Cairo University
Faculty of Engineering
Computer Engineering Department
CMPN402

Machine Intelligence - Research Assignment

(Research paper review)

HTN Guided Adversarial Planning for RTS Games

Team 5

Team Members:

Mohamed Hassan Mohamed	1190118
Noureldin Ashraf	1190219
Moustafa Mohamed El-Nashar	1190212
John Maurice	1190403

Paper Information:

The paper was published in November 2020, by **Sun Lin**, **Zhu Anshi, Li Bo, Fan Xiaoshi,** in **IEEE**

Paper's Topic:

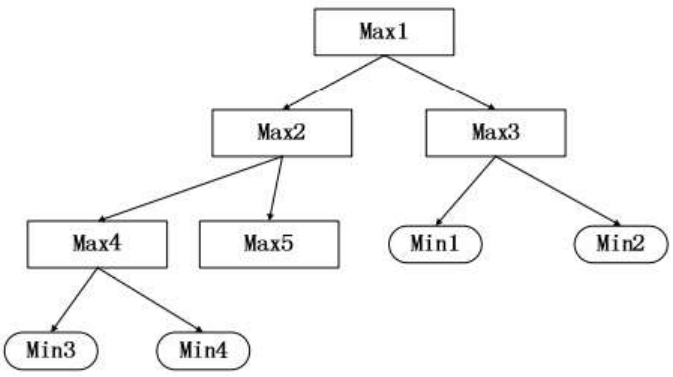
This paper is concerning the real time strategy (RTS) games, as it became an interesting topic in AI due to the great challenges it gets such as large state spaces, partial observation environment, limited decision times and adversarial behavior involved.

Due to those challenges if we applied the algorithms and approaches that perform well in board games such as Go or Chess the will not perform well at all in RTS games.

If we take an example to compare board game with RTS game we will find the following main differences;

1) state space in RTS is much larger than board games, as example 128×128 map in StarCraf [RTS] includes 400 unit if we will only consider the position of the units the number of possible states is about 10^{1685} , while the state space of Go [board game] is around 10^{150}. We can see that in RTS games the space grows combinatory with the number of units in the game and that's the main challenge with RTS games.
2) Actions in board game must be completed in turn, while in RTS games each action requires numerous steps to be executed.
3) Unlike board games RTS games does not consider turns, both sides can apply actions by employing multiple units concurrently.
4) RTS games have partial observable environment, while Chess have fully observable environment.

Thought time many algorithms have been applied on RTS games, such as Alpha Beta Considering Durative which is an extending alpha-beta pruning method. As studied in lecture alpha-beta reduces search space to $b^{m/2}$, but this usually not enough (if we used alpha-beta in the example of StarCraf we will get $10^{842.5}$ which is still very big), and that is where hierarchical



task network (HTN) algorithm is introduced, which reduces the searching space even further. Another algorithm was used and showed favorable results was adversarial hierarchical task network (AHTN), which use HTN to reduce the state space combined with game tree search.

AHTN assumes there are two players max and min and it searches the game tree in a similar to min-max game search by using the HTN planner to expand the nodes.

As we see now AHTN addressed large state as it use HTN planning, and it also cover the durative actions (multiple steps action), and it did achieved good performance compered to the algorithms before it. However it suffers from some weaknesses first covering the concurrent and simultaneous actions, as it does not care what max do while calculating min.

Findings:

AHTNCO better considers the concurrent nature of the problem by incorporating the opponent adversarial strategy in the search through generating min nodes unlike AHTN which uses the same HTN planner for both players. Afterwards AHTNCO executes the very first action it finds and alternates to max nodes. This ensures that plans made by a player are not short sighted as the adversarial player's actions are accounted for.

AHTNCO utilizes a time constraint in the search so that levels deeper in the search tree are more time limited than levers that are higher up. This ensures that the agent will not get stuck on a branch for too long. Here the paper talks about a compromise between the time limitation and being able to reach enough depth so the time limitation cannot be too high either. The time limitation used for the paper is t_{max} / d where d is the current tree depth.

Pseudo code for the min version of the AHTNCO is illustrated in the following figure. The algorithm does alpha beta pruning combined with the greedy approach of choosing the very first action that appears. If no action is available, the algorithm proceeds to consider complex choices aiming to reach a primitive task that can be executed. These are refered to as choice points by the algorithm. If the algorithm runs out of time, it will return the best action found so far. Otherwise it will consider the rest of the coice points and only alternating to a max player if an action is reached. This greedy approach of choosing the very first action serves the algorithm well as it handles concurrent actions better than continuing to consider all choice points. It is also stated in the paper that evaluation functions are used but this is not shown in the pseudo code.

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** $(CP^*_{\max}, CP^*_{\min}, v^*, \alpha, \beta)$

Suggestions:

While the paper uses a time limitation function t_{max} / d, more flexible time limitation functions can be used which can better capture the effect of different tree branching factors. This function currently does not account at all for tree branching factor. So, if a game with many actions is input to the algorithm, the agent will usually disregard most available actions as it will spend most of its time exploring few branches. If we were to assign time limitations that are inversely dependent on the branching factor, this would limit the time that the agent spends on one branch. There is a choice to be made between exploitation and exploration when choosing how dependent the time limitation function is on the branching factor, because if we choose an extreme function such as t_{max} / b^d, this will ensure that the agent will spend equal time in all branches but at the compromise of how far into the future the agent searches.