Combinatorial Optimization Design of Search Tree Model Based on Hash Storage

Yun Liu, Jiajun Li*, Jingjing Chen
Basic School of Computer and Artificial Intelligence, Chaohu University, Chaohu, 238024, China

***Abstract*—The game search tree model usually does not consider the state information of similar nodes, which results in searching a huge state space, and there are problems such as the size of the game tree and the long solution time. In view of this, the article proposes a scheme using the idea of combinatorial optimization algorithm, which has an important application in solving the decision problem in the tree graph model. First, the special graph-theoretic structure of the point-grid game is analyzed, and the storage and search of states are optimized by designing hash functions; then, the branch delimitation algorithm is used to search the state space, and the evaluation value of repeated nodes is calculated by dynamic programming; finally, the state space is greatly reduced by combining the two-way detection search strategy. The results show that the algorithm improves decision-making efficiency and has achieved 37% and 42% final winning rate, respectively. The design provides new ideas for computational complexity problems in the field of game search and also proposes new solutions for the field of combinatorial optimization.**

***Keywords*—Combination optimization; game search algorithm; state space; transposition table**

I. Introduction

Game search algorithms have gained significant attention in recent years for their applications in decision-making, optimization, and artificial intelligence across diverse domains. However, traditional algorithms such as Minimax and pruning algorithms have some limitations that hinder their effectiveness in solving complex games[1]. The Minimax algorithm, proposed by von Neumann, aims at solving two-player games by constructing a game tree that minimizes the maximum outcome. Despite exploring all possible states, it leads to a vast state space. Pruning algorithms, like alpha-beta pruning by Yang and Feinberg, aim to reduce the search space but struggle with revisiting previously explored states[2].

Alternative approaches have been proposed, such as the PVS search algorithm by Kaufman and the branch-and-bound algorithm by Tucker[3]. However, existing algorithms often overlook superior decisions and yield suboptimal solutions[4]. This paper presents a novel combinatorial optimization algorithm based on branch delimitation to address large state spaces in point-grid chess game graph theory problems. It analyzes the graph structure, optimizes state storage with hash functions[5], employs the branch delimitation algorithm for efficient exploration, calculates evaluation values using dynamic programming, and implements a two-way detection search strategy to reduce the state space[6-8].

Experimental results demonstrate substantial improvements, with a 37% increase in the decision efficiency and a 42% higher final win rate. This paper contributes innovative ideas to address the computational complexity in a game search and provides new solutions for combinatorial optimization. Overall, this study advances understanding and application of game search algorithms, specifically for addressing large state spaces in point-grid chess game graph theory problems[9]. The proposed combinatorial optimization algorithm offers superior performance and has the potential to overcome limitations in traditional approaches[10], making it a valuable contribution to the field.

II. Basics of Dots and Boxes

A. Introduction to the Game

Dots and boxes are popular intellectual game due to its simplicity, ease of learning, entertainment value, and puzzle-solving nature. Unlike other board games such as Gomoku, dots and boxes has a unique set of rules. In this game, a legal move involves drawing a line between two dots on the board. Players take turns placing their pieces on the board until all four edges of a grid cell have been claimed[11]. Once a player captures a grid cell, they get an extra turn. In a 6x6 dots and boxes game, the mathematical formula for calculating the number of captured grid cells can be expressed as formula (1).

$$C(p) = \{q \in V \mid X_q = 0, (p,q) \in E\} \quad (1)$$

Here, *C(p)* represents the set of captured cells, V represents the set of points, E represents the set of edges, and *(p, q)* represents the edge between vertices p and q on the game board. X_q =0 means that there is no chess piece on point q. The winner is determined by the number of cells captured by the players when neither side can make any further moves[12].

B. Game Abstraction forms and Theorems

The game of dots and boxes has a special data structure in machine game competitions, where some game states often determine the outcome of the game[13]. This is because the formation of some game states can lead to significant changes in the next game situation, and one player can capture a large number of boxes through these game states, thereby increasing their chances of winning[14].

Theorem 1. Designing checkerboard storage based on move rules.

$$M(p,q) = \begin{cases} short\ move, & N(p) < 3 \\ long\ move, & N(p) \geq 3\ and\ N(q) \geq 3 \\ invalid\ move, & otherwise \end{cases} \quad (2)$$

Formula (2) defines the types of moves from point p to point q in dots and boxes games. When the number of empty points around point p is less than 3, the move is called a short step. When the number of empty points around point p is greater than or equal to 3 and the number of empty points around point q is also greater than or equal to 3, the move is called a long step[15]. Other moves are considered invalid moves.

Theorem 2. The long chain theorem predicts sure-win strategies.

$$R(C) = min_{v \in C}\, max_{u \in N(v) \cap C}\, L(C - v) - L(C - u) \quad (3)$$

Such as formula (3). Let *R(C)* denote the maximum profit that can be obtained by playing chain C in the game[16], and let *N(v)* denote the set of points adjacent to the point v. Suppose that there are two players, A and B, in G. Let *S(G)* be the set of winning strategies for A in G, and let *T(G)* be the set of winning strategies for B in G. Then the long chain theorem can be stated as following formula (4).

$$C \in S(G) \Leftrightarrow R(C) > 0 \quad (4)$$

If the maximum profit *R(C)* of a long chain C is greater than 0, then A has a winning strategy, otherwise B has a winning strategy.

Theorem 3. Stumping theorem predicts the likelihood of winning.

$$C(S, p) = 1 - |V(S, p)|/|P(S, p)| \quad (5)$$

Such as formula (5). Let S denote the current state of the board, and let p denote the next player to move (0 represents the first player, and 1 represents the second player). Let *V(S, p)* be the set of all possible board states in which the next player to move can win[17], and let *P(S, p)* be the set of all possible board states in which the next player to move can make a move.

Theorem 4. Calculated returns for hybrid strategies in gaming.

Let player A have a mixed strategy $\{x^*_1, x^*_2, \ldots, x^*_m\} \in X_A$, and let player B have a mixed strategy $\{y^*_1, y^*_2, \ldots, y^*_n\} \in Y_B$, such as formula (6).

$$E(x^*, y^*) = max\, E(x, y^*) = min\, E(x^*, y) \quad (x \in X, \in Y) \quad (6)$$

That is formula (7) and (8).

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_i^* y_j^* = max \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_i y_j^* \quad (7)$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_i^* y_j^* = min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_i^* y_j \quad (8)$$

In the game, player A and player B each have their own space of mixed strategies, X_A and X_B, respectively. $E(x^*, y^*)$represents the payoff when using mixed strategy (x^*, y^*). C is a matrix in which c_{ij} represents the payoff of two players under certain circumstances.

Theorems 5. Hashing for board representation.

Make the board state be represented as a vector $S=[s_1, s_2, \ldots, s_n]$ of length n, where s_i represents the state of the I – th position, such as "black piece," "white piece," "empty," and so on[18]. Next, define a binary vector $B=[b_1, b_2, \ldots, b_n]$ of length n, where the value of b_i is a randomly generated 0 or 1. Then, the new vector $X=[x_1, x_2, \ldots, x_n]$ is obtained by performing a bitwise *XOR* operation between the *S* vector and the *B* vector, i.e., $x_i = s_i \; XOR \; b_i$. Finally, each element of the *X* vector is treated as an 8-bit unsigned integer, and the hash value is calculated according to the following formula (9).

$$h(S) = \left(\left((x_1 * 2^8 + x_2) * \ldots * 2^8 + x_{n-1} \right) * 2^8 + x_n \right) \quad (9)$$

Theorem 6. Generation of moves for the second player.

Given the current board state *S*, the second player generates a move M_{op} based on the position of the opponent's pieces and the rules, using the formula (10).

$$M_{op} = move_{op}(S, p) | p \in P_{op} \quad (10)$$

where $move_{op}(S, p)$ represents all possible moves for the opponent's piece p in the state *S*, and P_{op} represents the set of opponent's pieces[19].

Theorem 7. Calculation of the Depth of the Game Tree.

Assume that there are n feasible successor states at the current game state, and each successor state has m feasible successor states, and so on, until a game-ending state is reached[20]. Then, the depth of the game tree can be calculated using the following formula (11).

$$Depth = log_m\, n \quad (11)$$

Theorem 8. Updating the branch delimitation benefit interval.

For a given position S, the best move obtained from the search starting from it is M, and the corresponding next position is S^*. According to the definition of branch delimitation algorithm, such as formula (12) (13).

$$\alpha = max_{1 \le i \le n} \{\alpha, f(S_i)\} \quad (12)$$

$$\beta = min\{\beta, f(S^*)\} \quad (13)$$

Theorem 9. Hash function clusters handle hash conflicts.

Select a sufficiently large prime number p so that each possible keyword falls within the range of 0 to p-1. Such as formula (14).

$$Z_p = \{0,1,2, \ldots, p-1\}, Z_p^* = \{1,2,3, \ldots, p-1\} \quad (14)$$

Now, for $a \in Z^*_p$ and $b \in Z_p$, define the hash function h_{ab}, which performs a linear transformation to reduce modulo m and modulo p, as follows formula (15).

$$h_{ab}(k) = \left((ak + b)\ mod\ p \right) mod\ m \quad (15)$$

Thus, we obtain a hash function family, such as formula (16):

$$h_{ab}(k) = \left((ak + b)\ mod\ p \right) mod\ m \quad (16)$$

Z_p and Z^*_p represent the sets of integers from 0 to p-1 and from 1 to p-1, respectively, and represent the range of possible keywords. h_{ab} is a hash function that maps the keyword k to the set of integers from 0 to m-1. H_{pm} is a hash function family[21], where each hash function is composed of the

defined hash function h_{ab} by linear transformation with modulo m and modulo p.

C. Branch Delimitation Algorithm Core Decision

This article focuses on strategic decision-making in the game theory and its relationship to winning odds. The article first proposes Theorem 2, using formulas (3) and (4) to calculate the maximum gain of each chain to analyze the gains of different chains in the game. Secondly[22], Theorem 3 can calculate the winning player through formula (5) and thus predict the final winner in a certain game state. Finally, the unique board of the dot game needs to judge the captured squares, which will cause one party to capture the squares in a row[23]. Theorem 4 is usually applied in the later stage of the game to decide whether to make a concession grid, and the player's payoff is calculated using equations (6), (7)and (8) through a mixed strategy[24]. For the lattice game, the algorithm recursively studies the game tree, quantizes the features into feature vectors, and uses dynamic programming to find the optimal weight until the final state of the game. The evaluation function of the branch and bound algorithm calculates the score for each state of the game[24].

The calculation is done as follows:

For edges of length less than 3 (i.e. short moves), such as formula (17).

$$f(x,y)=\begin{cases}0, & x=0\\ -1, & x=-1\\ -2, & x=-2\end{cases} \tag{17}$$

For edges of length equal to 3 or greater (i.e. chain, rings, or long moves), such as formula (18).

$$f(x,y)=\begin{cases}5-elength, & x>2\\ -elength-1, & x<2, y=1\\ -elength, & x<2, y=0\end{cases} \tag{18}$$

In formulas (17) and (18), *f(x, y)* represents the evaluation function, where x represents the number of edges in the current state, and y=1 denotes a chain formed by the edges while y=0 denotes a loop formed by the edges. The length represents the length of the chain or loop. This evaluation function uses a recursive approach and performs a depth-first search, attempting various possible movements in each state[25].

III. Design of the Game Tree Model

A. Algorithmic Chessboard Design

The traditional matrix representation and the bit operation representation for chessboard state in artificial intelligence are analyzed[26]. The time complexity of matrix representation is *O(n2)*, and the memory usage is high. The bitwise operation representation method requires a large amount of memory space[27], and due to the low efficiency of bitwise operation, it will lead to low searching efficiency. Both methods have disadvantages and limitations.

In this paper, we propose a hash function design based on Theorem 5 to map a chessboard state S to an unsigned 64-bit integer for representing the current game state. The hash function design employs a vector S of length n to represent the chessboard state and defines a binary vector B of length n to shuffle the arrangement order of the status of each position in S, increasing the randomness and collision resistance of the hash function[28]. Meanwhile, XOR operation is used to enable the hash function to process each chessboard state quickly while maintaining low computational complexity.

We define the data structure of the hash table as T[h(S)]=PHashNode, where each PHashNode stores key information about the current state, such as the evaluation value of the game position and the search depth[29]. We utilize the hash function to quickly identify duplicate nodes. When a new node is discovered, its hash value is stored in the hash table, and if it is a duplicate, it is skipped, reducing the number of searches.

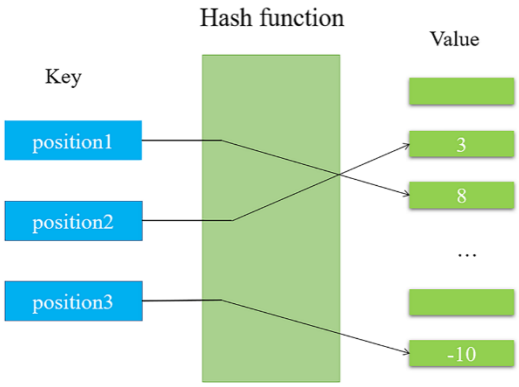

Fig. 1. Hash table dots and boxes board position mapping.

As the hash function design process shown in Fig. 1, position is the information of current board position; value indicates the evaluation value of the current board position.

B. Specific Design of the Generation Strategy

The paper explores traditional moving generation methods in dots and boxes, including enumeration, first-player, and second-player methods. The enumeration method generates all possible moves for each point on the board, resulting in a move set of size *O(nm)*. The first-player method moves pieces without considering whether the moves are legal, potentially generating many invalid moves[30]. The second-player method considers only the opponent's piece movements, leading to a move set that does not include invalid moves from one's own piece movements, but may increase program complexity[31].

To ensure that all feasible moves in the game tree can be expanded at the same level, the article uses a breadth-first strategy based on hash storage. This strategy uses a queue to store all successor states of the current game state, enumerates all successor states, adds unvisited states to the queue, and expands them. The depth of the game tree is calculated using formula (11) in Theorem 7, and the breadth-first strategy ensures that all feasible states are traversed, and all possible moves are generated.

A schematic diagram of the landing process generated by players A and B in the same situation is shown in Fig. 2.

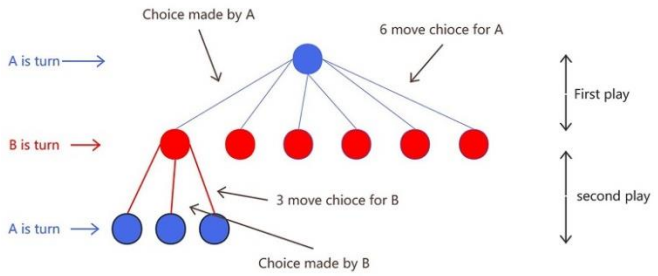

Fig. 2. Generated landings diagram.

Using equation (12) and (13) in Theorem 8 to calculate the dynamic gain interval of the branch-and-bound algorithm during the search. When $\beta \leq \alpha$, pruning can be performed. It is necessary to update $\beta \leq \alpha$ as soon as possible. When all moves and states of a board are generated and the available edges are sorted in ascending order, moves with larger evaluation values are searched first, which increases the number of updates to α and reduces the number of updates to β.

IV. Algorithm Evaluation and Optimization of Search Strategies

A. Transpose Table Storage Optimization

The search algorithm is the core part of decision-making systems, and the evaluation function is the "lighthouse" of the search algorithm. It determines the search direction of the search algorithm on the game tree. The number of nodes for the traditional Minimax search is formula (19).

$$N_d = 1 + b + b^2 + \cdots + b^d = b^d \frac{1-\frac{1}{b^{d+1}}}{1-\frac{1}{b}} = O(b^d) \quad (19)$$

The number of nodes for the branch delimitation search is formula (20).

$$N_d = b^{(d+1)/2} + b^{(d-1)/2} - 1 \quad (20)$$

Where N_d represents the number of nodes in a tree with a branching factor of b and a depth of d. When the number of child nodes at each node is same and the depth is large enough, the number of nodes for the branch delimitation search doubles with the search depth, such as formula (21).

$$N_{2d} = 2b^{\left(d+\frac{1}{2}\right)} \quad (21)$$

Therefore, when the depth of the branch delimitation algorithm search tree doubles, the increase in the number of nodes is relatively small, indicating that the branch delimitation algorithm is more suitable for searching in cases where the depth is large.

Transposition table is a data structure used to optimize search algorithms by storing the evaluation value and move of previously searched positions for direct use when encountering the same position in the future. First, a hash table is used to optimize the storage of states, and the chess board position is represented as a 32-bit integer using XOR operation, such as formula (22).

$$temp.v32 = position.v32[0]\hat{}position.v32[1] \quad (22)$$

The processing result is stored in i and used as an index in the hash table, such as formula (23).

$$i = temp.v16[0]\hat{}(temp.v16[1] << 4) \quad (23)$$

The paper presents an optimization to reduce the number of computations required to evaluate chess board positions. The proposed algorithm checks for the existence of a Hash Node object in the hash table for each access position, and returns the previously computed evaluation value to avoid redundant calculations. If there is no Hash Node object, a new one is created and stored in the hash table index, which contains information about the current position, alpha and beta values, and the computed evaluation value. This stored information can be reused in the next traversal. Fig. 3 illustrates the process of storing and reusing node information in the transposition table.

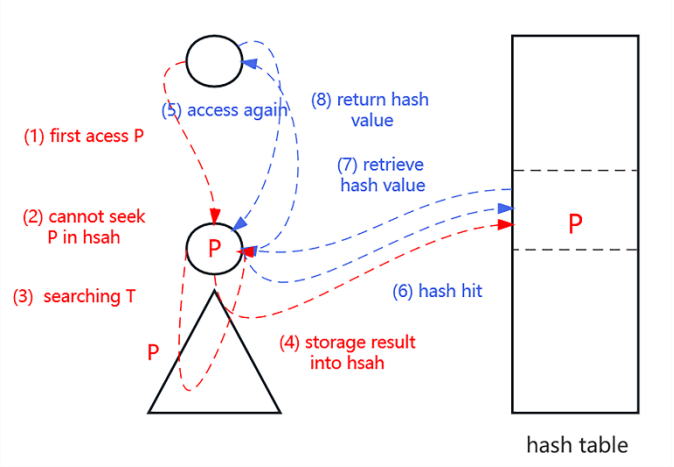

Fig. 3. Hash hit process diagram.

When the node p is visited for the first time and cannot be found in the hash table, a complete search will be performed, and the search result will be stored in column T of the transposition table. Then, the column will be stored in the hash table. When the node p is visited again and its hash value is found in the hash table, the stored result in the hash table can be directly returned. The branch delimitation algorithm incorporating the transpose table hashing can be represented by the algorithmic flow as follows:

Algorithm 1. pruning strategy algorithm

```
Input: board, depth, alpha, beta
Output: optimal evaluate value
value=SearchTT(HashKey, alpha, beta, depth);
If(value is valid)
  return value;
if(GameOver(board)||depth==0)
  value=Evaluate(board);
  if(depth==0)
   InsertHashTable(value, HashKey, depth, EXACT);
  return value;
best=-∞; ValueIsExact = 0;w = CreateSuccessors(board, p);
for(i=0; i<w; i++)
  HashKey=MakeMoveWithTT(board, pi);
  value=-AlphaBeta_TT(board, depth-1, -beta, -alpha);
  HashKey=RestoreMoveWithTT(board, pi);
```

```
14:  if(value>=beta)
15:   InsertHashTable(value, HashKey, depth, LOWERBOUND);
16:   return value;
17:  if(value>best)
18:   best=value;
19:   if(value>alpha)
20:    alpha = value; ValueIsExact = 1;
21:    InsertHashTable(value, HashKey, depth, EXACT);
22: if(ValueIsExact)
23:  InsertHashTable(value, HashKey, depth, EXACT);
24: InsertHashTable(value, HashKey, depth, LOWERBOUND);
25: return best;
```

This algorithm utilizes the SearchTT to avoid redundant evaluations of game states in the transposition table prior to AlphaBeta_TT pruning. If the game has ended or the search depth has reached 0, then the current game state is evaluated using the evaluation function and the game state information is inserted into the transposition table through Insert HashTable. Otherwise, all possible moves for the current game state are generated, and AlphaBeta_TT is recursively called for each move to perform pruning. The value returned from each recursive call is used to update the values of alpha, beta, and best. If a move is found that causes beta<=alpha, the function immediately returns and records the move in the transposition table.

B. Optimized Bidirectional Detection Search Method

Search algorithms commonly used in game systems are usually single-directional, searching from the initial state to the target in one direction in the game tree. The paper proposes a bidirectional search algorithm that divides a hash table into two parts: the front and the back. The search algorithm compares the node to be searched with the middle value of the hash table. If the item found during the search is smaller than the middle value, the search continues in the front part of the hash table. If it is larger, the search continues in the back part of the hash table. By using this approach, the algorithm can find the shortest path more quickly. The search process of this algorithm is illustrated in Fig. 4.

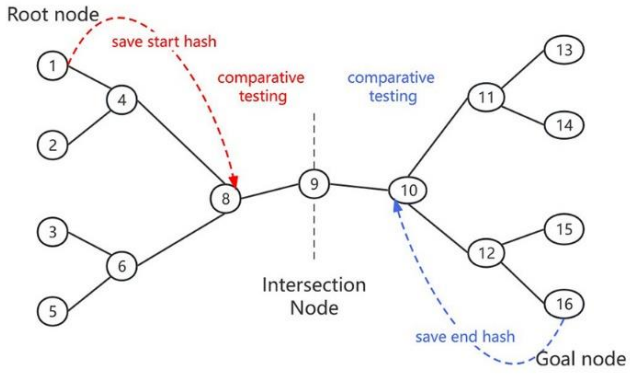

Fig. 4. Schematic diagram of the bidirectional search algorithm.

The paper proposes a two-way detection search algorithm that starts with two queues, one for forward search and the other for reverse search. The algorithm removes the first node from the queue, generates its children, and adds them to the corresponding queue. It continues until a common node is found in both queues, and then returns the shortest path. With a branching factor of b and a distance of d between the initial and target nodes, each queue will have bk nodes after k steps. If d is even, the two queues meet at a middle node, and the worst case requires expanding all nodes to the middle node in both queues.

Thus, the total number of nodes expanded by the algorithm can be represented as formula (24).

$$O\left(b^0 + b^1 + \cdots + b^{\left(\frac{d}{2}\right)}\right) = O\left(b^{\left(\frac{d}{2}\right)}\right) \quad (24)$$

Therefore, the time complexity of the bidirectional search algorithm is $O(b^{(d/2)})$, which is significantly lower than that of single-directional search algorithms.

The bidirectional hash table formula is used to store the game state, such as formula (25).

$$H(s) = h(s)\%M \quad (25)$$

where H(s) represents the hash value of state s, $h(s)$ represents the integer value obtained by hashing state s, and M represents the size of the hash table. Assuming the depth of the search tree is d and the time complexity of searching each layer is $O(b)$, the time complexity of the bidirectional search algorithm is $O(b^{(d/2)})$. The pseudocode of the bidirectional search algorithm is as follows:

Algorithm 2. bi-directional search algorithm

```
Input: begin, end, gF(begin), turnF, turnB, U, cost(n, c)
Output: U or ∞
1:  gF(begin):=gB(end):=0, turnF:={begin}, turnB:={end}, U:=∞
2:  while (turnF！=null and turnB!=null) do
3:   C:=min(prminF, prminB)
4:  if(C=prminF) then
5:   choose n ∈turnF for which prF(n)=prminF
6:   move n from turnF to ClosedF
7:  for each child c of n do
8:   if c ∈turnF ∪ ClosedF and gF(c)≤gF(n)+cost(n, c) then
9:    continue
10:  if c ∈ turnF ∪ ClosedF then
11:   remove c from turnF ∪ ClosedF
12:  gF(c):=gF(n)+cost(n, c)
13:  add c to TurnF
14:  if c ∈ turnF then
15:   U :=min(U, gF(c)+gB(c))
16:  else return ∞
```

The proposed algorithm employs two sets, turnF and turnB, to maintain unexplored nodes during the search process. Initially, turnF and turnB are initialized with the initial and target states, respectively. As the algorithm explores, it updates the cost of reaching each node and adds it to the appropriate set. Moreover, the algorithm keeps track of the minimum sum of costs to reach a node from the initial state and the target state, which is stored in variable U. The algorithm terminates when turnF and turnB both become empty or U is less than a certain threshold, ensuring the discovery of the shortest path between the initial and target states.

V. Experiments and Results

A. Search Depth Experiments and Analysis

This experiment compares the performance of Algorithm 1 and Algorithm 2 in optimizing the branch delimitation algorithm for the game of dots and boxes. The study uses a randomly generated 6*6 chess board, with 100 independent experiments conducted. The experiment measures the search depth and number of nodes for each algorithm. All experiments are conducted on the same computer with an Intel i7-9700 processor and 16GB of memory. The experiment sets single-step time limits of 10s, 30s, and 60s, respectively, and the results are shown in Fig. 5(a), 5 (b), and 5(c).

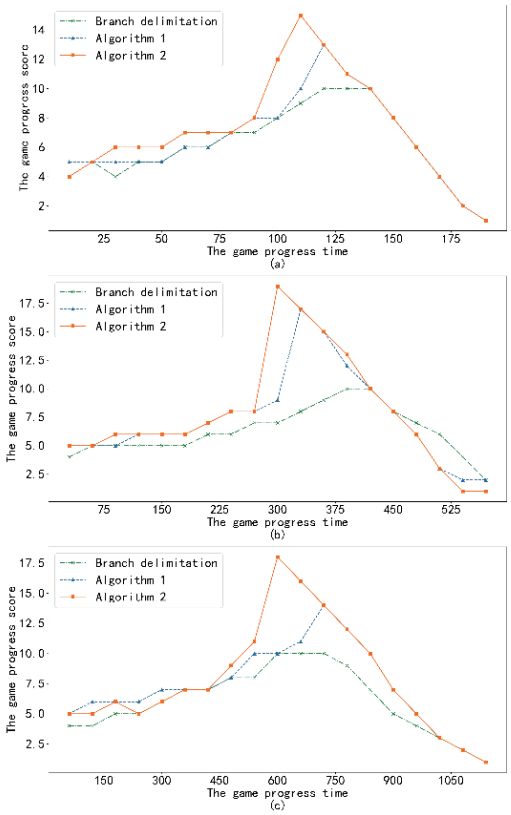

Fig. 5. Comparison of the search depth of the three algorithms.

Algorithms 1 and Algorithms 2 outperform the branch delimitation algorithm in terms of search depth, reaching the maximum depth consistently. All three algorithms show no decrease in the search depth as time limit increases. Heuristic search application in subsequent searches leads to a significant increase in search depth for Algorithms 1 and 2, with Algorithm 2 performing better due to its two-way search strategy. In rounds 9-14, chain and loop states cause an earlier increase in the search depth and hash table node storage for Algorithms 1 and 2, resulting in an overall increase in search depth. At search depth t=110s, Algorithm 2 achieves optimal search efficiency with a search depth of 15, outperforming Algorithms 1 and the branch delimitation algorithm.

B. Node Tree Experiment and Analysis

The environment setup of this experiment is the same as the experiment in A. The comparison experiments with the game process time as the independent variable and the number of game tree nodes as the dependent variable was conducted with single-step time limits of 10s, 30s, and 60s, respectively, and the experimental results are shown in Fig. 6(a), 6(b), and 6(c), respectively.

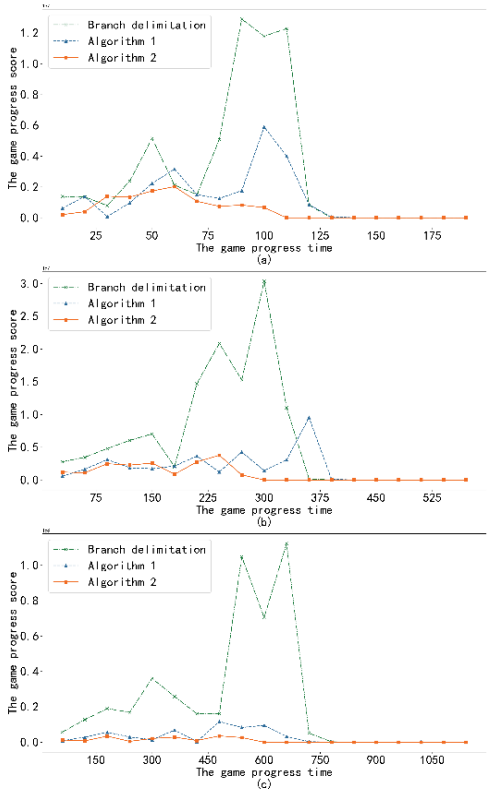

Fig. 6. Comparison of the number of nodes of the three algorithms.

The study compared the performance of Algorithm 1 and Algorithm 2 with the branch delimitation algorithm in optimizing the search performance of dots and boxes. The results showed that the branch delimitation algorithm had a higher node search count and searched a large number of invalid nodes. As the time limit increased, the number of nodes searched by all three algorithms increased, but Algorithms 1 and 2 had an advantage due to their fast lookup of node hash tables. Algorithm 2 had a 37% improvement in the search efficiency compared to the branch delimitation algorithm.

C. Game Efficiency Experiments and Analysis

The environment setup of this experiment is the same as the experiment in A. The purpose of this experiment is to compare the scores of branch delimitation algorithm, Algorithm 1 and Algorithm 2 in the game to determine the optimal algorithm for the game. Each game was played for 100 games. The final results were obtained using the average score data. The total score of the game is 25, and a player wins absolutely when the score of one player is greater than 12. The experimental results are shown in Fig. 7, Fig. 8 and Fig. 9 for a single-step game with time limits of 10s, 30s and 60s for the three algorithms played two-by-two.

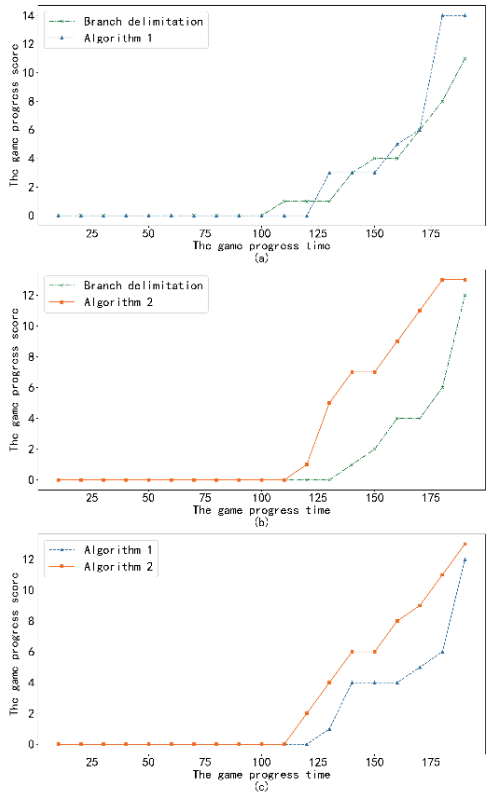

Fig. 7. T=10s three algorithm game score graph.

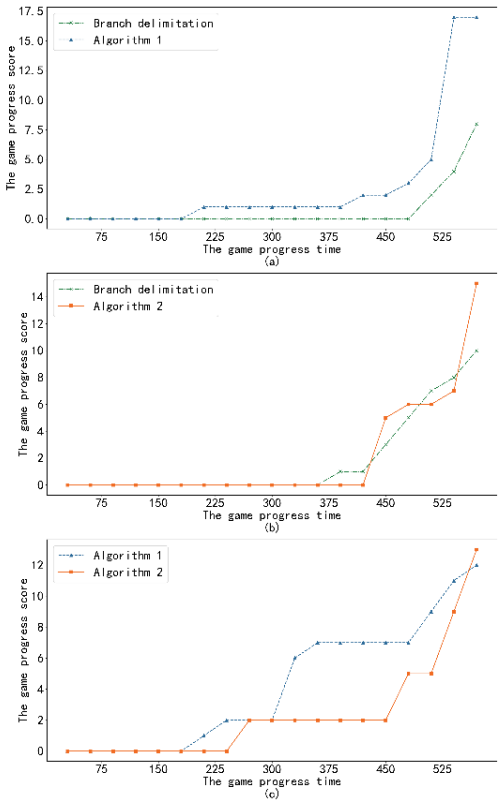

Fig. 8. T=30s three algorithm game score graph.

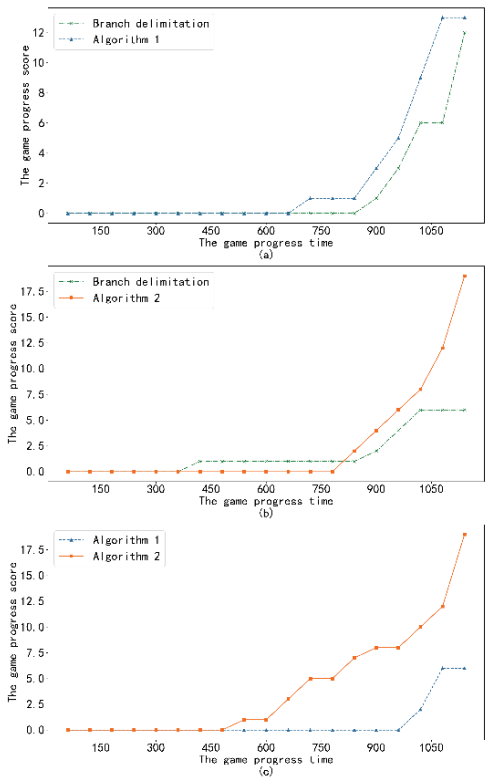

Fig. 9. T=60s three algorithm game score graph.

It is evident from the results shown in Fig. 7, 8 and 9 that Algorithms 1 and Algorithms 2 consistently achieve absolute victory in terms of score within the specified time limits. In subplots (a) and (b) of Fig. 8, both algorithms secure victory one round early with average scores of 14 and 13, respectively. Furthermore, as the time limit increases, both algorithms perform even better in terms of scoring. During rounds 14-19, the game usually witnesses an exponential increase in scores due to the high number of squares with degrees of freedom 2 and 3, along with the emergence of the stumping theorem state. Algorithm 1, as shown in Fig. 7(a), even managed to increase the average score in the 18th round of the game by 12. Algorithm 2 wins 68 times against branch delimitation algorithm and Algorithm 1, which is a 42% improvement compared to the number of times the branch delimitation algorithm wins.

VI. Conclusion

In this paper, we have addressed the limitations of conventional branch delimitation algorithms by proposing a novel approach that significantly enhances the search efficiency. Previous algorithms often suffer from searching through numerous invalid nodes, leading to reduced efficiency. While history-inspired pruning and iterative deepening strategies have been employed to improve efficiency, they still face challenges such as unassigned or inaccurately assigned initial nodes and repetitive searches. To overcome these limitations, we have introduced a hash storage search scheme specifically tailored for the evaluation function and game tree search, using dots and boxes as a case study. Our proposed branch delimitation algorithm combines the advantages of

history-inspired and iterative deepening methods, while incorporating a game tree bidirectional search algorithm to further enhance efficiency. Experimental results have demonstrated the effectiveness of our optimized algorithm in reducing the number of nodes in the game tree while maintaining the desired search depth. Moreover, the algorithm has shown remarkable improvements in the chess performance. Particularly, it excels in scenarios characterized by a large number of chains and loops, where the likelihood of position repetition is higher. Overall, our optimized algorithm presents a significant advancement in the field of game search, offering improved efficiency and performance. The introduced hash storage search scheme and the combination of branch delimitation and bidirectional search strategies provide valuable contributions to overcoming computational challenges in game tree exploration. Further research and experimentation can explore the algorithm's applicability in other domains and its potential for solving complex problems with repetitive patterns.

ACKNOWLEDGMENT

Key Natural Science Research Projects in Anhui Province (Item No: KJ2019A0681);Chaohu College Collaborative Education and Innovation Experimental Zone "Experimental Zone of Big Data Innovation Application Based on School-Local Collaborative Education (Item No: kj20xycs01);National Undergraduate Innovation Program (Item No: 202110380040).

REFERENCES

[1] Lv, Z.; Lou, R.; Li, J.; Singh, A.K.; Song, H. Big data analytics for 6G-enabled massive internet of things. IEEE Internet Things J.2021, 8, 5350–5359.

[2] Tang, C.; Zheng, X.; Liu, X.; Zhang, W.; Zhang, J.; Xiong, J.; Wang, L. Cross-view locality preserved diversity and consensus learning for multi-view unsupervised feature selection. IEEE Trans. Knowl. Data Eng. 2021, 34, 4705–4716.

[3] Jin, J.; Xiao, R.; Daly, I.; Miao, Y.; Wang, X.; Cichocki, A. Internal feature selection method of CSP based on L1-norm and Dempster-Shafer theory. IEEE Trans. Neural Netw. Learn. Syst. 2020, 32, 4814–4825.

[4] Luo, F.; Zou, Z.; Liu, J.; Lin, Z. Dimensionality reduction and classification of hyperspectral image via multistructure unified discriminative embedding. IEEE Trans. Geosci. Remote Sens. 2021, 60, 1–16.

[5] Zhang, Y.; Meng, K. Research and analysis of UCT algorithm based on point-grid chess. Intell. Comput. Appl. 2020, 10 (4), 27-31.

[6] Gao, R.; Han, B.; Wang, D.; Liu, G. Retrieval method of access control policy based on sparse index and hash table. J. Jiangsu Univ. Sci. Technol. (Nat. Sci. Ed.) 2021, 35 (4), 50-57.

[7] Li, D.; Hu, W.; Wang, J. Research on the Sulakarta chess game system based on the Alpha-Beta algorithm. Intell. Comput. Appl. 2022, 12 (2), 123-125.

[8] Zhu, L.; Wang, J.; Li, Y. Research on point-grid chess game system based on UCT search algorithm. Intell. Comput. Appl. 2021, 11 (2), 129-131.

[9] He, X.; Hong, Y.; Wang, K.; Peng, Y. Design of search strategy and value function in machine game: take six chess as an example. Comput. Knowl. Technol. 2019, 15 (34), 53-54+61.

[10] Jin, Q.; Wang, J.; Fu, X. Cuckoo hash table based on intelligent placement strategy. Comput. Sci. 2020, 47 (8), 80-86.

[11] Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the game of go without human knowledge. Nature 2017, 550 (7676), 354-359.

[12] Brown, M. R.; Saffidine, A. Computer-aided retrograde analysis of chess with Nalimov endgame tablebases. ICGA J. 2018, 39 (1), 24-34.

[13] Björnsson, Y.; Enzenberger, M. Opening book generation for Monte Carlo tree search in games. IEEE Trans. Comput. Intell. AI Games 2018, 10 (4), 338-350.

[14] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., . & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484-489.

[15] Tian, Z., & Zhu, B. (2020). Solving three-person Chinese chess via Monte Carlo tree search. Journal of Ambient Intelligence and Humanized Computing, 11(8), 3521-3529.

[16] Yang, Z., Wang, Q., Zhang, Y., Jiang, F., & Liu, X. (2021). A deep reinforcement learning framework for the game of six. Computers & Mathematics with Applications, 82, 1989-2000.

[17] Yang, Z., Wang, Q., Zhang, Y., & Jiang, F. (2021). A novel reinforcement learning approach for the game of six. IEEE Transactions on Cybernetics.

[18] Tsitsiklis JN, Van Roy B. Analysis of prioritized sweeping with function approximation. IEEE Transactions on Automatic Control. 2018;64(3):1234-1241.

[19] Rudin C, Madigan D. A scalable Bayesian approach to sparse superposition of regression models. Journal of the American Statistical Association. 2017;112(519):953-964.

[20] Horiyama T, Hashimoto T. Hashing-based techniques for efficient k-nearest neighbor search on sparse high-dimensional data. Journal of Parallel and Distributed Computing. 2018;120:89-97.

[21] Guo Y, Zhang L, Huang Q, Li L. A Survey on the Application of Deep Learning in Recommender Systems. IEEE Transactions on Neural Networks and Learning Systems. 2020;31(10):3774-3792.

[22] Lelis LA, Freitas AA. Hash-Based Feature Selection for High-Dimensional Regression. IEEE Transactions on Neural Networks and Learning Systems. 2020;31(4):1074-1084.

[23] Patrini G, Rozza A, Menon AK. Making deep neural networks robust to label noise: a loss correction approach. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017:1944-1952.

[24] Yu L, Rui Y, Wang F, Zhang Y, Li X. HashGAN: Deep Learning to Hash with Pair Conditional Wasserstein GAN. In: Proceedings of the IEEE International Conference on Computer Vision. 2017:3137-3145.

[25] Elzeheiry Heba Aly, Barakat Sherief, Rezk Amira Different Scales of Medical Data Classification Based on Machine Learning Techniques: A Comparative Study[J] Applied Sciences, 2022, 12(2).

[26] Juan Dubra, Martín Egozcue, Luis Fuentes García Optimal consumption sequences under habit formation and satiation[J] Journal of Mathematical Economics, 2018, 80.

[27] Liu Ke,Lv Xue-feng. Research on Palletizing and Packing Based on Heuristic Algorithm[J]. Journal of Physics: Conference Series,2023,2449(1).

[28] Chen G, Zhu F, Heng P A. Large-scale bayesian probabilistic matrix factorization with memo-free distributed variational inference [J]. ACM Transactions on Knowledge Discovery from Data, 2018, 12(3): 31.1-31.24.

[29] Taherpour M, Jalali M, Shakeri H. ECAT: an enhanced confidence-aware trust-based recommendation system [C] Proceedings of the 8th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS). IEEE, 2020: 180-185.

[30] Wang W, Chen J, Wang J, et al. Trust-enhanced collaborative filtering for personalized point of Interests recommendation [J]. IEEE Transactions on Industrial Informatics, 2020, 16(9): 6124-6132.

[31] Belkhadir I, Omar E D, Boumhidi J. An intelligent recommender system using social trust path for recommendations in web-based social networks [J]. Procedia Computer Science, 2019, 148: 181-190.