Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A.A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

HISTORY-BASED SINGLE BELIEF STATE GENERATION FOR PARTIALLY OBSERVABLE REAL-TIME STRATEGY GAMES

Weilong Yang
Yong Peng
Quanjun Yin

Department of Modeling and Simulation
National University of Defense Technology
Sanyi Avenue, 137
Changsha, 410000, CHINA

Lin Sun

National Defense University
Taiping Avenue, 23
Beijing,100000,CHINA

ABSTRACT

Researches of AI planning in Real-Time Strategy (RTS) games have been widely applied to human behavior modeling and war simulation. Due to the fog-of-war, planning in RTS games need to be implemented under partially observable environment, which poses a big challenge for researchers. This paper focuses on extending Hierarchical Task Network (HTN) Planning in partially observable environment, and proposes a partially observable adversarial hierarchical task network planning with repairing algorithm named PO-AHTNR. By adding sensing action into HTN domain knowledge, a reconnaissance strategy and a history-based single belief state generation method are presented to obtain the best action. In order to verify the proposed algorithm, an empirical study based on μRTS game is carried out, and the performance of modified algorithm is compared to that of AHTNR and other state-of-the-art search algorithms developed for RTS games.

1 INTRODUCTION

Real-Time Strategy (RTS) games are popular real-time war simulation games in which players instruct units to gather resources, build structures, destroy opponent's buildings to win the game. As typical agent-based game, RTS games pose a huge challenge for AI researchers due to the large state space, limited decision time and dynamic adversarial environment involved. Especially for the commander agent modeling, which refers to the global planner research. And AI planning becomes an important research area for real-time adversarial planning and non-deterministic decision (Buro 2004).

Previous works in this area such as Min-Max algorithm (Stanescu et al. 2014), Alpha-Beta Considering Duration algorithm (Churchill and Buro 2012), Monte Carlo Tree Search (Lanctot 2009; Shleyfman et al. 2014) and Hierarchical Task Network (HTN) algorithm (Ontañón and Buro 2015) have been widely used in agent technology and war simulation. Among them, the HTN algorithm achieves satisfying result in solving complex problem since its planning process is similar to human decision process and its hierarchical network is more sufficient to handle the large state space than other planning algorithms (Ghallab et al. 2004). However, previous HTN algorithms are most conducted in fully observable environment, few work has been accomplished in the partially observable environment, which still remain a substantial challenge.

Comparing with fully observable environment, planning in partially observable environment includes following characteristics (Ontañón 2013): (1) The initial state is small; (2) The environment information under the fog-of-war is not available even though the area has been explored before; (3) The environment information needs to be collected and fused. Due to the rapid changes and short decision-making time in RTS games, the application of traditional Markov decision-making process to HTN planning will lead to

numbers of branches and increase the difficulty in decision-making. Seeking a simple and efficient plan method for partially observable environment is an important direction of automatic planning research.

We address this problem by integrating single belief state generation with adversarial hierarchical task network repairing algorithm (AHTNR), using *determinization* theory to incorporate observed state and history state into belief state. In the remainder of this paper, after discussing related work, we first present the definition of PO-AHTNR description and the architecture of PO-AHTNR algorithm. Then we propose reconnaissance strategy based on sensing action and history-based single belief state generation method, followed by extensions to apply it to μRTS games, a minimalistic RTS game used for planning algorithm evaluating.

2 RELATED WORK

HTN is an automatic plan method which decomposes complicated task into sub-tasks until all the sub-tasks can be executed, first proposed by Sacerdoti (1975). Rather than exploring the entire combination of possible actions, HTN planning can guide the search direction based on domain knowledge to reduce the search space. Since the planning process is similar to human reasoning process, HTN planning is widely used to solve complex problems. Combining HTN planning with search methods has been demonstrated to speed up planning dramatically, and this has led to the application of HTN methods to RTS games (Kelly and Botea 2007; Humphreys 2013).

Muñoz-Avila and Aha (2004) first attempted to employ HTN planner to provide human players with explanations for the reasons which lead to current states or events. After that, Laagland (2014) presented the design, implementation, and evaluation process of HTN planner in an open source RTS game denoted as Spring, and further summarized the advantages and disadvantages of HTN planning in RTS games. Naveed et al. (2010) employed HTN planners to reduce the size of path-finding search space in RTS games, and tested their algorithm in games developed using ORTS. Ontañón and Buro (2015) combined the hierarchical task network planning approach with game tree search denoted as adversarial HTN (AHTN) algorithm. Most recently, Sun et al. (2017) proposed a modified AHTN planning algorithm with failed task repairing component denoted as AHTN-R, making planning process dynamic and repairable. Of the above studies, we note that no HTN researches for RTS planning have considered extending HTN algorithms to solve the partially observable RTS games.

For traditional automated HTN planning domains, studies have addressed partially observable problem. Nau et al. (2003) proposed Cond-SHOP2 algorithm to solve the problem of planning in partially observable system, which modified the SHOP2 planner (Nau et al. 1999) by integrating forward-chaining planners. It represented states in the form of state variables. During the algorithm execution, Cond-SHOP2 decomposed the task network and uses the execution actions to transfer part of the states, and branched the possible partially states generated by the sensing actions.

In RTS games, *determinization* is one of popular techniques used for solving imperfect information, which sampled states from an information set and analyzed the corresponding games of perfect information (Schwalbe and Walker 2001). Since the game tree of imperfect information games tended to be complex even for simple games, researchers attempted to use approximation algorithms to simplify game state. Zinkevich et al. (2007) presented Counterfactual Regret Minimization to sample all the different states. Lanctot et al. (2009) avoided sampling all the states by using Monte Carlo sampling. Uriarte and Ontañón (2017) investigated sampling a single belief state consistent with a perfect memory of all the past observations in the current games, and proposed a single belief state generation for partially observable RTS games.

In this paper, taking the advantage of sensing action and single believe-state generation strategy, we add the sensing action and information set to HTN planning algorithm, specifically focus on the simplification of partially observable environment in RTS games. By combining the sensing strategy and history-based single belief state generation, a modified AHTNR algorithm for partially observable RTS games is proposed.

3 PO-AHTNR ALGORITHM

In this section, an extended AHTN description is proposed to describe partially observable RTS games. Then the overall framework of the PO-AHTNR algorithm is provided, and the state reasoning component is explained. Finally, the reconnaissance strategy and belief state updating strategy is proposed.

3.1 Definition of PO-AHTNR Description

According to the partially observable context in RTS games, we modify the representation of AHTN planning model (Ontañón and Buro 2015) and add observation state and history state to record the partially observable features, which integrated into the current belief state. We use history information and domain knowledge to overcome the insufficiency of information caused by the partial observation.. The adversarial HTN planning model in partially observable environment can be expressed as a multivariate group.

$$\mathrm{PO-AHTNR} =< P, Z, B, R, TN_{max}, TN_{min}, M_{max}, M_{min}, O_{max}, O_{min}, S, \gamma >$$

- P is the current observed state set.
- Z is the history state set, consisting of a period of game times.
- $\mathrm{B} = \mathrm{P} \cup \mathrm{Z}$ is the belief state set.
- $R(p) \rightarrow z$ is the updating strategy, planner converts current observed state P to belief state.
- TN_{max}, TN_{min} is the current task network. It is a tree whose nodes are tasks, methods, or phases.
- M_{max}, M_{min} is the set of task decomposition methods of player and opponent player. Each method $\mathrm{m} \in \mathrm{M}$ can be applied to decompose a task into a set of subtasks.
- O_{max}, O_{min} is the set of operators for player and opponent. Each operator $\mathrm{o} \in \mathrm{O}$ is an execution of a primitive task.
- S is the set of world state, consisting of all information that relevant to the planning process.
- γ is the state transform function. Given $\mathrm{s} \in \mathrm{S}$, $\gamma(\mathrm{s}, \mathrm{o})$ defines the transition of the state when an action is executed.

3.2 Architecture of PO-AHTNR Algorithm

The overall framework of PO-AHTNR algorithm is illustrated in Figure 1, consisting of four components: state reasoning component, plan generation component, execution component, and task repairing component. In each planning cycle, player for which we can treat as commander agent, execute the planning process following these four components.

Comparing with AHTNR algorithm framework, state reasoning component is added to form the belief state. For AHTNR algorithm, the game state is fully observable, so the AHTN algorithm can generate the original plan. But in partially observable environment, we cannot obtain the whole world state, so we use the state reasoning component to obtain. In each decision cycle, state reasoning component executes reasoning when there are units without task. First, sensing strategy is used to obtain more information about the environment. After that, by analyzing the belief state, state reasoning component combines the current observed information with the history information to construct belief state, which is the input of plan generation component.

Plan generation component works out the unit actions for execution component. AHTN algorithm generates the best plan to plan execution component and provides a list of alternative plans to the task repair component. Execution component executes the plan and sends failed tasks to task repairing component. The task repairing component selects suitable alternative plan, repairs the failed tasks and returns the repaired tasks to plan generation component.

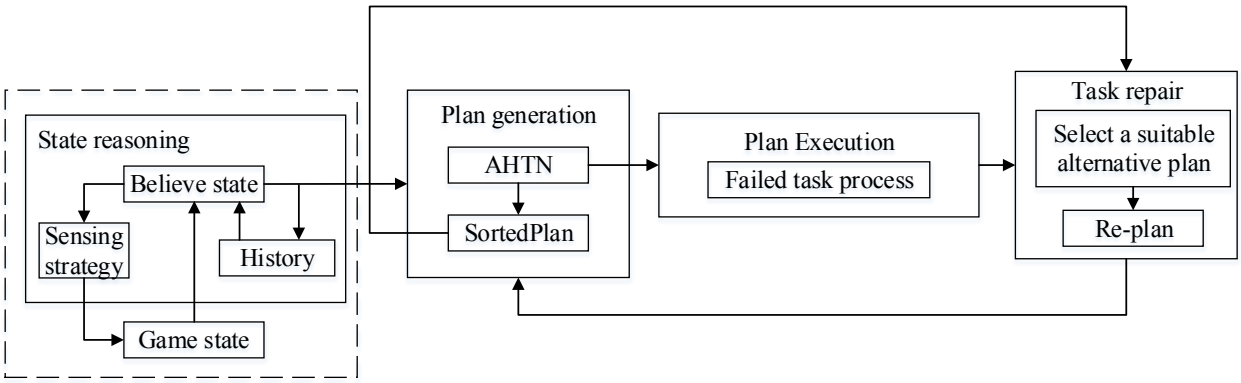

Figure 1: Overall framework of the PO-AHTNR algorithm.

3.3 Sensing Action and Single Belief State Generation

Comparing with fully observable environment, the search space in partially observable environment becomes larger since the value of each observation is no longer a single state but a state set. In this paper, we use the *determinization* theory to improve plan efficiency, including pre-control strategy and post-control strategy. Pre-control strategy use the initiative executing sensing action to sensor the game environment. Post-control strategy use information reasoning strategy to obtain the unobserved environment information.

3.3.1 Sensing Action

In automatic planning, sensing action is an action to sense environment and decrease non-deterministic (Ghallab et al. 2004). Unlike other actions, sensing action is designed on current state to get more information about the initial state of next planning turn, which only affects the validity of the planning algorithm and does not directly determine the success of games. Sensing action can be executed by following categories of actions:

Instantaneous sensing action are actions which can immediately perceive the information in part of game area, such as some specific skills of hero units. The main constraint of instantaneous sensing actions is the resource consumed by executing the action. Since it is not general exist in RTS games, this article does few research on it. ***Delayed sensing action*** are sensing actions that need executing time, which means this kind of sensing actions cannot immediately perceive the information of current game state, this article mainly studies about this kind of action.

Delayed sensing action can be classified in two kind: fixed sensing actions and mobile sensing actions. For fixed sensing actions, units can expand its sensor radius by switching the unit state. For example, in StarCraft, Zerg insects can convert from ground state to flight state, which can expand its sensing radius without position moving. For mobile sensing actions, units observe the environment by moving to unknown area and get the information among its sensor radius. We add sensing action into the domain knowledge of AHTNR, assuming that each unit has a sensor to get information around, which classifying sensing action into three primary actions: turn on sensor, turn off sensor, expand sensor radius. When the sensor is turned on, the environment information within sensor radius can be obtained. When the sensor is turned off, the surrounding environment information cannot be obtained. When the sensor radius expands, the sensing area expands. In RTS games, it is usually default that the sensor is on. Therefore, the mobile sensing action can be simplified into planning of move actions. A mixed evaluation function for state s is used to calculate whether executing the sensing action or not: if $E(s) > threshold$, execute the sensing action; if $E(s) \leq threshold$, not execute the sensing action. The $threshold$ is the threshold value set according by different player preference.

$$\mathrm{E(s)} = \frac{\Delta\mathrm{Info}}{\mathrm{Cost}}, \Delta\mathrm{Info} = \sum \Delta H(s) * w(s), \mathrm{Cost} = C(sensing\_action) * \mathrm{occupy(unit)} \quad (1)$$

The evaluation function $\mathrm{E(s)}$ is the ratio of information entropy to consumption value. $\Delta\mathrm{Info}$ is the dispersion of information entropy caused by executing sensing action, $Cost$ is the total resources consumed for executing the sensing action. $\Delta H(s)$ is the uncertainty of the game state, which can simply as the increment of sensing area, and $w(s)$ is the importance of information, different type of units have different importance. $C(sensing\_action)$ is the cost of executing one sensing action, and $occupy(unit)$ is the summation of units which execute sensing action.

For example, in game state s, player can execute a sensing action or not. Using Equation 1, we can calculate the cost and the $\mathrm{E(s)}$. The cost for executing sensing action $C(sensing\_action)$ is 5 resources, and will occupy a worker to execute $\mathrm{occupy(unit)} = 1$, the Cost is 5. By the increment of observed grids is 3 by executing the sensing action, and no new unit is observed, so $\Delta H(s) = 3$, $w(s) = 1$, $\Delta\mathrm{Info} = 3$. $\mathrm{E(s)}$=5/3. Assuming the $threshold$=1, then in this state, $\mathrm{E(s)} > threshold$, the sensing action will be executed.

3.3.2 Single Belief State Generation

By using sensing action, the partially observable environment has been minimized but still cannot be fully observed. Since the decision time is short, using probabilistic method to plan will bring a huge search space. Single belief state generation (Uriarte and Ontañón 2017) is a method to get valuable information from the observed state, consisting of three strategies: target tracking strategy, incomplete memory strategy, complete memory strategy. These strategies make use of the observed state to generate the belief state. For example, when dealing with unobserved units, one of the most recent invisible positions is randomly selected as the inferred position. These generation strategy can quickly generate the belief state but the units' history information and characteristics of adversarial antagonism are not taken into account.

A history-based single belief state generation method is proposed to obtain the unobserved game state. First, information sets of units' positions are created, consisting of history state set, current state set and belief set. Among them, history state set records the history information of each unit, which has multi layers to record periods of history state, current state set records the current information of each unit, and belief state set records the current belief information of each unit which reasoned from history state set and current state set. Then, at each planning process the information sets are updated as follow: if the unit's state is observable in current state set, record it in belief state set; if the unit's state is not observable in current state set but is observable in history state set, then compare different phase history sets' records about the unit state, and choose the highest frequency position or the nearest reachable position to the observation area as the unit's belief state. Lastly, the belief state set is used for planning and decision.

The process of single belief state generating algorithm is as follow: Line 1-2 show that, player obtains the observation set of all units and saves it as this observation record. Line 3-4 show that the observed states of all the opponents' units are added or removed from the history state record. Line 5-14 show the principle of update belief state, which is about the belief state set's generation from the history state set and observed state set. Line 15-17 show the output of algorithm.

```
class Single_Belief_State_Generation {
1      While exist unit to be planned
2           Get the game state denoted to observed_state
3           Add the observed_state to history_state_set
4           Remove the invalidate history
5           For unit ∈ observed_state_set
6               If unit belongs to opponent then
```

```
7                  Get unit state applied to fix_unit_set
8              End If
9              If unit ∈ history_state_sets then
10                  Get belief_state by o_s and h_s
11             Else
12                  Get belief_state by r_s and o_s
13             End If
14       End For
15       Set belief_state by reasoning
16    End While
17  Return belief_state
};
```

4 EXPERIMENTAL RESULTS

In order to verify the proposed algorithm, an empirical study based on μRTS game is carried out, and the performance of modified algorithm is compared to that of AHTNR and to the performances of other state-of-the-art search algorithms developed for RTS games on μRTS games, which has been used in the past to evaluate various algorithms applied in RTS games.

4.1 Experimental Environment and Setting

We evaluate the performance of PO-AHTNR using the free-software μRTS (https://githubs.com/santiontanon/microrts), which has been used by several researchers to validate new algorithms for RTS games. Figure 2 shows a screenshot of a μRTS game, in which two players compete to destroy the opponent's units.

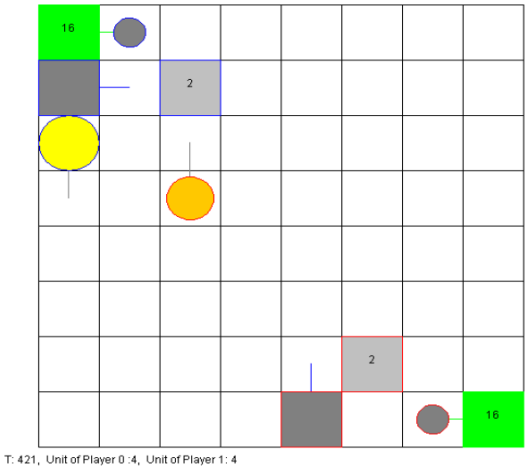

Figure 2: A screenshot of the μRTS game environment. The two players are distinguished according to the blue and red outline colors. The green squares are resources. The light gray squares are bases. The dark gray squares are barracks. The gray circles are workers, the yellow circles are heavy attackers, and the orange circles are light attackers. The number in base, resources or workers refer to the number of resource carried by them.

To test PO-AHTNR algorithm, we craft two different HTN domain knowledge: Low Level domain knowledge and flexible domain knowledge. The operators are primitive tasks, the tasks are compound tasks, the methods are approaches to achieve the tasks. **Low Level** domain knowledge contains 10 operators

(primitive tasks) and 9 methods for 3 types of tasks (destroy-player, destroy-player-internal, unit-order). Each method is a **Flexible** domain knowledge contains 12 operators, but provides 49 methods for 9 types of tasks (destroy-player, destroy-player-rush, destroy-player-rush-reserved-unit, destroy-player-rush-reserved-unit-rounds, worker-rush-unit-behavior, worker-rush-unit-behavior, destroy-player-light-rush, destroy-player-light-rush-reserved-unit, light-rush-unit-behavior).

The maps used in our experiments are three: M1 (8 × 8 tiles), M2 (12 × 12 tiles), and M3 (16 × 16 tiles). Maximum game time of M1 is limited to 3000 cycles, of M2 is 3000 cycles, and of M3 it is to 10000 cycles. A round-robin tournament is conducted, in which each algorithm plays 20 games (with various starting positions) against all other algorithms in each of 3 different maps (11 × 11 × 20 × 3 = 7260 games in total). The method used to compute the score of each algorithm is: the winner of each game is awarded 1 point, and both algorithms are awarded 0.5 points in the event of a tie. Each of the two AI players in all competitions begins with a single base, an equivalent resource value, and a single worker.

4.2 Experimental Results and Analysis

In this section, we compare the performance of PO-AHTNR with other algorithms in terms of the average score in three maps. Each algorithm plays 220 games against the other algorithms as player 1 or player 2 and 20 games against itself. Because data is collected from both players, the average score of each algorithm shown in Figure 3 is obtained effectively over 660 games.

The comparison evaluation considers the PO-AHTNR algorithm in addition to the following collection of search algorithms.

1. Random: A random strategy AI which executes actions randomly.
2. PO-LightRush: A hard-coded strategy AI, which only produces light attackers, and commands them to attack the enemy immediately.
3. PO-UCT: A Monte-Carlo AI, which employs an implementation with the extension for accommodating simultaneous and durative actions, with the single belief state strategy.
4. AHTNR-LL\F: An AHTNR AI with Low Level\ Flexible domain knowledge.
5. AHTNR-LL\F-Belief state: An AHTNR-LL\F AI with belief state generation strategy.
6. AHTNR-LL\F-Sensing action: An AHTNR-LL\F AI with sensing action.
7. PO-AHTNR-LL\F: An AHTN-LL\F AI with both sensing action and belief state generation strategy.

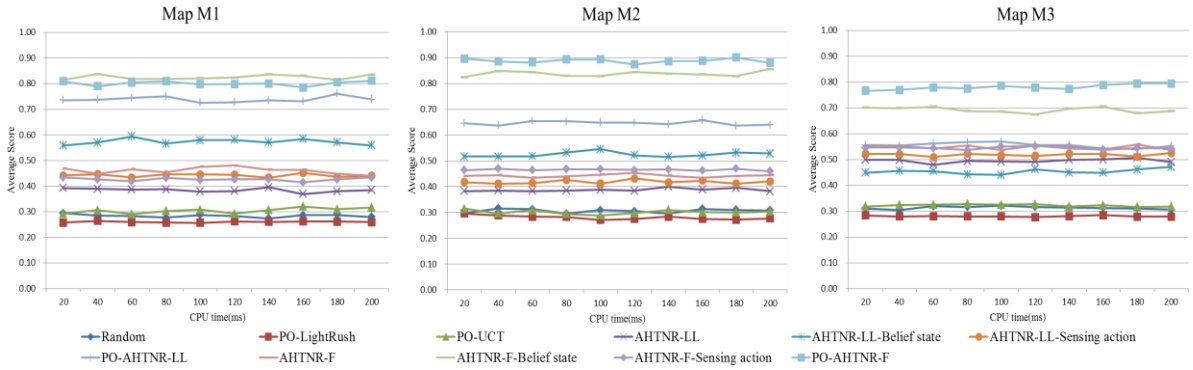

Figure 3: The average score of each algorithm obtained over the 660 games of the round-robin competition for map M1(8 × 8 tiles), M2(12 × 12 tiles), M3(16 × 16 tiles) with respect to the CPU time from 20 to 200 ms. The playout time is 100 cycles.

Figure 3 presents the comparison of the average scores obtained for each algorithm during the round-robin competition with respect to the CPU time from 20 ms to 200 ms for three maps. According to the

results, we can see that whether in low level or flexible domain knowledge, the PO-AHTNR algorithm outperforms all the other algorithms on all three maps under different CPU times. In addition, the performance of PO-AHTNR varies little with the increasing scale of the maps, and is very stable with respect to CPU time. PO-AHTNR outperforms AHTNR because of the consideration of unobserved environment, which allows PO-AHTNR to make more comprehensive plan with the available units and resources. PO-AHTNR-F performs best because it uses an flexible HTN domain knowledge for guiding the game tree search, which yields a good plan in a relatively short time.

With respect to other algorithms, we note that AHTN-LL\F-Sensing action algorithm and AHTN-LL\F-Belief state algorithm have both improved AHTNR algorithm in partially observable environment, and the combination of two strategy which noted PO-AHTNR performs even better. However, the performance of both PO-AHTNR-LL and PO-AHTNR-F deteriorate whether the map size is too small or large. The performances of both players deteriorate in map M1 because the sensing radius is relatively bigger, and the improvement of belief state strategy become relatively smaller. While in map M3, more units are produced on the larger maps and the domain knowledge is too simple to adapt to such a complex game. For AHTNR and PO-AHTNR, the Low Level domain knowledge means that the previous plan may not be suitable for the current condition, and it would be better to construct a new plan. The relative performances of the scripted methods Random, PO-Light-Rush and PO-UCT are also observed to improve with increasing map size, which is owing to the underperformance of PO-AHTNR and AHTNR on larger maps.

With respect to CPU time, the performances of all AI players change very little for all three maps. This is because all AI players require little time to make decisions. However, if the CPU time is very short, the performance will deteriorate. Because the performances of all AI players are similar under different CPU time settings, we employ a single CPU time setting of 100 ms in subsequent experiments.

Table 1: Average Scores of each HTN algorithm with different domain knowledge.

Domain Knowledge	**AHTNR**	**AHTNR-Sensing action**	**AHTNR-Belief state**	**PO-AHTNR**
Low Level	0.42	0.46	0.513	0.65
Flexible	0.483	0.48	0.786	0.823

Table 1 shows the average scores of each HTN algorithm with different domain knowledge. We can find that adding sensing action strategy improve win rate about 9% with Low Level domain knowledge, but not obvious improving with the Flexible domain knowledge. The addition of belief state generation strategy can bring a huge improvement for both domain knowledge, which is 22% for low level domain and 62% for flexible domain. And the win rate increment for proposed PO-AHTNR algorithm reaches to 54% and 70% for two domain knowledge.

Figure 4 shows the average decision times of AHTNR, AHTNR-LL\F-Belief state, AHTNR-LL\F-Sensing action, PO-AHTNR algorithms obtained over 20 games against with the 11 algorithms. We find that PO-AHTNR usually requires more decision time than AHTNR when they have similar domain knowledge. This is caused by the added time of reasoning the belief state and the enlarged decision space by the added of sensing action. Both the modifications will increase the decision time, adding the sensing action bring an average time increment of 38% for low level domain and 13% for flexible domain, adding the belief state generation bring an average time increment of 330% for low level domain and 214% for flexible domain. And the PO-AHTNR algorithm bring an average time increment of 384% for low level domain and 323% for flexible domain. However, the time increment is within one order of magnitude compared to the decision time of AHTNR. Therefore, although PO-AHTNR requires greater decision time than AHTNR, it is within an acceptable range.

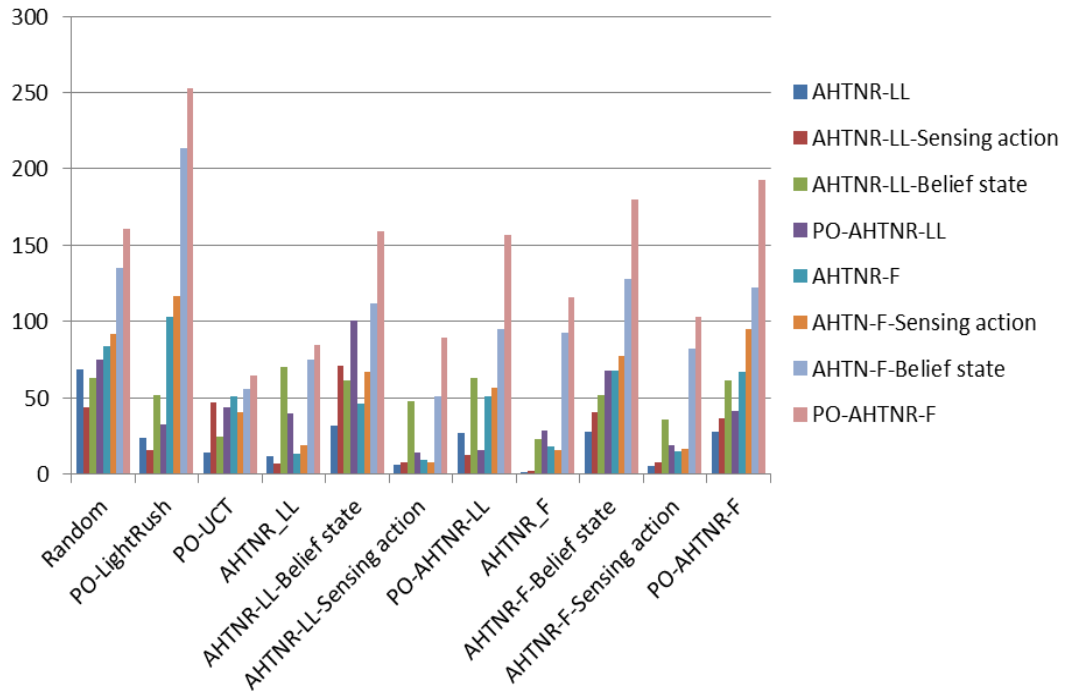

Figure 4: The average decision times of eight AHTNR algorithms obtained over 20 games against each of the 11 algorithms, where the CPU time is 100 ms for map M1 (8 × 8 tiles). The playout time is 100 cycles.

5 CONCLUSIONS AND FUTURE WORK

In summary, we have performed both experimental and theoretical study of planning in partially observable environment. An extended HTN description employing three additional elements denoted as history state set, observed state set and belief state set are introduced to express player's comprehend about the partially observed environment. In correspondence to dispose the unobserved environment, reconnaissance strategy based on sensing action and history-based single belief state generation are employed in PO-AHTNR algorithm. The experimental results in μRTS game successfully verify the algorithm's validation.

In future work, the PO-AHTNR algorithm can be extended in multiple directions. Our experiments demonstrate that the level of domain knowledge of the HTN has a significant influence on the performance of AHTN-R. However, encoding perfect knowledge in an HTN is a difficult and time-consuming process even for domain experts. The automatic extraction of HTN domain knowledge from thousands of RTS game replays may provide an efficient approach. In addition, we note that using probability to handle the non-deterministic problem may improve the veracity of planning. Therefore, a modified PO-AHTNR algorithm using probability may enhance its performance in partially observable environment.

ACKNOWLEDGMENTS

We want to thank Zhang Qi and Yang Mei for discussion about the PO-AHTNR algorithm's application for agent-based simulation. The work described in this paper is sponsored by the National Natural Science Foundation of China under Grant No. 61473300.

REFERENCES

Buro, M. 2004. "Call for AI Research in RTS Games". *In Challenges in Game AI: Papers from the AAAI Workshop*. AAAI Technical Report WS-04-04. 139–142. Palo Alto, CA: AAAI Press.

Churchill, D., and M. Buro. 2012. "Incorporating Search Algorithms into RTS Game Agents". In *Artificial Intelligence in Adversarial Real-Time Games: Papers from the 2012 AIIDE Workshop*, AAAI Technical Report WS-12-15. 2–7.Palo Alto, CA: AAAI Press.

Ghallab, M., D. Nau, and P. Traverso. 2004. "Automated Planning: Theory and Practice". 1st ed. Elsevier(Singapore). Pte. Ltd.

Humphreys, T. 2013. "Exploring HTN planners through examples". In *Game AI Pro 2: Collected Wisdom of Game AI Professionals*. A K Peters, 1st ed. Ltd.: Natick, MA, USA, 149–167.

Kelly, J.P. A. Botea, and S. Koeing. 2007. "Planning with hierarchical task networks in video games". In *Proceedings of the ICAPS-07 Workshop on Planning in Games*, Providence, September 22nd-26th ,RI, USA.

Laagland, J. 2014. "A HTN Planner for a Real-Time Strategy Game". Available online: http://hmi.ewi.utwente.nl/verslagen/capita-selecta/CS-Laagland-Jasper.pdf.

Lanctot, M, K. Waugh, M. Zinkevich, and M.H. Bowling. 2009. "Monte Carlo Sampling for Regret Minimization in Extensive Games." *Conference on Neural Information Processing Systems*. December 7th-10th, Vancouver, British Columbia, Canada DBLP, 1078-1086.

Muñoz-Avila, H., and D. Aha. 2004. "On the Role of Explanation for Hierarchical Case-Based Planning in Real-Time Strategy Games". In *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004*. Lecture Notes in Computer Science. Berlin: Springer.

Nau, D.S, Y. Cao, A. Lotem, and H. Muftoz-Avila. 1999. "SHOP: Simple hierarchical ordered planner". In *Proceedings of the 16th International Joint Conference on Artificial Intelligence-Volume 2*. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA. 968-973.

Nau, D.S., O. Ilghami, U. Kuter, J.W. Murdock, D.Wu, and F. Yaman. 2003. "SHOP2: An HTN planning system". *Journal of Artificial Intelligence Research*. 20(1):379-404.

Naveed, M., D. Kitchin, and A. Crampton. 2010. "A Hierarchical Task Network Planner for Pathfinding in Real-Time Strategy Games". In *Proceedings of the Third International Symposium on AI & Games*, Daniela M. Romano and David C. Moffat (Eds.), AISB, 1-7.

Ontañón, S. 2013. "The combinatorial Multi-armed Bandit problem and its application to real-time strategy games". *Journal of Essential Oil Research Jeor*, 18(2):185-188.

Ontañón, S., and M. Buro. 2015. "Adversarial hierarchical-task network planning for complex real-time games". In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press: Palo Alto, CA, USA. 1652-1658.

Sacerdoti, E.D. 1975. "The nonlinear nature of plans". In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*. No. SRI-TN-101. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA. 206-214.

Schwalbe, U. and P., Walker. 2001. "Zermelo and the early history of game theory". *Games and economic behavior*. 34(1). 123–137.

Shleyfman, A., A. Komenda, and C., Domshlak. 2014. "On combinatorial actions and CMABs with linear side information". In *Proceedings of the 21st European Conference on Artificial Intelligence*. Springer: Berlin, Germany. 263. 825-830.

Stanescu, M., N.A. Barriga, and M. Buro. 2014. "Hierarchical adversarial search applied to Real-Time Strategy games". *Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, 66-72.

Sun, L., P. Jiao, and K. Xu. 2017. "Modified Adversarial Hierarchical Task Network Planning in Real-Time Strategy Games". *Applied Sciences*. 7(9):872.

Uriarte, A., and S. Ontañón. 2017. "Single Believe State Generation for Partially Observable Real-Time Strategy Games". In *Computational Intelligence in Games* (CIG) August 22nd-25th, New York, USA, 296-303.

Zinkevich, M., M. Johanson, M.H. Bowling, and C. Piccione. 2007. "Regret minimization in games with incomplete information". *International Conference on Neural Information Processing Systems*. Curran Associates Inc. 1729–1736.

AUTHOR BIOGRAPHIES

WEILONG YANG is a PhD student for System Simulation at the National University of Defense Technology, CN. His research interests include HTN planning and Deep Learning in RTS games. His email address is yangweilong09@nudt.edu.cn.

LIN SUN is a Senior Lecturer at National Defense University, CN. He holds a PhD in System Simulation from the National University of Defense Technology. His research interests include HTN planning and Intelligence Decision. His email address is mksl163@nudt.edu.cn.

YONG PENG is an Associate Professor in the Department of Simulation and Modeling at the National University of Defense Technology, where he also obtained his Ph.D. His research interests lie broadly in HLA/RTI, Parallel and Distributed Simulation System. His email address is yongpeng@nudt.edu.cn.

QUANJUN YIN is a Professor in the Department of Simulation and Modeling at the National University of Defense Technology. His research interests include Cognitive Process Modeling, Qualitative Spatial Reasoning and Planning, Cooperation and Negotiation. He is a research group leader for Human Behavior Modeling. His email address is yinquanjun@nudt.edu.cn.