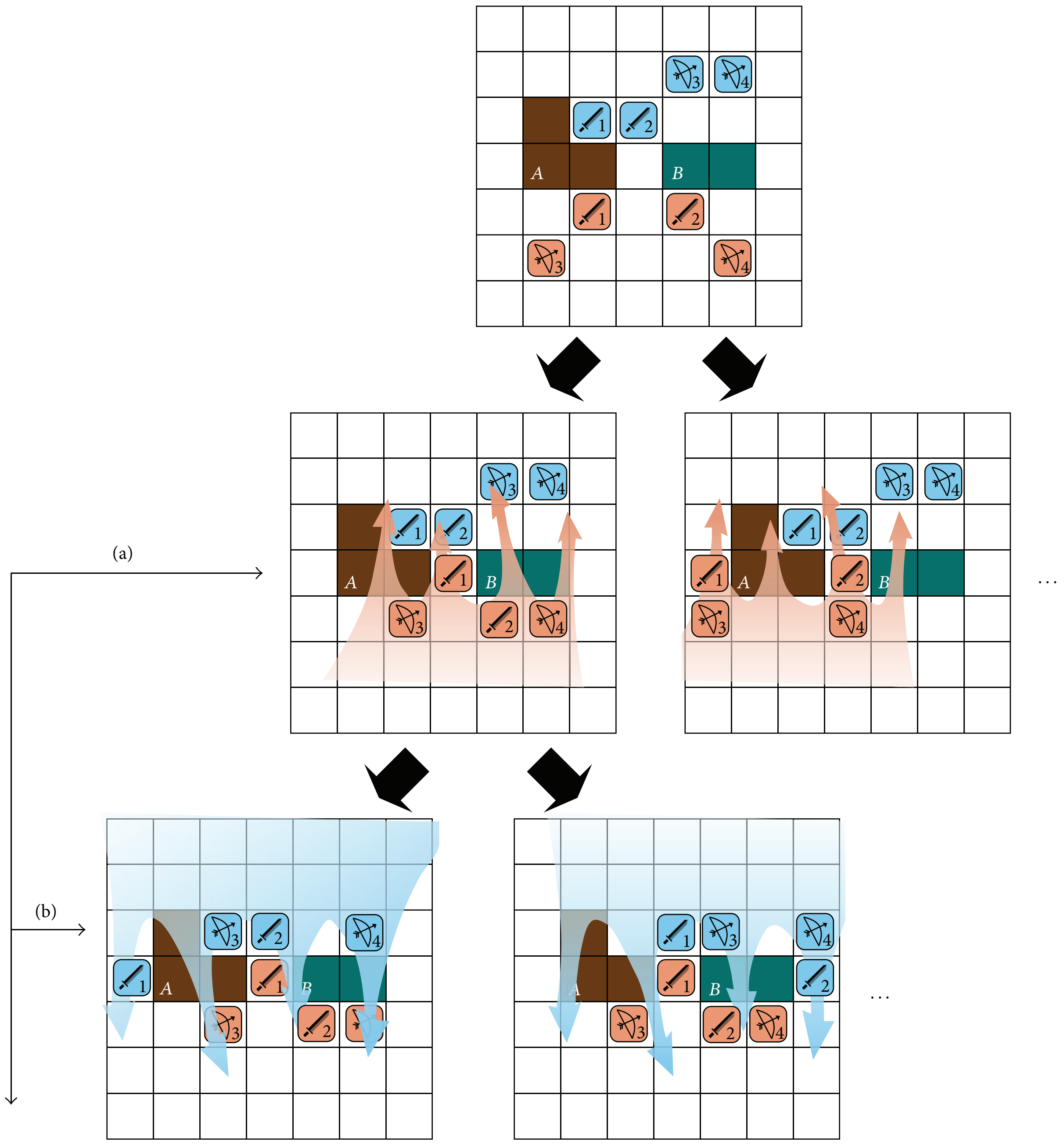


Figure 9: Search tree of TBW game: (a) red side's turn; (b) blue side's turn.

(Figure 9). Vertically, it can be seen as a tree search problem with a huge branching factor. Laterally, the relationship between layers is a series of phased combination optimizations, which is like a programming problem of multiagent collaboration. Therefore, the new game model generated by the expansion of the explosive branches needs to be researched by new algorithms.

Because the large number of states in a single round is the key problem by which the TBW games are distinguished from other board games, the optimization search and pruning of a single round have become the most important issues and processes for solving TBW games. That the search of a single round can be efficiently completed guarantees that the entire game tree can be extended. In the following, we propose two single-round search algorithms and compare them.

4. Single-Round Search Algorithms

4.1. Algorithm 2: Dictionary Sequence Enumeration Algorithm. Each side of a TBW game (hereafter, unless otherwise stated, referring specifically to Alex's side) wants to achieve a single turn search. Based on Section 3.1, we need to first determine the sequence of actions of n_A units and then enumerate all of the action plans of the units in each sequence.

```
Input: the original permutation sequence
  $Ord = Ord_1 Ord_2 \cdots Ord_{j-1} Ord_j Ord_{j+1} \cdots Ord_{k-1} Ord_k Ord_{k+1} \cdots Ord_n$
(1) find $j = \max\{i \mid Ord_i < Ord_{i+1}\}$
(2) if $j$ doesn't exist then
(3)     exit, and next permutation sequence doesn't exist.
(4) else
(5)     find $k = \max\{i \mid Ord_i > Ord_j\}$
(6)     swap($Ord_j, Ord_k$)
(7)     reverse the sub-sequence $Ord_{j+1} \cdots Ord_{k-1} Ord_j Ord_{k+1} \cdots Ord_n$
(8)     Output: $Ord' = Ord_1 Ord_2 \cdots Ord_{j-1} Ord_k Ord_n \cdots Ord_{k+1} Ord_j Ord_{k-1} \cdots Ord_{j+1}$ is the
        next permutation sequence.
(9) end if
```

Algorithm 1: *next_permutation*(*Ord*).

```
(1) initialize a sequence Ord which is the first sequence in dictionary sequences
(2) while Ord exists do
(3)     call Search(1)
(4)     $Ord \leftarrow$ next_permutation(Ord)
(5) end while
```

Algorithm 2: Dictionary sequence enumeration algorithm.

4.1.1. Action Sequence of Units Algorithm. Determining an action sequence of n_A units requires a permutation algorithm. There are some famous permutation algorithms, such as the recursive method based on exchange, the orthoposition trade method, the descending carry method, and the dictionary sequence method [22–25]. Their execution strategies are different, their time and space complexities vary, and they have been used in different problems. We first apply the dictionary sequence method, whose time complexity is lower. The idea of all permutation generation from n elements (e.g., $\{1, 2, \ldots, n\}$) is that with the beginning of the first sequence $(123 \cdots n)$ a series of subsequent larger sequences are generated lexicographically until reaching the reverse order $(n \cdots 321)$. The algorithm, called *next_permutation*, which generates the next sequence from an original one, is illustrated as in Algorithm 1.

For example, 754938621 is a sequence of numbers 1–9. The next sequence obtained by this algorithm is 754961238.

4.1.2. Algorithm 2: Dictionary Sequence Enumeration Algorithm. Enumerate all of the plans of units' actions in a particular order. Because the search depth is limited (equal to the number of units), depth-first search is an effective method. Because the depth is not great, realizing the depth-first search by the use of recursion requires smaller space overhead, which leads to the sequential enumeration algorithm with permutation and recursion, as in Algorithm 2.

Here *Search*(i) is the algorithm for enumerating all of the action plans of the ith unit (see Algorithm 3).

4.2. Algorithm 5: Recursive Enumeration Algorithm. Algorithm 2 comes from a simple idea that always starts enumeration from the first unit in every search for the next sequence. However, compared with the previous sequence, the front parts of units whose orders are not changed are not required to be enumerated again, which creates redundant computing and reduces efficiency. For example, when the search of sequence $Ord_1, Ord_2, \ldots, Ord_i, \ldots, Ord_j, \ldots, Ord_n$ is finished, if the next sequential order is adjusted only from the ith to the jth unit, then in the recursive enumeration phases the units from the first one to the $i - 1$th can directly inherit the enumeration results of the previous sequence and we only need to enumerate the units from the ith to the last one recursively. On the basis of this feature, we switch to the recursive permutation algorithm to achieve the arrangement so that the recursive algorithm combines with the recursive depth-first search algorithm for the purpose of removing the redundant computation, which is the improved algorithm called the *recursive enumeration algorithm* illustrated as in Algorithm 4.

In Algorithm 4, n is the size of our sequence (lines (1), (6)). With respect to the predefined procedure, we generate the permutations from the ith to the last unit in the sequence by calling the function *recursive_permutation*(i). The latter is realized using the subpermutations from the $i + 1$th to the last unit in the sequence, which are generated by calling the function *recursive_permutation*($i + 1$) recursively (lines (5)–(11)). The index j points to the unit swapped with the ith unit (line (7)) in every recursive call, after which the two units must resume their orders (line (9)), for the next step.

By initializing the sequence *Ord* and running the function *recursive_permutation*(1), we can obtain a full permutation of all the elements.

Based on the above, the improved single-round search algorithm, called the recursive enumeration algorithm, is described as in Algorithm 5.

```
(1) if $i > n$ then
(2)     return
(3) else
(4)     for each action plan of the $i$th unit
(5)         execute the current plan
(6)         call $Search(i + 1)$
(7)         cancel this plan and rollback to the previous state
(8)     end for
(9) end if
```

Algorithm 3: *Search*(i).

```
(1) if $i \geq n$ then
(2)     output the generated sequence $Ord_1, Ord_2, \ldots, Ord_n$
(3)     return
(4) else
(5)     $j \leftarrow i$
(6)     while $j \leq n$ do
(7)         swap($Ord_i$, $Ord_j$)
(8)         call $recursive\_permutation(i + 1)$
(9)         swap($Ord_i$, $Ord_j$)
(10)        $j \leftarrow j + 1$
(11)    end while
(12) end if
```

Algorithm 4: *recursive_permutation*(i) (enumerate sequences from ith to the last element).

The framework of this new algorithm is similar to that of the *recursive_permutation* algorithm, where n is the number of units. In the new algorithm, all the action plans of the ith unit, which involve selecting targets for attack, are enumerated and executed separately (lines (7)-(8)) after the required swap process. Then, after solving the subproblem using the recursive call *Plans_Search*(i+1), a rollback of the current plan is necessary and the state needs to be resumed (line (10)).

To enumerate the actions plans of all the units, the sequence Ord is initialized, and then the function *Plans_Search*(1) runs.

From step (3) of Algorithm 5, before enumerating the action plans of the unit, we do not need to generate all of the sequences; that is, for each unit, determination of its order and enumeration of its actions are carried out simultaneously.

4.3. Comparison. First, we compare the time complexities of the two algorithms.

The time consumption of the recursive enumeration algorithm lies in an n times loop and an $n-1$ times recursion, such that the time complexity is $O(n(n-1)(n-2)\cdots 1) = O(n!)$ [23]. It is the same as the time complexity of the dictionary sequence enumeration algorithm [23]. Moreover, the states searched by the two algorithms are also the same.

Theorem 1. *The states searched by Algorithms 2 and 5 are the same.*

Proof. Suppose $S(Ord)$ is the set of the states in the sequence Ord, and Pre_a are the sequences beginning with a in Q^A. According to Algorithm 2, it first determines the order of a sequence Ord and then enumerates all of the states S_1 under this sequence:

$$S_1 = \bigcup_{Ord^A \in Q^A} S\left(Ord^A\right). \tag{5}$$

According to the outermost layer of the recursion in Algorithm 5, we can obtain all of the states S_2:

$$S_2 = \bigcup_{a \in A} S\left(\mathrm{Pre}_a\right). \tag{6}$$

Because $\cup_{a \in A} \mathrm{Pre}_a = Q^A$ and $\cup_{Ord^A \in Q^A} S(Ord^A) = S(Q^A)$, therefore, $S_1 = S_2$. □

The difference between Algorithms 2 and 5 reflects the efficiency of their enumerations. In the searching process, an important atomic operation (ops1) expands each unit's action plan on each position it moves to. This is because (1) the states taken by search are mainly composed of every unit moving to every position and (2) every unit arriving at every position and then attacking or choosing other options for action is a time-consuming operation in the searching process. Suppose the number of ops1 in Algorithms 2 and 5 is H_1 and H_2, respectively. For simplicity, we make the following assumptions.

Assumption 2. Assume that every unit's movement range does not overlap another's, the sizes of which are all equal; that

```
(1) if $i > n$ then
(2)   return
(3) else
(4)   $j \leftarrow i$
(5)   while $j \le n$ do
(6)     swap($Ord_i$, $Ord_j$)
(7)     for each action plan of the $i$th unit
(8)       execute the current plan
(9)       call $Plans\_Search(i+1)$
(10)      cancel this plan and rollback to the previous state
(11)    end for
(12)    swap($Ord_i$, $Ord_j$)
(13)    $j \leftarrow j+1$
(14)  end while
(15) end if
```

Algorithm 5: Recursive enumeration algorithm: *Plans_Search*(i) (search action plans from ith to the last unit).

is, $|R_1| = |R_2| = \cdots = |R_n| = r$, and $R_1 \cap R_2 \cap \cdots \cap R_n = \varnothing$. Moreover, every unit cannot attack after moving (i.e., none of the enemies are inside the attack range).

In the following, we calculate H_1 and H_2, respectively.

In Algorithm 2, in each identified sequence, ops1 corresponds to the nodes of the search tree formed by enumerating states (except the root node, which represents no action). The depth of the tree is n, and each of the branching factors is r; then, the number of nodes is $r^n + r^{n-1} + \cdots + r$. Moreover, the number of all sequences is $P_n^n = n!$ and therefore

$$H_1 = n!\left(r^n + r^{n-1} + \cdots + r\right). \tag{7}$$

In Algorithm 5, suppose that the number of ops1 of n units is h_n. The first unit performing an action according to the order of the current sequence is a. According to Algorithm 5, every time a moves to a tilt, it will make a new state combining the following $n-1$ units, such that the number of the ops1 is $1 + h_{n-1}$. Because the number of tilts a can move to is r and the recursion operates n times, we can deduce that $h_n = nr(1 + h_{n-1})$, where $h_1 = r$; thus,

$$H_2 = h_n = n!\sum_{i=0}^{n-1}\frac{r^{n-i}}{i!}. \tag{8}$$

Accordingly,

$$H_1 - H_2 = n!\sum_{i=2}^{n-1}\frac{i!-1}{i!}r^{n-i}. \tag{9}$$

It is easy to see that the number of ops1 of Algorithm 5 is smaller than that of Algorithm 2. Table 2 lists the experimental results, showing H_1 under Assumption 2, H_2 under a general condition, and their differences.

Conclusion. On the premise that the search states are exactly the same, Algorithm 5 is better than Algorithm 2 regarding the consumption of ops1 and actual running time.

5. Experimental Evaluation

In this section, we present our experimental evaluation of the performance of Algorithms 2 and 5 under all types of conditions and their comparison. Because they are both single-round search algorithms, we set only one side's units on the board, ignoring the other side's, whose interference is equivalent to narrowing the range of units' movement. Experiments are grouped based on the following conditions: the number of units, the unit's movement point, and the dispersion of units. The number of units is set to 3 and 4 (setting to 2 is too simple with a lack of universality, while setting to 5 leads to timeout). The movement point is set to 2, 3, and 4, and the movement cost of each tilt is set to 1. The dispersion is set to the most dispersive ones and the most centralized ones. The most dispersive cases mean that the movement ranges of all of the units are independent without overlapping each other, corresponding to the worst case in Section 3.1. The most centralized cases mean that all of the units are put together (Figure 10), which maximizes the overlap degree and corresponds to the best case in Section 3.1. The experimental groups set above cover all of the actual situations. The board used in the experiments is completely open without any boundary and barrier. The case of a board with boundaries and barriers can be classified into cases where a smaller movement point of units is set. The experimental tool is a PC with Intel Core i7-2600@2.40 GHz CPU and 4.00 GB memory, and the program was written with Visual C++ 2005 with optimized running time.

From Table 2, we can see that in all cases the number of ops1 of Algorithm 5 is less than that of Algorithm 2 for different levels. Assuming that the number of units is invariable, the optimization level of Algorithm 5 will become low by increasing the movement point, which can be deduced from (8) and (9): under Assumption 2, $\widehat{D}_{\text{ops1}}$ which shows the reduced percentage of using ops1 in Algorithm 5 instead of in Algorithm 2 is

$$\widehat{D}_{\text{ops1}} = \frac{H_1 - H_2}{H_1} = \frac{\sum_{i=2}^{n-1}\left(\left(i!-1\right)/i!\right)r^{n-i}}{\sum_{i=0}^{n-1}r^{n-i}}. \tag{10}$$

Table 2: (a) For three units, the comparison of ops1 under different movement points and different dispersions. (b) For four units, the comparison of ops1 under different movement points and different dispersions.

(a)

	Movement point 2		Movement point 3		Movement point 4	
	Dispersed	Compact	Dispersed	Compact	Dispersed	Compact
Algorithm 2	14274	9804	97650	79260	423858	371652
Algorithm 5	14235	9771	97575	79191	423735	371535
$D_{\text{ops1}}{}^{*}$	0.27%	0.34%	0.077%	0.087%	0.029%	0.031%
$\widehat{D}_{\text{ops1}}{}^{**}$	0.30%		0.080%		0.030%	

(b)

	Movement point 2		Movement point 3		Movement point 4	
	Dispersed	Compact	Dispersed	Compact	Dispersed	Compact
Algorithm 2	742560	345264	9765600	6394800	69513696	53301744
Algorithm 5	740272	343804	9757600	6388468	69492704	53283548
$D_{\text{ops1}}{}^{*}$	0.31%	0.42%	0.082%	0.099%	0.030%	0.034%
$\widehat{D}_{\text{ops1}}{}^{**}$	0.30%		0.080%		0.030%	

$^{*}D_{\text{ops1}}$ shows the reduced percentage of using ops1 in Algorithm 5 instead of in Algorithm 2.

$^{**}\widehat{D}_{\text{ops1}}$ shows the estimated value of D_{ops1} under Assumption 2.

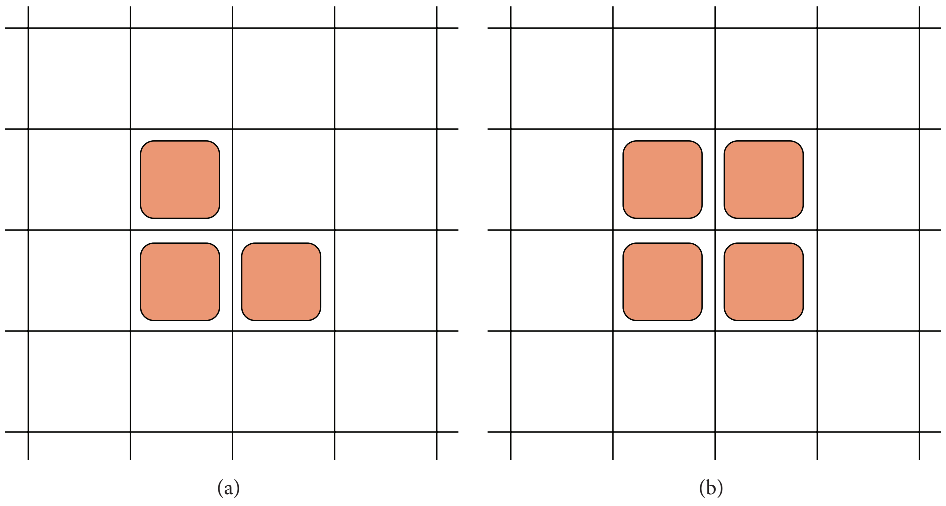


Figure 10: (a) Compact placement of three units. (b) Compact placement of four units.

In (10), the numerator is the infinitesimal of higher order of the denominator; that is,

$$\widehat{D}_{\text{ops1}} \approx \frac{1}{2} r^{-2}. \tag{11}$$

Table 2 lists the values of $\widehat{D}_{\text{ops1}}$ when the movement point is 2, 3, and 4, which are consistent with the experimental results.

Under the same conditions of the movement point and the number of units, the value of D_{ops1} with compact units is more than that with dispersed units. This is because the more the units are compact, the stronger the interference the units will cause to each other, which is equivalent to a narrow movement range of r. According to (11), therefore, D_{ops1} will increase correspondingly. Moreover, under the same conditions of the movement point and the degree of units' dispersion, D_{ops1} will also increase with the increase of the number of units. In summary, the experiments show that, regardless of whether Assumption 2 is satisfied, Algorithm 5 always performs better than Algorithm 2 on the number of ops1, which coincides with $\widehat{D}_{\text{ops1}}$ from (11). Because the degree of optimization is not very prominent, the running times of these two algorithms are almost the same.

6. Conclusions

Based on a modest study of turn-based war chess games (TBW), a common gaming model and its formal description are first proposed. By comparison with other chess type models, the most important feature of TBW has been discussed: the player needs to complete actions for all of his units in a turn, which leads to a huge branching factor. Then, a game tree theory framework to solve this model is proposed. Finally, two algorithms for single-round search from the most complex part of the framework are proposed: Algorithm 2 is the dictionary sequence enumeration algorithm and Algorithm 5 is the recursive enumeration algorithm. Finally, based on theoretical derivations and experimental results, respectively, the completeness of these algorithms is proven. Also, the performance comparison shows that under all conditions the number of ops1 of Algorithm 5 decreases to a certain extent compared to that of Algorithm 2.

Although these two algorithms are designed from classical algorithms, they can be used to solve the single-round search problem completely and effectively. Moreover, the research angles of the two algorithms are completely different, which provide two specific frameworks for a further study on TBW.

(1) The dictionary sequence enumeration algorithm is implemented in two steps. The first step consists of the generation of sequences; and the second step consists of the enumeration of action plans under these sequences. Therefore, this algorithm is based on sequences. Different permutation algorithms can be used to generate different orders of sequences, which may be more suitable for new demands. For instance, the orthoposition trade method [23] can minimize the difference of each pair of adjacent sequences. Thus, more action plans from the former sequence can be reused for the next, which can improve the efficiency.

(2) The recursive enumeration algorithm is also implemented in two steps. The first step consists of the enumeration of action plans of the current unit; and the second step consists of the generation of the sequences of the next units. Therefore, this algorithm is based on action plans. Pruning bad action plans in the depth-first search process can easily cut off all the following action sequences and action plans of later units, which will lead to a significant improvement of efficiency.

In the current era of digital entertainment, TBW games have broad application prospects. They also have a profound theoretical research value. However, in this study, TBW theory has been discussed partially. The game model framework we proposed is composed of the combinatorial optimization problem on one hand, and the game tree search problem on the other hand. Thus, our future research will mainly start with the following two points:

(1) Introduce the multiagent collaborative planning approach to efficiently prune the huge branches of the game tree. Moreover, by introducing the independent detection approach [26], we can separate the independent units that have no effect on each other into different groups with the purpose of decreasing the number of units in each group.

(2) Introduce the Monte Carlo Tree Search method to simulate the deep nodes. The single-round search algorithms proposed in this paper are complete algorithms and can be used to verify the performance of the new algorithm.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by the National Key Basic Research Program of China (973 Program 2013CB329103 of 2013CB329100), the National Natural Science Foundations of China (NSFC-61173129, 61472053, and 91420102), the Fundamental Research Funds for the Central University (CDJXS11182240 and CDJXS11180004), and the Innovation Team Construction Project of Chongqing Institute of Engineering (2014xcxtd05).

References

[1] L. V. Alus, H. J. van den Herik, and M. P. H. Huntjens, “Go-Moku solved by new search techniques,” *Computational Intelligence*, vol. 12, no. 1, pp. 7–23, 1996.

[2] C. Deng, *Research on search algorithms in computer go [M.S. thesis]*, Kunming University of Science and Technology, 2013.

[3] H. C. Neto, R. M. S. Julia, G. S. Caexeta, and A. R. A. Barcelos, “LS-VisionDraughts: improving the performance of an agent for checkers by integrating computational intelligence, reinforcement learning and a powerful search method,” *Applied Intelligence*, vol. 41, no. 2, pp. 525–550, 2014.

[4] Y. J. Liu, *The research and implementation of computer games which based on the alpha-beta algorithm [M.S. thesis]*, Dalian Jiaotong University, 2012.

[5] J. Jokić, *Izrada šahovskog engine-a [Ph.D. thesis]*, University of Rijeka, 2014.

[6] N. R. Sturtevant and R. E. Korf, “On pruning techniques for multi-player games,” in *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence (AAAI-IAAI ’00)*, vol. 49, pp. 201–207, Austin, Tex, USA, July-August 2000.

[7] C. Luckhart and K. Irani B, “An algorithmic solution of N-person games,” in *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI ’86)*, vol. 1, pp. 158–162, Philadelphia, Pa, USA, August 1986.

[8] C. Browne, “A problem case for UCT,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 1, pp. 69–74, 2013.

[9] M. H. M. Winands, Y. Björnsson, and J.-T. Saito, “Monte Carlo tree search in lines of action,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 239–250, 2010.

[10] P. Nijssen and M. H. M. Winands, “Monte carlo tree search for the hide-and-seek game Scotland Yard,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 282–294, 2012.

[11] G. Wang, H. Liu, and N. Zhu, “A survey of war games technology,” *Ordnance Industry Automation*, vol. 31, no. 8, pp. 38–41, 2012.

[12] M. H. Bergsma and P. Spronck, “Adaptive spatial reasoning for turn-based strategy games,” in *Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE ’08)*, pp. 161–166, Palo Alto, Calif, USA, October 2008.

[13] S. Paskaradevan and J. Denzinger, “A hybrid cooperative behavior learning method for a rule-based shout-ahead architecture,” in *Proceedings of the IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT ’12)*, vol. 2, pp. 266–273, IEEE Computer Society, December 2012.

[14] S. Wiens, J. Denzinger, and S. Paskaradevan, “Creating large numbers of game AIs by learning behavior for cooperating units,” in *Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG ’13)*, pp. 1–8, IEEE, Ontario, Canada, August 2013.

[15] G. Badhrinathan, A. Agarwal, and R. Anand Kumar, “Implementation of distributed chess engine using PaaS,” in *Proceedings of the International Conference on Cloud Computing Technologies, Applications and Management (ICCCTAM ’12)*, pp. 38–42, IEEE, Dubai, United Arab Emirates, December 2012.

[16] X. Xu and C. Xu, “Summarization of fundamental and methodology of computer games,” *Progress of Artificial Intelligence in China*, pp. 748–759, 2009.

[17] J. Song and M. Muller, “An enhanced solver for the game of Amazons,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 1, pp. 16–27, 2015.

[18] D. Gesang, Y. Wang, and X. Guo, “AI arithmetic design and practical path of a war-chess game,” *Journal of Tibet University (Natural Science Edition)*, vol. 26, no. 2, pp. 102–106, 2011.

[19] X. Xu and J. Wang, “Key technologies analysis of Chinese chess computer game,” *Mini-Micro Systems*, vol. 27, no. 6, pp. 961–969, 2006.

[20] B. Bošanský, V. Lisý, J. Čermák, R. Vítek, and M. Pěchouček, “Using double-oracle method and serialized alpha-beta search for pruning in simultaneous move games,” in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pp. 48–54, AAAI Press, Beijing, China, August 2013.

[21] A. Guez, D. Silver, and P. Dayan, “Scalable and efficient Bayes-adaptive reinforcement learning based on Monte-Carlo tree search,” *Journal of Artificial Intelligence Research*, vol. 48, pp. 841–883, 2013.

[22] R. Sedgewick, “Permutation generation methods,” *ACM Computing Surveys*, vol. 9, no. 2, pp. 137–164, 1977.

[23] D. E. Knuth, *The Art of Computer Programming Vol. 4A: Combinatorial Algorithms, Part 1*, The People’s Posts and Telecommunications Press, Beijing, China, 2012.

[24] B. R. Heap, “Permutations by interchanges,” *The Computer Journal*, vol. 6, no. 3, pp. 293–298, 1963.

[25] F. M. Ives, “Permutation enumeration: four new permutation algorithms,” *Communications of the ACM*, vol. 19, no. 2, pp. 68–72, 1976.

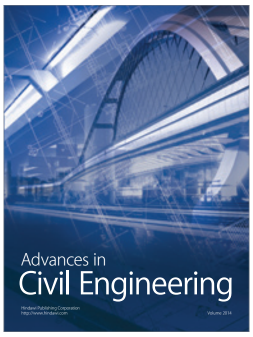
[26] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, “The increasing cost tree search for optimal multi-agent pathfinding,” *Artificial Intelligence*, vol. 195, pp. 470–495, 2013.

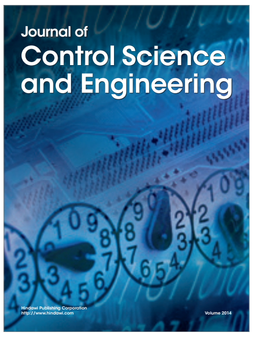
Journal of Engineering

The Scientific World Journal

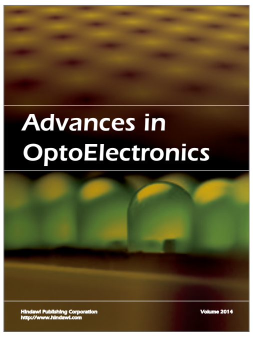
International Journal of Rotating Machinery

Journal of Sensors

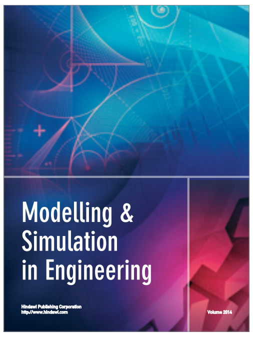
International Journal of Distributed Sensor Networks

Advances in Civil Engineering

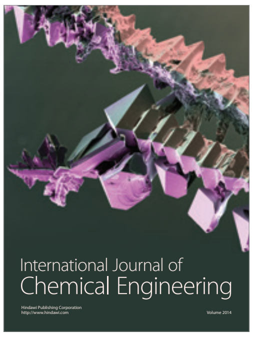
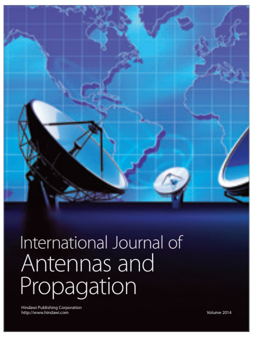

Journal of Control Science and Engineering

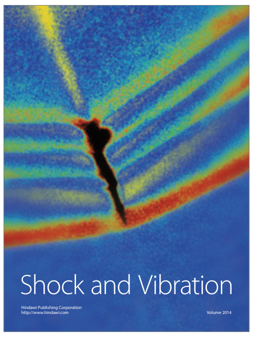
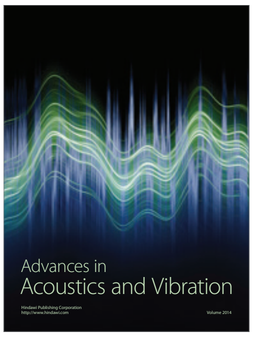
Journal of Robotics

Journal of Electrical and Computer Engineering

Advances in OptoElectronics

VLSI Design

International Journal of Navigation and Observation

Modelling & Simulation in Engineering

International Journal of Aerospace Engineering

International Journal of Chemical Engineering

International Journal of Antennas and Propagation

Active and Passive Electronic Components

Shock and Vibration

Advances in Acoustics and Vibration