DYNAMIC REAL-TIME HIERARCHICAL HEURISTIC SEARCH FOR PATHFINDING

M. Naveed, D. Kitchin And A. Crampton
University of Huddersfield, Queensgate, Huddersfield HD1 3DH, UK

ABSTRACT

Movement of Units in Real-Time Strategy (RTS) Games is a non-trivial and challenging task mainly due to three factors which are constraints on CPU and memory usage, dynamicity of the game world, and concurrency. In this paper, we are focusing on finding a novel solution for solving the pathfinding problem in RTS Games for the units which are controlled by the computer. The novel solution combines two AI Planning approaches: Hierarchical Task Network (HTN) and Real-Time Heuristic Search (RHS). In the proposed solution, HTNs are used as a dynamic abstraction of the game map while RHS works as planning engine with interleaving of plan making and action executions. The article provides algorithmic details of the model while the empirical details of the model are obtained by using a real-time strategy game engine called ORTS (Open Real-time Strategy). The implementation of the model and its evaluation methods are in progress however the results of the automatic HTN creation are obtained for a small scale game map.

Keywords Pathfinding, Real-Time Strategy Games, Automated Planning, Real-Time Heuristic Search.

1 INTRODUCTION

Real-Time Strategy games are characterised as the war games where players command armies which fight with the opponent's armies in real-time. The most popular titles in the genre of RTS games are StarCraft, Warcraft (Blizzard Entertainment) and Age of Empire (Ensemble Studios). In these games, the players compete for collecting the resources (scattered over a 2D terrain) by building economies & armies and engaging the armies into battle to occupy/defend the resources. In RTS games, the characters controlled by the computer (usually called Non-Player Characters or NPC) are required to solve complex tasks very quickly and effectively during the game play. One of these tasks is pathfinding. Pathfinding is a search process to find a minimum set of obstacle free nodes (called a shortest path) from one node to another in a game world. A game world is represented by a data structure called a game map.

Pathfinding in RTS becomes a complex and challenging task mainly due to the presence of a huge search space and very little time-span to solve it. In commercial games, a very small amount of CPU time is spared for the pathfinding e.g. Bioware Corp, a big game company, allows 1-3 milliseconds for the pathfinding of all units (Bulitko et al (2007)). Concurrency is another factor in RTS games which makes pathfinding challenging as various players compete for the executions of actions to achieve their goals at the same time.

Jeff Orkin (2006) defines automated planning (AI Planning) as a formalized searching mechanism to find a set of actions to meet a goal. According to this definition, Pathfinding is a planning problem and provides the motivation for the exploration of sophisticated automated planning methods in games. Dana Nau (Gallab et al (2004)) characterises AI planning into three kinds: domain-specific, domain-independent and configurable planning methods. Domain-specific planner are made or tuned for a specific domain while domain-independent planners can work in more than one domain and do not require information other than given in the domain model. Configurable planners are the domain-independent planners with an additional input, containing domain knowledge, about how to solve the tasks in the domain. One example of configurable planner is the HTN (Hierarchical Task Network) planning.

HTN (Gallab et al (2004)) is the network of tasks organized in a tree form in such a way that a parent node of the tree represents a higher level task (abstract task) while a child node represents a lower level abstract task or a primitive task. A primitive task is an action which can be executed automatically while an abstract task is decomposed into another low level abstract task or a primitive task. An HTN does not provide a complete solution rather it gives clues how to solve it.

Real-time Heuristic Search (Korf (1990)(Bulitko et al (2008)) is a family of search algorithms where algorithms exhibit a unique property of search. This property says that an agent must plan actions and executing them repeatedly, within a fixed time interval that is independent of the size of problem space. These kinds of algorithms retain the property of completeness even when a permanent limit is imposed on per-action planning time. These algorithms seem suitable for solving the pathfinding problem in RTS with fixed time interval. As argued by Bulitko et al (2008), RHS can give unacceptable suboptimal solutions due to converging of heuristic function into local minima (a condition called heuristic depression). Therefore, the research on real-time heuristic search is still continuing and requires further exploration.

In this article, a novel HTN based real-time heuristic search is proposed for solving the pathfinding problem effectively in RTS games. HTNs are used to restrict the boundaries of the game map for real-time heuristic search to reduce the search space and to keep the search in the direction of the goal while the heuristic function of RHS is also modified to avoid exploring the nodes which can lead to dead-ends. HTNs are created automatically and the procedure is independent of the kind of game or game map. Main contributions of this work are the automated creation of HTNs for the RTS games and the exploration of a hybrid of HTN and RHS in a real-time system. ORTS (Buro&Furtak (2005)) is used a testbed. ORTS has four sample RTS games for the research purpose.

The rest of this paper is organized in a way that the section 2 provides a description of the related work to the proposed model while section 3 describes the proposed model. In section 4, the results of initial experiments are shown and discussed. Section 5 concludes the work of this paper.

2 RELATED WORK

This section first describes the relevant existing works regarding the hierarchical representation of the game map and then those which are related to the real-time heuristic search.

HPA (Botea et al (2004)) (Hierarchical Path-Finding A*) divides the game map into '*N'* distinct but connected local blocks. The top hierarchy is the abstract representation of main blocks while lower hierarchies represent the sub-blocks of the main blocks. A data structure called a transition is used to represent the connection between two abstract blocks. A transition is composed of two nodes and one edge. Both nodes in the transition are located at the boundaries of two different blocks and the edge between them is always of length '1'. Edges are also established between boundary nodes of the same block called intra-edges. The abstract representation in this case stores the information of transitions, intra-edges and the nodes of the intra-edges. Those boundary edges are selected which are obstacle free. This representation remains static during the game play.

In this work, HPA is explored in 120 maps of different sizes and is compared with a low level A*. The low level A* finds a path between two locations using the whole game map where as HPA searches for the path between them using the abstract representation of the game map. The average results show that HPA takes far less CPU cycles for the paths with large size but low level A* performs better than HPA in the small paths and in the cases where the start node and the target node are in straight line.

HPA has some limitations. For example, the costs of traversing from one block to another block and moving from one boundary node to another are static while the game world changes continuously during game play. There is no formal way of finding the number of blocks per hierarchy and the quality of the plan generated by the model depends on the number of blocks in the abstract representation.

PRA* (Sturtevant& Buro (2005)) (Partial Refinement A*) creates an abstract representation of the game map like a family tree rather than an overlaying structure as used in HPA. Another notable feature of this abstract representation is that it does not rely on an arbitrary number *N*, for the division of the game map into blocks. PRA* uses the basic unit of the grid map i.e. tile to create the abstraction. It uses two patterns of nodes: clique and orphans. In a clique all nodes are connected with each other so the length of distance between any two nodes of a clique is '1'. Each clique is then represented by an abstract node (like a parent node). Once all cliques are found and abstracted then the search for the second pattern, called orphan nodes, is started. An orphan node requires only a single move action to reach it. Once all nodes are abstracted, the abstracted nodes are joined together to form an abstract graph.

PRA* applies heuristic search for the selection of the level of abstract graph for finding an abstract path between source node and the target node. In this work, PRA* (with interleaving plan making and execution) is explored in a RTS game (assuming a static world) and the quality of the path found by PRA* is compared to a simple A* search. The results show that PRA* finds a path faster than A* and gives 98% optimality (i.e. performance is reduced by 2%) if time to run is reduced to 1msec/frame from 100msec/frame on a dual-processor Power-MAC computer of 2GHz CPU speed and 1 GB of RAM.

LRTA* (Korf (1990)) (Learning Real-time A*) is a real-time heuristic search method which applies action selection and execution alternatively during the planning process. To keep the path search focused towards the actual goal along with the execution of actions, a non-negative (heuristic) value is associated with each traversed node of the game map. This heuristic value of a given node keeps the information about the approximation of how far this node is from the actual target. This heuristic value is updated during each action selection mechanism and is updated using a look-ahead method. Look-ahead process finds the estimated goal distances for the outcomes of all available actions at a given state. An action is selected for the execution which gives minimum cost (estimated goal distance) in a look-ahead search. In addition to having a capability of interleaving path planning and execution, LRTA* also tunes the heuristic function *f(s)* (estimation of goal distance) in real-time. The results show that LRTA* tunes the heuristic function during the planning process and finds the acceptable path. We have modified it by using a new heuristic parameter to avoid the obstacles in the ever changing game environment for pathfinding.

PR LRTS (Partial Refinement Learning Real-time Search) (Bulitko et al (2007)) uses a variation of real-time heuristic search called LRTS (Bulitko and Lee (2006)) with the abstraction of game map. LRTS modifies the LRTA* by adding two control parameters: optimality weight $\gamma \in (0,1]$ and learning quota $T \in [0,\infty]$. Optimality weight is used to tune the heuristic value associated with each node of the game map while T decides what actions to choose for the execution. In PR LRTS, LRTS searches for the path plan on the nodes of an abstract graph instead of on actual ones. This work is very similar to our pathfinding model except we use an abstract representation of the game map which also provides partial solutions i.e. game map representation in the form of HTN. In PR LRTS, LRTS is applied on abstract graph to create a small set of lower level abstract nodes from the start position to the goal positions while actual path is calculated by A*. The abstract graph of the game map is built using the abstraction mechanism of PRA*. The results taken on six different game maps show that PR LRTS give the best results if LRTS is applied on top hierarchies and A* on the lowest level.

To automatically adjust the depth of the look-ahead process in real-time heuristic search, Bulitko et al (2008) applied decision-tree and a pre-computed depth database approach in a game environment for this purpose. The training dataset for the construction of decision-tree classifier is built using the features of the state-space search (e.g. the initial heuristic estimate of the distance from the current state to the goal etc) and a large number of search iterations are performed to assign the label to the training instances. Once the classifier is built at the pre-processing stage, it is attached with real-time heuristic search for making the decision in real-time where the classifier makes an estimate for the depth of the look-ahead at each current state during the planning process. To tune the heuristic function of real-time search (for more accurate estimates), the authors have used intermediate sub-goals to measure the heuristic values rather than using the global goal. Sub-goals are selected using the entry points from one abstract state to another (using the abstract representation of PRA*). This approach is known as PR LRTA* and empirical results show that dynamic LRTA* is significantly faster than PR LRTA* and PRA* in planning per move.

3 MODEL DESIGN

In our work, a hybrid of HTN and LTRA* is explored for solving the path-finding problem in real-time strategy games. An automatic and domain-independent mechanism of HTN creation is proposed and evaluated in ORTS. While LTRA* is modified by adding a new heuristic function $b(s)$ to estimate the volume of obstacles in a look-ahead search. The reason behind adding this parameter is to avoid exploring the nodes which can lead to an obstacle. The proposed cost function $f(s)$ of a current state s in a state space S to move a character from an initial state s_o to a goal state s_t is shown in equation (1) while $b(s)$ is shown in equation (2).

$$f(s) = g(s_o, s) + h(s, s_t) + b(s) \qquad \forall s, s_o, s_t \in S \qquad (1)$$

$$b(s) = \sum_{i=1}^{|\bar{S}|} \eta(\bar{S}_i) \qquad (2)$$

Where $\bar{S}$ is the set of all possible successor states of s and $g(s_o, s)$ is the exact distance from start state s_o to s while $h(s, s_t)$ is the estimated distance from current state to the goal state. In the initial experiments, we are using three possible values for the obstacle parameter η as shown in equation (3) where state enumeration *"per_obst"* means it has a permanent obstacle; *"temp_obst"* means state has a temporary obstacle while *"free_obst"* means it is an obstacle free state.

$$\eta(s) = \begin{cases} 1 & if \quad s = per\_obst \\ 0.5 & if \quad s = temp\_obst \\ 0 & if \quad s = free\_obst \end{cases} \tag{3}$$

The pseudo-code of the proposed HTN based LTRA* is given in figure 1. HTN are created at the pre-processing stage according to the procedure given in figure 2. The state space is divided into *K* blocks called zones and each block is further divided into *K* sub-blocks called sub-zones. This partitioning of state space is somewhat similar to the one used in the HPA. However, the abstract representation is in the form of HTNs. we have used an automated approach for the creation of HTN where HTNs are created using a heuristic search method (i.e. A*). The top hierarchy of the HTN shows an abtract path from one zone to another one. The lower hierarchy represents an abstract path from the sub-zone of the start position to that of the target position.

```
/* Pre-processing work*/
    // for a give set of 2D vertices V and an arbitrary number K
    1-  N=factorial(K);
    2-  Struct HTN[N];
    3-  HTN=CreateHTN(V, K);
/*Real-time search*/
HTN_LRTA*(HTN, V, S, G)  // S the start position and G is the goal position
{
    1   HTN hn=GetHTN(HTN, S, G);
    2   V^T=CreateNewVerticesSet(hn);  // V^T is a smaller version of V
    3   Apply LTRA*(V^T, S, G);
}
```

Figure 1: Pseudo-code of the proposed HTN based LRTA*

```
/* V is the set of 2D vertices, k is an integer value, the structures ZV and SVZ are to represent
the zones and sub-zones respectively. */

HTN CreateHTN(V, K)
{
1- divides into k zones

2- All zones are again divided into k subzones, /*search space is divided into k^2 parts*/

3- Execute A* to find the list of subzones and zones  from one subzone to all other subzones

4- create HTNs to move from each i^th zone to all j^th zones
        HTN[k,k]=new HTN(); // Structure
        HTN[i,j]->Precondition.CurrrentZone=PTR(I);  // PTR means pointer
        HTN[i,j]->Precondition.TargetZone=PTR(j);  //zone of goal state

If(i!=j)
{
/**Preconditions/

For m=1 till m<k
  For n=1 till n<k
        HTN[i,j]->Precondition.CurrentsubZone=getSubZone(PTR(i), m);
        HTN[i,j]->Precondition.targetSubzone=getSubZone(PTR(j), n);

        /*Decompositions*/
        If(SVZ[m,n]->nodelist!=NULL)
        {
            HTN[i, j]->subzones=SVZ[m,n]->nodelist;
            HTN[i,j]->Zones=SVZ[m,n]->Zonelist;
        }
        Return HTN;
}
```
Figure 2: Pseudo-code of HTN creation

4 EXPERIMENTAL RESULTS

Experiments are performed using Microsoft Windows XP (Version 2002) on a Pentium (R) 4 machine with 3.00 GHz CPU cycles and 1.00 GB of RAM. The testbed RTS game (ORTS) is modified in VC++ 2008 to implement the model. The complete implementation of the model is in progress however the HTN creation part of the model has been implemented for a game map of 484 nodes. The initial results are shown in Table 1. The CPU times shown in table 1 are the average of 5 runs.

Table 1: Average CPU time of HTN creation with respect to number of zones

Sr No	Zones	Game-map Size	CPU Time (Seconds)
1	2	484	28
2	3	484	35
3	4	484	39
4	5	484	70
5	6	484	116
6	7	484	190

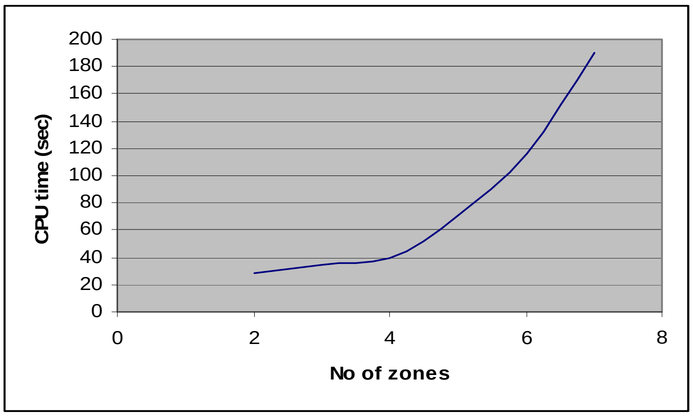

Figure 3: Profile of CPU time with respect to number of zones

The results shown in table 1 and figure 3 reveal that the time for HTN creation grows exponentially with increasing the number of zones in a given game map. However the results are performed only for a small scale game map and more experiments are in process for the large scale game maps. Another interesting investigation in HTN creation would be to see the relationship between number of zones and memory consumed by HTNs which could lead us to decide what number of zones can be explored for the model investigation.

5 CONCLUSION

A hybrid solution for solving the pathfinding problem in the RTS games has been proposed and the novelty of the work has been highlighted. However, the work on the empirical evidence of the solution is in progress. The initial experiments show that the pre-processing phase of HTN creation can consume a huge CPU time if increasing the number of zones to partition the state space. The existing literatures related to the use of game map abstraction for making the higher level path plans show the advantage of the approach and a suitable combination of real-time heuristic search and abstract representation of the game map can enhance the performance of the hybrid solutions for the pathfinding in real-time systems like RTS games. In future, we are aiming to use perceptions with real-time heuristic search to explore the combination of machine learning and AI planning in real-time strategy games.

REFERENCES

BOTEA A., MULLER M., and SCHAEFFER J. (2004). *Near Optimal Hierarchical Path-Finding*. Journal of Game Development, Vol. 1, No.1, pp: 7-28.

BULITKO V. and LEE G. (2006). *Learning in real time search.* Journal of Artificial Intelligence Research (JAIR), Vol. 25, pp: 119-157.

BULITKO V., LUSTREK M., SCHAEFFER J., BJORNSSON Y., and SIGMUNDARSON S. (2008). *Dynamic Control in Real-Time Heuristic Search.* Journal of Artificial Intelligence Research (JAIR), Vol. 32, pp: 419-452.

BULITKO V., STURTEVANT N., LU J., and YAU, T. (2007). *Graph Abstraction in Real-time Heuristic Search.* Journal of Artificial Intelligence Research (JAIR), Vol. 30, pp: 51-100

BURO M. and FURTAK T. (2005). *On the Development of a Free RTS Game Engine*. Proceedings of Gameon'NA Conference, Montreal, Canada.

GALLAB M., NAU D., and TRAVERSO P. (2004). *Automated Theory: Planning and Practice*. Morgan Kaufmann Publishers, ISBN 1-55860-856-7.

GENESERETH M. and NOURBAKHSH I. (1993). *Time-Saving tips for problem solving with incomplete information*. Proceedings of the AAAI-93, pp: 724-730, Washington, DC, USA.

KORF R. (1990). *Real-time heuristic search*. Artificial Intelligence, Vol. 42, No.2-3, pp: 189-211.

ORKIN J. (2006). *Three States and a Plan: The A.I. of F.E.A.R*. Proceedings of Game Developer Conference 2006.

STOUT B. (1996). Smart Moves: I*ntelligent Pathfinding*. Game Developer Magazine.

STURTEVANT N., and BURO, M. (2005). *Partial Pathfinding Using Map Abstraction and Refinement*. Proceedings of AAAI, pp: 1392-1397.

.