HTN Guided Adversarial Planning for RTS Games

Conference Paper · November 2020

DOI: 10.1109/ICMA49215.2020.9233614

2 authors, including:

Sun Lin
National University of Defense Technology
19 PUBLICATIONS **43** CITATIONS

SEE PROFILE

HTN Guided Adversarial Planning for RTS Games

Sun Lin, Zhu Anshi, Li Bo and Fan Xiaoshi

Joint Logistics Collage
National Defense University
Beijing, CHINA

mksl163@nudt.edu.cn

***Abstract* – The real-time strategy (RTS) game has become a suitable testbed for artificial intelligence (AI) technologies owing to the large state space, limited decision times, partial observation environment and adversarial behavior involved. To address these challenges, the approach named AHTN which combined hierarchical task network (HTN) with game tree search has been applied in RTS games and achieved favorable performance. However, both the opponent strategy and time constraint of generating the sub-game tree are not considered by AHTN during planning. Therefore, this paper proposes a novel method which considers the time constraint when generating the subgame tree by HTN planning and introduces the opponent strategy as the adversarial player. The empirical results are presented based on a μRTS game and show our algorithm performs better than original HTN planning methods.**

Index Terms - adversarial planning, real-time strategy game, game tree search , decision time, HTN

I. INTRODUCTION

The real-time strategy (RTS) game has bring great challenges for artificial intelligence (AI) owing to the very large state spaces, partial observation environment, limited decision times and adversarial behavior involved. [1]. Lots of AI researchers regard RTS games as a simplification of the real-life environment and a suitable testbed for investigating activities such as adversarial planning [2]. Moreover, AI techniques that have been demonstrated to be effective in RTS games could also be applied in real-world domains [2].

The board games such as *Go* also can serve as a suitable testbed for AI techniques. However, approaches which perform well in board games cannot play well in RTS games owing to the differences between these two kinds of games. Compared with the board games, the primary differences are as follows [2]:

1) large state space: the state space of RTS games is much larger than board games'. For example, a typical 128×128 map in *StarCraft* which is a kind of typical RTS games generally includes about 400 units. Only considering the location of each unit, the number of possible states is about 10^{1685}, whereas the state space of *Go* is around 10^{150}.

2) concurrent and simultaneous actions: in RTS games both sides can pursue actions by employing multiple units concurrently without considering turn by turn.

3) actions are durative: each action requires numerous steps to be executed. This is also much more complex than conventional board games. In board games, the actions must be completed in a single turn.

4) partial observation environment: The environment of an RTS game is partial observation because of the fog of war while the board games are completed observable.

Because of the above differences, the game tree search methods which are suitable for board games may not perform well in RTS games [3]. Therefore, many variants of game tree search methods have been proposed to address above differences [4]. Chung [5] investigated the applicability of Monte Carlo game tree search (MCTS) in RTS games. Balla and Fern [6] address the complications associated with durative actions by the upper confidence bound for trees (UCT) algorithm, which is a variant of MCTS. Balla's method is believed to adapt automatically to the effective smoothness of the tree. Churchill et al. [7] propose an extending alpha-beta pruning method called Alpha Beta Considering Durative (ABCD) to address the complications associated with simultaneous and durative actions. However, the challenge of large state space is still left. In order to reduce the state space during searching, some researchers propose methods which take advantage of domain knowledge to guide the search directions. Therefore, HTN planning is applied into RTS games [8][9][10]. However, most researchers applied the HTN planning method into RTS games directly. Few researchers modify the HTN planning approach or combine it with other methods owing to the features of RTS games.

Ontañón and Buro [10] propose an algorithm called adversarial hierarchical task network (AHTN) which takes advantage of HTN to reduce the state space of game tree search. However, the algorithm doesn't consider its opponent strategy. It uses the same HTN planner as the opponent player. And it does not consider the time constraint when generating the subgame tree. In this paper we introduce the opponent player as the HTN planner's enemy during planning and limit the time spending on generating the subgame tree. In this way our approach can get better action plan after each decision cycle.

This paper is organized as follows: section 2 talks about related work and analyze their weakness; section 3 provides our novel HTN planning algorithm. At last the experiment result is provided and analyzed particularly.

II. RELATED WORK

Since HTN planning was proposed in 1975 [11], it has been applied in many domains. A few researchers have attempt

to apply HTN planner to RTS games because it can dramatically speed up searching by reducing searching space and provide good explanations to human players. Laagland [12] applied HTN planner in Spring which is an open source RTS game. Laagland dividing the game planning into three levels: strategic, tactic and action. The HTN planner was applied into strategic level. Furthermore, Laagland analyzed the advantages and disadvantages of HTN planners. Naveed et.al attempted to reduce the pathfinding space of ORTS which is a kind of RTS game by employing HTN planner in 2010.

Though the above studies have got some achievement in RTS games, all of them have not considered the features of RTS games. The above approaches change the planning problem of RTS games into the static, un-adversarial problem and they do not consider the actions are durative. In order to overcome the weaknesses, Ontañón and Buro proposed the AHTN algorithm and applied it into μRTS [10]. AHTN addressed the large state space by combining HTN planning approach with MCTS. And it also considered the simultaneous and durative actions. Although AHTN has achieved good performance compared with other search algorithms, it still suffers some weaknesses: 1) it cannot repair failed tasks; 2) its performance drops in partial observation environment. In order to overcome these weaknesses, Sun et al. designed AHTN-R algorithm by introducing a novel failed task repair approach [13]. Yang et al. proposed the single belief generated method based on fuzzy inference to increase the performance in partial environment [14]. However, the above algorithms neither consider the opponent strategies or the time constraint of generating subgame tree. All of them take their own HTN planner and domain knowledge as the opponent's planner and knowledge which cannot construct real adversarial environment.

III. Approach

This section presents the AHTNCO planner which is modified based on the AHTN planner [10]. The AHTN planner assumes there are two players: *max* and *min* and it searches the game tree in a similar fashion as min-max game search by using the HTN planner to expand the nodes. But the AHTN algorithm has some differences with stand min-max game search by considering durative, concurrent and simultaneous actions. And the max or min node will not be expanded alternately. It will not expand the other type of nodes until the action which can be applied on the current node is found out.

Fig.1 shows part of the game tree generated by AHTN as instance. The rectangle nodes in Fig.1 stand for max node and the rounded rectangle ones stand for min nodes. As shown in Fig.1, Max1 can be expanded into Max2 or Max3 by decomposing the current complex task. However, there is no executable primitive task among the subtasks generated. It means the game state cannot be changed at the Max2 node. Furthermore, the influence on the opponent decision cannot be observed. The AHTN planner will go on decomposing the complex task at Max2 and get the generation max nodes such as Max4 and Max5. When Max4 has been expanded and get an available primitive task, the AHTN planner will execute this task in the simulation environment and start to expand the min nodes such as Min3 or Min4 by using the same strategy.

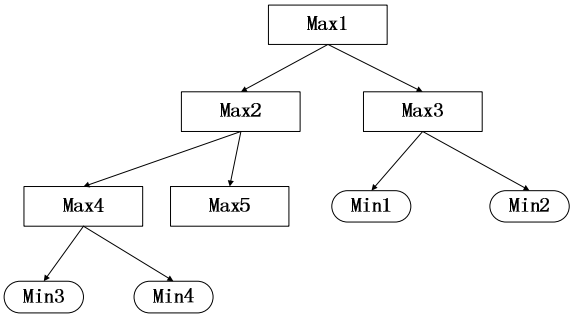

Fig. 1 Example of game tree generated by AHTN

The AHTN planner has two shortages according to above analysis: 1) it employs the same HTN planner to expand the max nodes and min ones; 2) it does not consider the time constraint when generating subgame tree. The first shortage may lead the AHTN planner cannot find the best plan because the real opponent strategy often is different with the HTN planner in AHTN. Sometimes it may lead to disaster result like our experiments show. The second shortage can lead no plan will be find out as the AHTN algorithm spends too much time on one branch.

In order to overcome the above two shortages of the AHTN planner, we proposed the AHTNCO algorithm. In AHTNCO algorithm, the opponent strategy is employed to simulated the adversarial player's behavior. During planning, AHTNCO employs the HTN planner to generate the max nodes as AHTN until AHTNCO get a primitive task and prepare to generate the min nodes of the game tree. AHTNCO will employ the opponent strategy to generate the min nodes instead of employing the same HTN planner as AHTN. AHTNCO will continue generating min nodes until it finds out the first available action by employing the opponent strategy. Then AHTNCO will execute the action and turns to generate the max nodes. In this way, AHTNCO can provide a better plan for long term decision because it considers the adversarial player's reaction.

As each cycle decision time is limited and the player can make many decisions in RTS games, an available plan is acceptable and it is much better than do nothing. In order to get an available plan, AHTNCO introduces the time constraint during generating game tree nodes in case of spending too much decision time on one branch. During AHTNCO searching, the time spent on each branch cannot be longer than the given time limitation. Once the time limitation is reached, AHTNCO will stop continuing searching current branch and choice another one on the same level. The time limitations on the game tree levels are different. The time limitation of each upper level should larger than the limitation of its next level. However, too little time limitation is also not suitable as it may lead the planner has no enough time to reach a suitable depth

of the game tree. In this paper, $t_{\max}/d$ is used as time limitation example. $t_{\max}$ is the cycle decision time and d is the current depth of the game tree. In this way, AHTNCO can get a balance between depth and width during searching the game tree. The algorithm of expanding min nodes is shown as Alg.1. And the algorithm of expanding max nodes is similar.

Alg.1 AHTNCOMIN

Input : $s, t_{\max}, CP_{\max}, CP_{\min}, \alpha, \beta, d$

1. **If** $GameState(s) = \perp$ **then**
2. **Return** $(CP_{\max}, CP_{\min}, e(s), \alpha, \beta)$
3. **End if**
4. $(Actions, ChoicePoint) = GenerateTreeNode(s)$
5. **If** $Actions \neq \varnothing$ **then**
6. $a = Action(s)$
7. **Return** $AHTNCOMAX(\gamma(s,a), t_{\max}, CP_{\max}, CP_{\min}, \alpha, \beta, d+1)$
8. **End if**
9. $CP^*_{\max} = \perp, CP^*_{\min} = \perp, v^* = -\infty$
10. **For all** $CP_{\min} \in ChoicePoint$ **do**
11. **If** $time \geq t_{\max}/d$ **then**
12. **Break**
13. **End if**
14. $v' = AHTNCOMIN(s, t_{\max}, CP_{\max}, CP_{\min}, \alpha, \beta, d+1)$
15. **If** $v^* < v'$ **then**
16. $v^* = v', CP^*_{\max} = CP_{\max}, CP^*_{\min} = CP_{\min}$
17. **End if**
18. **End for**
19. **If** $\beta \geq v^*$ **then**
20. **Return** $(CP^*_{\max}, CP^*_{\min}, v^*, \alpha, \beta)$
21. **End if**
22. $\alpha = \max(\alpha, v^*)$
23. **Return** $(CP^*_{\max}, CP^*_{\min}, v^*, \alpha, \beta)$

The input of Alg.1 is game state s ,time limitation $t_{\max}$,game tree nodes $CP_{\max}$, $CP_{\min}$ and the tree depth d . The initial value of α is $-\infty$ and β is ∞ . Line 1-3 shows the stop condition of searching. Line 4 means the algorithm tries to generate all nodes of the next level of the game tree. And the generated nodes can be divided into two parts: actions which can change the game state and choice point which can be continued decomposing. If there is an available action, the algorithm will execute it and start to searching the max node as shown in Line 5-8. If there is no available action, AHTNCOMIN will try to expand each node as shown Line 9-18. Line 11-13 means the searching must be stopped if the time is longer than $t_{\max}/d$. Line 14 shows the algorithm continue expanding the next level min nodes. It can avoid spending too much time on one branch which leads to that there is no time to search other branches of the game tree.

At the beginning of each decision cycle, AHTNCO will generate the game tree from the max nodes by employing the HTN planner until it find an available action. Then the algorithm will generate the min nodes by employing Alg.1 algorithm until an action can be applied on the game state. The game tree's generation will last until time is longer than the limitation or the game finish condition is satisfied.

IV. Experiment and Analysis

A. Experimental Environment and Settings

μRTS software [15] is an open source RTS game which has been used to evaluate various AI algorithms and been used in AI competition in 2017 [16]. A screenshot of the μRTS game is shown in Fig2.

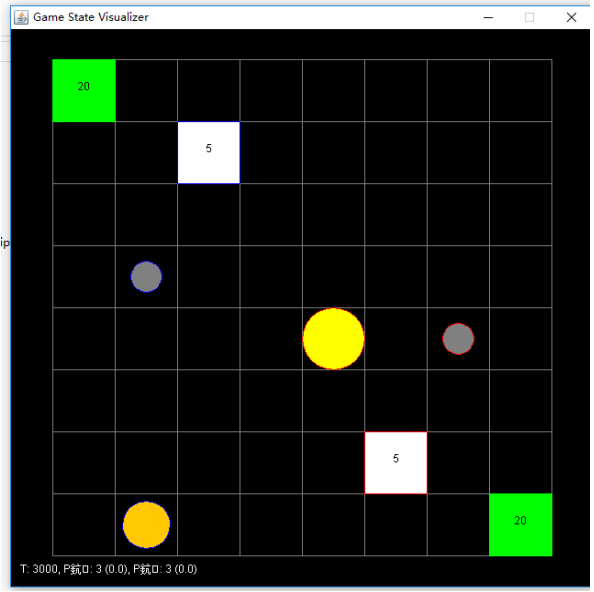

Fig 2. A screenshot of the μRTS game environment

The two players are distinguished according to the blue and red outline colors. Each one has the same types of units. The green squares are limited resources. The white squares are bases which can produce new units. The gray circles are workers, the yellow circles are heavy attackers, and the orange circles are light attackers. The workers can attack enemies, build bases, and harvest and transport resources. Both heavy and light attackers have larger hit and health points than workers, but they cannot harvest or transport resources. There are six types of actions for units: move into any empty space, attack any enemy in range, harvest resources from resource units, return resources to bases, produce new units and remain idle. Though μRTS is simple, it still has the features of RTS games and is complex enough.

The following parameters were employed in the testing.

• CPU time: The limited amount of CPU time allowed for an AI player per game frame. In our experiments, we employ different CPU time setting from 100ms to 300ms to test the performances of the algorithms.

• Playout policy: The Random strategy is employed as the playout policy.

• Playout time: The maximum running time of a playout. The playout time is 100 ms.

• Pathfinding algorithm: the A* pathfinding algorithm is employed to obtain the best path from a current location to a destination location.

• Evaluation function: The evaluation function is a variant of LTD2 [16].

• Maximum game time: The maximum game time is set as 3000 cycles.

• Maps: The maps used in our experiments are M1 (8×8 tiles) and M2 (16×16 tiles).

To evaluate our algorithm, we compare three kinds of algorithms: original HTN planner, AHTN and AHTNCO. Each algorithm will employ two kinds of HTNs as its domain knowledge. The two HTNs are as follows:

1) Flexible-Portfolio: contains the 12 operators, 49 methods and 19 types of tasks.
2) Portfolio: contains the 12 operators as Flexible-Portfolio, but has 76 methods and 28 types of tasks.

We construct 5 experiments in one map for one CPU time. The experiments are: *HTN-FP vs HTN-P, HTN-FP vs AHTNCO-P-FP*, *HTN-P vs AHTNCO-FP-P, AHTN-P vs HTN-FP, AHTN-FP vs HTN-P*. The algorithm-FP means it employs the Flexible-Portfolio HTN as the domain knowledge. AHTNCO-P-FP means the Portfolio HTN is employed as own domain knowledge while the Flexible-Portfolio HTN is used as its opponent's domain knowledge.

Each experiment will play 200 competitions in each map for three CPU time (2*3*200*5=6000 competitions in total). The score of each algorithm is computed as follows: the winner is awarded 1 point, the loser got 0 point and both algorithms are awarded 0.5 points in the event of a tie. Both of the two AI players in all competitions began with a single base, the same resource, and a single worker. And the game environment is set total observation.

B. Results and Analysis

In this section, we compare the performance of AHTNCO with other algorithms. The score of each algorithm is shown in Fig.3-14. The same color in one figure means the two algorithms are in the same competitions.

Fig.3-5 present the comparisons of the scores obtained from AHTNCO and HTN algorithms in M1 for CPU time is 100ms, 200ms and 300ms. In all of the three figures, HTN-P performs worse than HTN-FP by losing 143.5 points, 130 points and 128 points. While AHTNCO-P-FP wins 128 points, 131 points and 135.5 points when it competes with HTN-FP. The points of AHTNCO-P-FP raises about 108.9%, 87.1% and 88.2% than HTN-P when each of them competes with HTN-FP. The points of AHTNCO-FP-P also raise about 7.6%, 15.4% and 12.1% than HTN-FP when they compete with HTN-P. The above results show that AHTNCO performs much better than the HTN planner in all competitions even though its domain knowledge is worse than the opponent's. The similar conclusion also can be got from Fig.6-8. AHTNCO performs better than HTN planner because it considers its opponent strategy. During AHTNCO game tree searching, it will infer the opponent's action and take the most suitable action to generate its own plan.

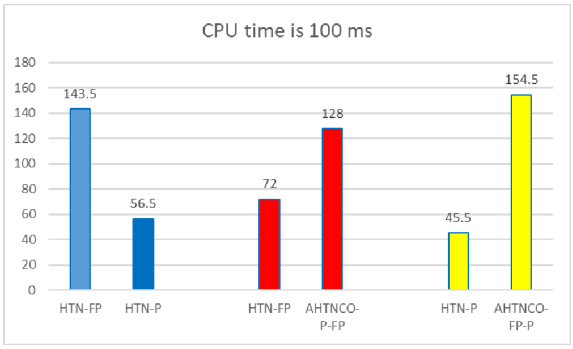

Fig 3. The scores of HTN-FP vs HTN-P, HTN-FP vs AHTNCO-P-FP and HTN-P vs AHTNCO-FP-P in M1 for CPU time is 100 ms.

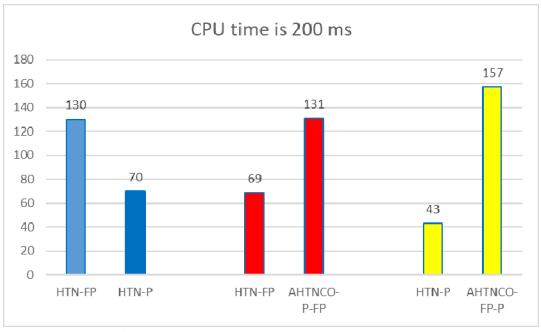

Fig 4. The scores of HTN-FP vs HTN-P, HTN-FP vs AHTNCO-P-FP and HTN-P vs AHTNCO-FP-P in M1 for CPU time is 200 ms.

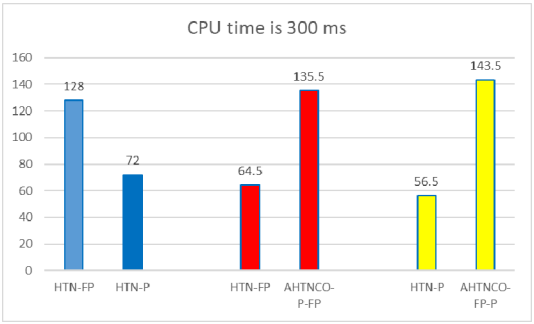

Fig 5. The scores of HTN-FP vs HTN-P, HTN-FP vs AHTNCO-P-FP and HTN-P vs AHTNCO-FP-P in M1 for CPU time is 300 ms.

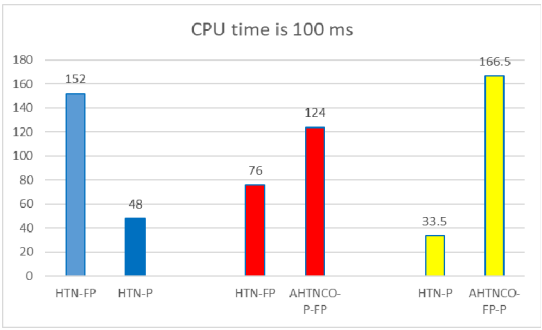

Fig 6. The scores of HTN-FP vs HTN-P, HTN-FP vs AHTNCO-P-FP and HTN-P vs AHTNCO-FP-P in M2 for CPU time is 100 ms.

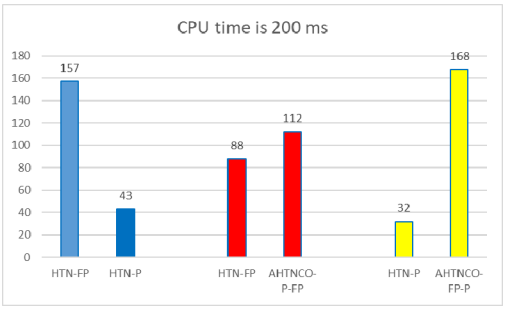

Fig 7. The scores of HTN-FP vs HTN-P, HTN-FP vs AHTNCO-P-FP and HTN-P vs AHTNCO-FP-P in M2 for CPU time is 200 ms.

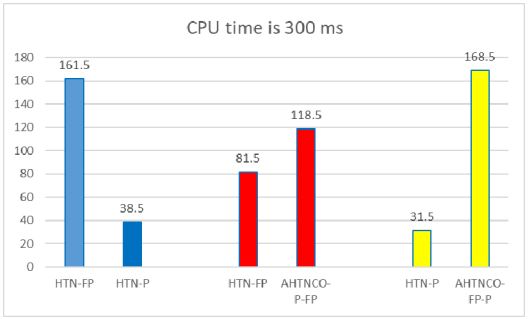

Fig 8. The scores of HTN-FP vs HTN-P, HTN-FP vs AHTNCO-P-FP and HTN-P vs AHTNCO-FP-P in M2 for CPU time is 300 ms.

Fig.9-11 present the comparisons of the scores obtained from AHTN, AHTNCO and HTN algorithms in M1 for CPU time is 100ms, 200ms and 300ms. The points of AHTNCO-P-FP raises about 250%, 106.8% and 113.8% than AHTN-P when each of them competes with HTN-FP. The points of AHTNCO-FP-P also raise about 66%, 1.9% and 2.8% than AHTN-FP when they compete with HTN-P. The above results show that AHTNCO performs better than AHTN especially when decision time is limited in very short time such as 100ms. When decision time is longer, the advantage of AHTNCO-FP-P will be reduced. That is because the Flexible-Portfolio HTN can provide better domain knowledge than the Portfolio HTN. And AHTN-FP could expense much time to search more game tree node to get better plan. It will lead AHTN-FP can get a very high score when it competes with HTN-P for longer decision time. The similar conclusion also can be got from Fig.11-14. AHTNCO performs better than AHTN because the AHTN algorithm does not consider the opponent strategy. During its planning, it takes itself as the opponent which cannot simulate the real opponent strategy exactly. Moreover, AHTN also did not consider the time constraint and spent too much time on one branch which led it could not find a better plan. Therefor the planning result cannot be the best for the real adversarial environment. But the AHTNCO algorithm will employ the opponent's strategy during planning to find the best plan and it considered the searching time limitation for sub game tree in case of spending too much time on one branch.

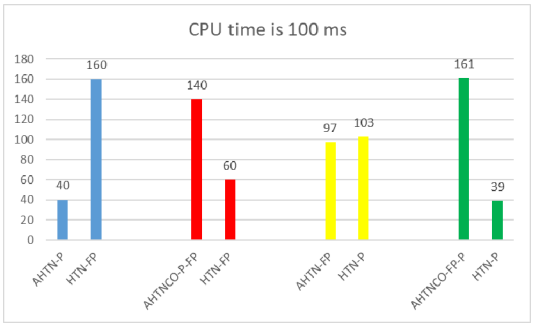

Fig 9. The scores of AHTN-P vs HTN-FP, AHTNCO-P-FP vs HTN-FP, HTN-FP vs HTN-P and AHTNCO-FP-P vs HTN-P in M1 for CPU time is 100 ms.

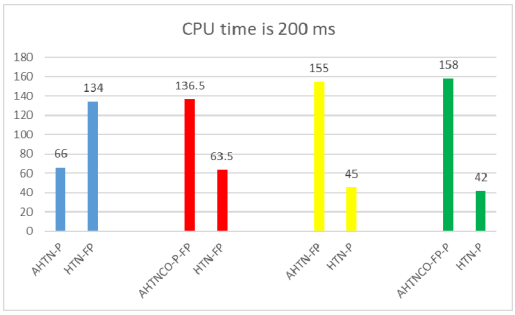

Fig 10. The scores of AHTN-P vs HTN-FP, AHTNCO-P-FP vs HTN-FP, AHTN-FP vs HTN-P and AHTNCO-FP-P vs HTN-P in M1 for CPU time is 200 ms.

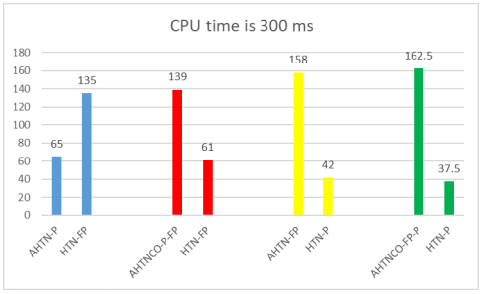

Fig 11. The scores of AHTN-P vs HTN-FP, AHTNCO-P-FP vs HTN-FP, AHTN-FP vs HTN-P and AHTNCO-FP-P vs HTN-P in M1 for CPU time is 300 ms.

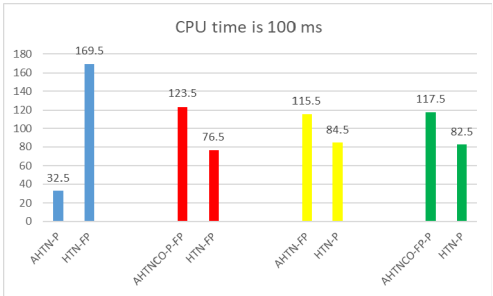

Fig 12. The scores of AHTN-P vs HTN-FP, AHTNCO-P-FP vs HTN-FP, AHTN-FP vs HTN-P and AHTNCO-FP-P vs HTN-P in M2 for CPU time is 100 ms.

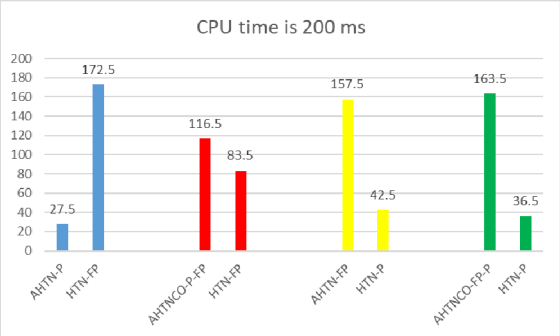

Fig 13. The scores of AHTN-P vs HTN-FP, AHTNCO-P-FP vs HTN-FP, HTN-FP vs HTN-P and AHTNCO-FP-P vs HTN-P in M2 for CPU time is 200 ms.

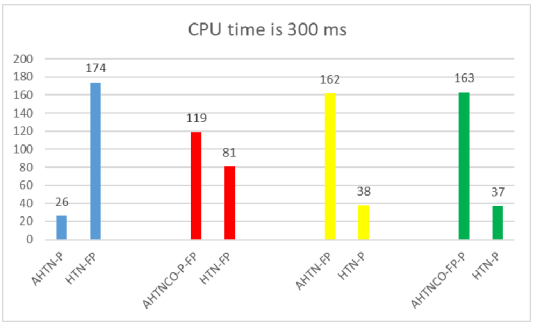

Fig 14. The scores of AHTN-P vs HTN-FP, AHTNCO-P-FP vs HTN-FP, HTN-FP vs HTN-P and AHTNCO-FP-P vs HTN-P in M2 for CPU time is 300 ms.

V. Conclusion

The AHTN algorithm has addressed the problem of large state space in RTS games by combining HTN planning with game tree search. However, it neither not consider the opponent strategy nor the time constraint of generating subgame tree. Therefore, this paper proposed AHTNCO algorithm which will employ the opponent strategy to guide the HTN planning and consider the time limitation of generating each subgame tree during planning. The experiment results show that AHTNCO performs better than AHTN and original HTN planner when they compete with the same opponent.

In the future work, the automatic identification of the opponent strategy according to the observation can provide great assistance during the RTS competitions. In addition, we note neither AHTN nor AHTNCO consider indetermination and partial observation, which frequently appear in RTS games. This may affect the algorithms greatly. Therefore, an improved AHTNCO algorithm under uncertainty may enhance its performance in RTS games.

Acknowledgment

The work described in this paper is sponsored by the National Society Science Foundation of China under Grant No. 2019-SKJJ-C-073.

References

[1] Buro M. "Real-time strategy games: A new AI research challenge," In Proceedings of IJCAI, pp:1534-1535, 2003.
[2] Glen Robertson, Ian Watson. "A Review of Real-Time Strategy Game AI," Artificial Intelligence, Vol.35, No.4, pp:75-104, 2014.
[3] Knuth D.E., and Ronald W.M. "An analysis of alpha-beta pruning," Artificial Intelligence, Vol.6, No.4, pp:293-326, 1975.
[4] Ontanón, Santiago, et al. "A survey of real-time strategy game AI research and competition in starcraft," IEEE Transactions on Computational Intelligence and AI in games, Vol.5, No.4, pp:293-311, 2013.
[5] Chung. M., Buro, M.and Schaeffer. J. "Monte carlo planning in RTS games," In Proceedings of CIG, pp:1-8, 2005.
[6] Balla R.K., and Fern A. "UCT for tactical assault planning in real-time strategy games," In Proceedings of IJCAI, pp:40–45, 2009
[7] Churchill David. "Heuristic Search Techniques for Real-Time Strategy Games," University of Alberta, 2016.
[8] Muñoz-Avila, Hector, and David Aha. "On the role of explanation for hierarchical case-based planning in real-time strategy games," In Proceedings of ECCBR-04 Workshop on Explanations in CBR, pp:1-10, 2004.
[9] Menif, Alexandre, Eric Jacopin, and Tristan Cazenave. "SHPE: HTN Planning for Video Games," Third Workshop on Computer Games, pp:119-132, 2014.
[10] Ontanón, Santiago, and Michael Buro. "Adversarial hierarchical-task network planning for complex real-time games," In Proceedings of IJCAI, pp:1652-1658, 2015.
[11] Sacerdoti E.D. "The nonlinear nature of plans," STANFORD RESEARCH INST MENLO PARK CA, 1975.
[12] Laagland, Jasper. "A HTN planner for a real-time strategy game," Unpublished manuscript. (hmi. ewi. utwente. nl/verslagen/capita-selecta/CS-Laagland-Jasper. pdf), 2008.
[13] Sun, Lin, Peng Jiao, and Kai. Xu. "Modified Adversarial Hierarchical Task Network Planning in Real-Time Strategy Games," Applied Sciences. Vol.7, No.9, pp:872-897, 2017.
[14] Weilong Yang, Xu Xie, Yong Peng. "Fuzzy Theory Based Single Belief State Generation for Partially Observable Real-time Strategy Games," IEEE ACCESS, Vol.7, pp:79320-79330, 2019.
[15] microRTS. https://github.com/santiontanon/microrts, 2019.
[16] Ontanón, Santiago, Nicolas A. Barriga etc. al. "The First MicroRTS Artificial Intelligence Competition," Aritificial Intelligence, Vol.39, No.1, pp:75-83, 2018.
[17] Churchill David and Buro Michael. "Portfolio greedy search and simulation for large scale combat in StarCraft," In Proceedings of CIG, pp:1-8, 2013.