Cluster-based Alpha-Beta Search for Real-Time Strategy Games

X Han¹, H P Yan¹, *, J G Zhang² and L F Wang³

¹ Department of Information Engineering, China University of Geosciences, No. 29 College Road, Haidian District, Beijing, China
² CRIPAC & NLPR, Institute of Automation, Chinese Academy of Sciences, Beijing, China, No. 95 Zhongguancun East Road, Haidian District, Beijing, China
³ NLPR, Institute of Automation, Chinese Academy of Sciences, Beijing, China, No. 95 Zhongguancun East Road, Haidian District, Beijing, China

* Corresponding Author: yanhp@cugb.edu.cn

Abstract. From an artificial intelligence point of view, Real-Time Strategy (RTS) game has been proven to be one of the most challenging areas. Due to the huge action state space, partial observability and real time property, the previous AI solutions are still slow in terms of speed. In this paper, we propose a Cluster-based Alpha-Beta Considering Durations (CABCD) algorithm that searches the optimal action for each unit based on the cluster. When the improved algorithm is applied to large RTS games, the enormous branching factors are significantly reduced. The approach is evaluated in StarCraft I, and outperforms competing methods promisingly.

1. Introduction

Real-Time Strategy (RTS) game belongs to a kind of strategy game that progresses immediately instead of incrementally in turns. In RTS game, we can accomplish activities including building, producing soldiers, developing technology or fighting with the enemy [1]. Due to the partial observability and the huge action state space, real-time strategy games still face plenty of challenges [2, 3]. It is difficult to build an integrated artificial intelligence to solve RST games, therefore, amounts of work split the tasks into collecting resources, producing soldiers, and fighting, etc.

War is one of the core components of the RTS game. The most important operation of a war is to choose the action of each unit. Nowadays, researchers mainly use Alpha-Beta search [4, 5], UCT [6, 7], Portfolio [8] or other algorithms to solve the problem mentioned above. However, RTS game requires huge resources to possess the accumulated units (more than 10 units) for actions and states, which hinders the applicability of the game tree search method such as Alpha-Beta search, UCT algorithm, etc. In order to ensure that the optimal action is obtained within a limited time, most work adopts the strategy of exchanging time with accuracy. For example, Churchill D et al. used Portfolio to search strategy, then adopted the strategy to determine the action. However, using these methods leads to less accurate action, lower intelligence, and less adaptability to the environment. So the war scene in the RTS game poses a great challenge to artificial intelligence.

Churchill D et al. have demonstrated that the Alpha-Beta search algorithm searches better actions when the number of units is small [4]. As shown in Figure 1, with the increase of the number of the
units and actions, the number of branches searched by Alpha-Beta search algorithm increases exponentially.

Figure 1. The relationships of the number of units, actions and the branches searched in the Alpha-Beta search algorithm.

This paper proposes an improved Alpha-Beta search algorithm to reduce the action state space, that is, the Cluster-based Alpha-Beta Considering Durations (CABCD) algorithm. CABCD can perform better when the action state space is large. The main idea of the CABCD is to aggregate units to reduce the number of branches and ultimately reduce the action state space. The specific steps of CABCD are: Firstly, CABCD groups units according to their locations and types; Secondly, CABCD performs the Alpha-Beta search algorithm for each group to find the current optimal action of each unit; Finally, CABCD integrates the best actions of units in all groups and implements them simultaneously.

The main contributions of this paper are listed as follows:

1). This paper proposes the CABCD algorithm to achieve the purpose of determining the optimal action of each unit within a limited time, meanwhile, CABCD ensures the accuracy of the searched action.

2). The CABCD algorithm is built to enable the agent to construct the Alpha-Beta tree according to the group to achieve the effect of reducing the action state space and shortening the search time.

The experimental results show that the CABCD algorithm can search for the better action in a limited time. The winning rate is above 0.5 comparing with other search algorithms. In addition, the influence of cluster on decreasing the number of branches is qualitatively analysed in this paper.

2. Related work

One of the earliest and most classic methods in the tactical decision-making of RTS games uses tree search to determine the action for each unit. Stanescu et al. proposed a hierarchical implementation of the search to reduce the action state space [9]. Ontanón S presented a new Monte Carlo Tree Search (MCTS) algorithm based on Naive Sampling [10]. Uriarte A and Ontanón S put forward the game state abstraction [11]. The map is decomposed into regions, and then the action based on the region is selected. However, the grouping error based on the region is large, and the units at the two regional boundaries might be divided into different groups. Ontanón S brought forward an improvement of the MCTS algorithm combined with the Bayesian probability distribution estimation model [12].

Notably, the RTS game is the real-time game, when the number of units is large, there is a huge space of states. Therefore, it is impossible to use a tree search algorithm to traverse the action state space to select suitable actions within a limited time. So most work proposes to find suitable strategies for the unit, such as attacking the nearest unit, attacking the lowest hit point of the unit, and then select the action according to the strategy. Churchill D et al. proposed the Portfolio Greedy Search algorithm to select actions via searching the predefined strategies for each unit [8]. Wang C et al. presented the improvement of the Portfolio Greedy Search algorithm that combined with online evolution [13]. Nicolas A. B et al. put forward a new tactical search framework based on strategy [14]. Although searching strategy can reduce the number of searched branches, it also reduces the accuracy of the
searched action. Therefore, this paper first reduces the action state space by clustering, and then determines the action of each unit in a more precise decision-making way.

The work of Justesen N et al. is similar to our work, which groups units and then applies Upper Confidence Bound Apply to Tree (UCT) algorithm to select the appropriate strategy for each cluster [15]. The main difference is that we use the Alpha-Beta search algorithm to determine the action for each unit after grouping. The advantage of our work lies in that the Alpha-Beta search algorithm can achieve higher accuracy when the action state space is smaller. The more appropriate action is determined for each unit, the higher accuracy can be obtained, which will result in the higher winning rate.

3. Background
In this section, we first introduce the Alpha-Beta search algorithm, and then provide the improved method which is clustering.

3.1. Alpha-Beta tree search algorithm and Alpha-Beta Considering Durations algorithm
Minimax search algorithm is commonly used in game search, such as Go, chess, etc., as shown in Figure 2. The Minimax search algorithm traverses the entire game tree to select the optimal route as far as possible. As the search depth increases, the tree grows exponentially. Therefore, Knuth D E et al. proposed the Alpha-Beta search, which is an improved algorithm of Minimax search algorithm, reducing the number of searched branches by pruning to make the result more accurate [16].

Alpha-Beta search based on the theory that when you have a choice, as long as the other options will not be better than the current situation, we can not explore other options and abandon them. The idea is to transfer two parameters in the search. The first parameter is Alpha, which is the minimum lower bound for the best score; the second parameter is Beta, which is the maximum upper bound for the best score.

3.2. Cluster
Applying cluster is the extension of ABCD. Cluster refers to the process of grouping multiple classes of similar objects in a collection of physical or abstract objects.
Given the input \{u_i\}_{i=1}^n \in U_j \in P_j$, the K-means algorithm is applied to CABCD,

$$u_i \rightarrow \tau_i(u)$$ (1)

Where $P_j$ is the player $j$, and $U_j$ is all $n$ units owned by $P_j$. $u_i = \{(x_i, y_i), t\}$ indicates the unit $i$, $(x_i, y_i)$ represents the coordinate of $u_i$, and $t$ is the type of $u_i$, such as the Zealot or the Dragoon in StarCraft II.

As shown in equation (1), the unit $u_i$ is assigned to the group $c$, $1 \leq c \leq k$. $k$ is the total number of current groups.

Units with similar positions and more consistent types would be assigned to one group, and their movements would be similar. However, it is not certain that the same type of units will be assigned to one group. This paper considers equalizing firepower when grouping.

4. Cluster-based Alpha-Beta Considering Durations algorithm
This section first introduces the combat model. Then the details of implementation are described and we propose the Cluster-based Alpha-Beta Considering Durations (CABCD) algorithm. Finally, we theoretically analyze the influence of the grouping number $k$ on time complexity.

4.1. The combat model
The CABCD algorithm is evaluated in SparCraft and modifies the combat model of SparCraft, which can be used to construct Alpha-Beta trees in groups. The SparCraft combat model consists of three data components and two logical components.

State $s = \langle t, U_1, U_2 \rangle$
- Current game time $t$
- Sets of units $U_i$ under control of player $i$

Unit $u = \langle p, hp, t_a, t_m, c, k, type \rangle$
- Position $p = \langle x, y \rangle$ in $R^2$
- Current hit points $hp$
- Time step when unit can next attack $t_a$, or move $t_m$
- Current group number $c$.
- The total number of current groups $k$.
- StarCraft unit type, defining all static unit properties such as damage, maximum hp, armor, speed, etc.

Move $m = \langle a_1, ..., a_k \rangle$ which is a combination of unit actions $a_i = \langle u, type, target, t \rangle$, with
- Unit $u$ to perform this action
- The type of action to be performed: Attack unit target, Move $u$ to position target, or Wait until time $t$

Player function $p \{ m = p(s, U) \}$
- Input state $s$ and units $U$ under player’s control
- Perform Move decision logic
- Return move $m$ generated by $p$

Game function $g \{ r = g(s, p_1, p_2) \}$
- Initial state $s$ and players $p_1$, $p_2$
- Perform game simulation logic
- Return game result $r$ (win, lose or draw)

4.2. CABCD algorithm
CABCD algorithm is as shown in Algorithm 1. The core idea is to use the K-means to reduce the branching factors, then build the Alpha-Beta tree for each group. In this paper, the experimental environment is the classic RTS game StarCraft I. When human players play StarCraft I, they tend to divide units of similar types and positions into one group. Consequently, we consider the location and type of units when grouping, then find the optimal action for each unit. Figure 3 shows an example of grouping in StarCraft I.

**Algorithm 1 Cluster-based Alpha-Beta Considering Durations**

1: Initialize \(d, m_0, \alpha, \beta\)  
2: Obtain \(s\)  
3: if \(s.num < 10\) then  
4: \(ABCD(s, d, m_0, \alpha, \beta)\)  
5: else  
6: \(data \leftarrow x, y, s, type\)  
7: \(K\text{-}means(data)\) \(//\)Use \(K\text{-}means\) grouping and store the group number \(c\).  
8: for \(i = k\) to 1 do  
9: \(\text{for } j = n \text{ to 1 do}\)  
10: if \(c_j = i\) then  
11: \(s_i = [s_i; s_j]\)  
12: \(\text{end for}\)  
13: \(ABCD(s_i, d, m, \alpha, \beta)\)  
14: \(\text{end for}\)

The enemy can also be grouped, but this experiment considers the best situation where the enemy is not grouped, which is equal to all the units of the enemy is one group.

![Grouping Example](image)

**Figure 3.** The example of grouping in StarCraft I.

There are two main problems when grouping. The first is whether the units should be the same type. In this paper, we just take the weighting type into account, and select the action for each unit in the same group, instead of selecting a strategy for the group. As a result, we do not need to stipulate the same type of the unit must be in the same group.
The second problem is that the total number of current groups $k$ should be selected dynamically. In RTS games it is one of the core elements of the game to save time, whereas dynamic mechanism takes a lot of time. Therefore, we set $k$ manually which uses less time and is more efficient. The number of the group should balance the higher number of units in the group and the performance requirements of the computer. Over-dividing groups must require a relatively high performance for computer. The results of ABCD in different numbers of units are shown in Figure 4. The maximum number of 10 units in each group is superior. The number of units in the group should not be divided equally.

When the player has units fewer than 10, the action state space has been greatly reduced. And no further grouping is required. So all units belong to one group that directly builds an Alpha-Beta tree to find the best action for each unit.

![Figure 4. The winning rate between Alpha-Beta and Portfolio Greedy Search.](image)

4.3. The influence of $k$ on the time complexity of CABCD

Theorem 1. For $b \in \{1, 2, \cdots, n\}$, the number of total nodes searched by Minimax algorithm is:

$$N_d \approx b^d$$

In equation (2), $b$ is the average number of branching factors in the search tree, $d$ is the search depth. The complexity of algorithm is $O(b^d)$. Theorem 1 is proved in the appendices.

Theorem 2. For $b \in \{1, 2, \cdots, n\}$, the total number of nodes searched by Alpha-Beta search algorithm is:
In equation (3), \( N_d \) is the total number of the searched nodes when the depth of search is even, \( N'_d \) is the total number of the searched nodes when the depth of search is odd. If the nodes are optimally arranged, the algorithm complexity is \( O(b^{d/2}) \); If the nodes are arranged randomly, the algorithm complexity is \( O(b^{3d/4}) \). This theorem has been proved by Knuth D E et al [16].

Consequently, according to Theorem 1, the number of branches increases drastically with the depth, in other words, the number of search points increases exponentially with the number of branches. Hence, this paper proposes to group the units to significantly reduce the number of search points, and the quantitative analysis is shown in Theorem 3.

Theorem 3. The max number of nodes searched using CABCD algorithm is:

\[
\begin{align*}
N^G_d &= k^*[2^* (b_{\text{max}})^{d/2} - 1] \\
N'_d &= k^*[ (b_{\text{max}})^{(d+1)/2} + (b_{\text{max}})^{(d-1)/2} - 1]
\end{align*}
\]

(4)

In equation (4), \( m \) is the number of actions of each units. \( k \) is the total number of groups. \( n_c \) is the number of units in group \( c \). Theorem 3 is proved in the appendices.

From Theorem 3, we know that \( N'_d \approx N^G_d \) (or \( N_d \approx N'_d \)). So after grouping it can exponentially decrease the number of the branches to reduce the search points. In the case of limited time and depth, it is possible to search all possibilities on the first floor to make the results more accurate.

Figure 5 (a) represents the difference of the maximum number of nodes between grouping and no grouping. It can be seen that there is an exponential increase in the number of nodes compared with no grouping.

The difference between grouping and no grouping is large. So in figure 5 (b), only a few cases of grouping are shown. The number of branches per group 1 (or 5) unit is very small, but it constructs plenty of the Alpha-Beta trees. With the reasonable allocation arrangement, the effect of uneven distribution is better.

5. Experiment

In this section, we evaluate the algorithm CABCD in the war platform SparCraft of the classic RTS game-StarCraft I, which mainly set up a group of experiments to compare CABCD, ABCD, UCTCD, and Portfolio Greedy Search algorithm.

5.1. Test scenarios

The algorithm is tested in SparCraft. Each player controls the same units and sets the parameter \( n \) in the scene to determine the number of units each player owns. There are five scenarios. The experimental result uses Equation (5) to calculate the winning rate.

\[
\text{Winning rate} = \frac{(\text{Wins} + \text{Draws} / 2)}{\text{total}}
\]

(5)

\( \text{Wins} \) indicates the number of wins. \( \text{Draws} \) is the number of draws. \( \text{total} \) represents the total number of tests, which is 100 there. When the number of units is less than 10, no grouping is performed.

5.2. Environment configuration

All experiments run on Windows 8.1 Intel (R) Core (TM) i5-5287U CPU @ 2.90GHz, and a total of 8GB of RAM is available. All experiments are single-threaded.

Search algorithm settings

Configuration of all Alpha-Beta search algorithms:
- Time Limit: 40 ms
- Max Children: 20
- Evaluation: NOK-AV vs. NOK-AV Playout
- Transposition Table Size: 100000 (13.2 MB)

**CABCD:**
- Max Clusters: 5
- The Minimum Number of Units Divide into Groups: 10

**UCT search:**
- Time Limit: 40 ms
- Max Children: 20
- Evaluation: NOK-AV vs. NOK-AV Playout
- Final Move Selection: Most Visited
- Exploration Constant: 1.6
- Child Generation: One-at-leaf
- Tree Size: No Limit (6 MB largest seen in 40 ms)

**Portfolio Greedy search:**
- Time Limit: 40 ms
- Improvement Iterations I: 1
- Response Iterations R: 0
- Initial Enemy Script: NOK-AV
- Evaluation: Improved Playout
- Portfolio Used: (NOK-AV, Kiter)

---

**Figure 6.** Results of CABCD vs other algorithms. (a) Results of CABCD vs Alpha-Beta; (b) Results of CABCD vs UCTCD; (c) Results of CABCD vs Portfolio. The results show the war for n vs n units in five different scenarios, n=8, 16, 32, 50.

5.3. Experimental results

Figure 6 shows the experimental results of CABCD vs ABCD, UCTCD and Portfolio Greedy Search algorithm in different scenes. Abscissa corresponds to the scene, P_D means that each player only controls Protoss_Dragoon; Z_Z means that each player only controls Zerg_Zergling; n/2 P_D n/2
P_Z means that each player controls \( n/2 \) Protoss_Dragoon and \( n/2 \) Protoss_Zealot; \( P_D \)
\( n/2 \) T_M means that each player controls \( n/2 \) Protoss_Dragoon and \( n/2 \) Terran_Marine; \( T_M \)
\( n/2 \) Z_Z means that each player controls \( n/2 \) Terran_Marine and \( n/2 \) Zerg_Zergling. Run
100 innings in each scenario.

It can be seen from Figure 6 that the winning rate of CABCD vs ABCD, UCTCD can be as high as
0.9 when the unit is the Protoss. Because of no use of parallelism, the winning rate of CABCD vs
Portfolio Greedy Search algorithm decreases with the increase of the unit number, but it can also reach
about 0.7. When the number of units is large, the more Alpha-Beta trees are constructed. The time of
constructing each tree in a limited time is less leading that it can not search for better action.

When the unit is Zerg, the winning rate of CABCD is very low. CABCD needs to be grouped first,
which takes time. However, the Zerg has low blood volume. Therefore, it may be defeated while
grouping.

CABCD vs ABCD still has a higher rate of success in less units, namely 8 vs 8. Since the CABCD
is divided into a group when less than 10 units, it is more likely to concentrate on fire to win.

Figure 7 shows the overall test results of CABCD vs ABCD, UCTCD and Portfolio Greedy Search
algorithm in the comprehensive five scenarios, which indicates that the winning rate of CABCD is
above 0.5. Figure 7 fully illustrates the effectiveness of CABCD algorithm.

![Figure 7](image_url)

**Figure 7.** The total winning rate of CABCD vs ABCD, UCTCD and Portfolio in five scenarios

### 6. Conclusions and discussion

Although search algorithms have made progress in recent years, choosing appropriate actions in large
RTS games is still a great challenge. The difficulty is that the RTS game has a large amount of action
state space, is partially observability and real-time. In this work, we propose CABCD algorithm,
which is an extension of the ABCD algorithm for searching better actions for each unit. CABCD is a
search algorithm for RTS games with a large number of units.

Our experimental evaluation shows that the CABCD algorithm is superior to the ABCD, UCTCD
and Portfolio Greedy Search algorithms in the RTS game. The winning rate is above 0.5. CABCD
algorithm can search for better actions for each unit when the number of units is large, and it can also
be applied to the search problems with the large action state space, not only in RTS games.

However, the CABCD algorithm has a low winning rate when using the Zerg, because the
grouping takes time and the Zerg blood volume is lower, which may be eliminated when grouping.
This should be considered in the future work.

### 7. Appendices

**Proof of Theorem 1**

For \( b \in \{1,2,\cdots,n\} \), the total number of nodes \( N_d \) searched by the minimal maximum algorithm is:

\[
N_d = 1 + b + b^2 + \cdots + b_d
\]

\[
= b^d - 1 - (1/b^d)
\]

\[
= b^d - (1/b)
\]

\[
\approx b^d
\]

(6)
\( b \) is the average branch number of the search tree, \( d \) is the search depth (Approximate value is obtained when \( b \) is large enough).

Proof of Theorem 3
The total number of nodes searched for Alpha-Beta search algorithm is:

\[
\begin{align*}
N_d &= 2b^{d/2} - 1 \\
N'_d &= b^{(d+1)/2} + b^{(d-1)/2} - 1
\end{align*}
\]

After grouping:

\[
m \sum_{i} n_i = m^N = b
\]

\( m \) is the number of actions of the unit, \( N \) is the total number of units currently owned by the player.

\[
\begin{align*}
n_{\text{max}} &= \max_i n_i \\
b_{\text{max}} &= m^{n_{\text{max}}}
\end{align*}
\]

Therefore \( b_{\text{max}} \) is the maximum number of branches after grouping. The max searched points after grouping are:

\[
\begin{align*}
N_{d}' &= k \times [2 \times (b_{\text{max}})^{d/2} - 1] \\
N_{d} &= k \times [(b_{\text{max}})^{(d+1)/2} + (b_{\text{max}})^{(d-1)/2} - 1]
\end{align*}
\]

References
[1] Ontañoñ S, Synnaeve G, Uriarte A, Richoux F, Churchill D, Preuss M et al. 2013 A survey of real-time strategy game AI research and competition in starcraft J. IEEE Transactions on Computational Intelligence and AI in games 5 293-311
[2] Robertson G and Watson I 2014 A review of real-time strategy game AI J. AI Magazine 35 75-104
[3] Šustr M, Malý J and Čertický M 2018 Multi-platform Version of StarCraft: Brood War in a Docker Container: Technical Report J. Preprint arXiv/1801.02193
[4] Churchill D, Saffidine A and Buro M 2012 Conf. Artificial Intelligence and Interactive Digital Entertainment Conference (USA: Stanford University) pp 112-7
[5] Zhang C P, Liu C H and Xu J C 2008 Research on Pruning Technique in Game Tree Heuristic Search J. Computer Engineering and Applications 44 54-6
[6] Silver D and Schrittwieser J et al. 2017 Mastering the game of Go without human knowledge J. Nature 550 354-9.
[7] Silver D and Huang A et al. 2016 Mastering the game of Go with deep neural networks and tree search J. Nature 529 484-9
[8] Churchill D and Buro M 2013 Conf. Computational Intelligence in Games (CIG) 2013 IEEE (Canada: Niagara Falls) pp 1-8
[9] Stanescu M, Barriga N A and Buro M 2014 Conf. Artificial Intelligence and Interactive Digital Entertainment Conference (USA: Raleigh)
[10] Ontañoñ S 2013 Conf. Ninth Artificial Intelligence and Interactive Digital Entertainment Conference (USA: Northeastern University)
[11] Uriarte A and Ontañoñ S 2014 Conf. Tenth Artificial Intelligence and Interactive Digital Entertainment Conference (USA: Raleigh)
[12] Ontañoñ S 2016 Conf. Computational Intelligence and Games (CIG) 2016 IEEE (Greece: Santorini) pp 1-8
[13] Wang C, Chen P, Li Y, Holmgård C and Togelius J 2016 Conf. Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference (USA: San Francisco)
[14] Barriga N A, Stanescu M and Buro M 2015 Conf. Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference (USA: California University)
[15] Justesen N, Tillman B, Togelius B and Risi S 2014 Conf. Computational Intelligence and Games (CIG) 2014 IEEE (Germany: Dortmund) pp 1-8
[16] Knuth D E and Moore R W 1975 An analysis of alpha-beta pruning J. Artificial intelligence 6 293-326