Proceedings of the Third International Symposium on AI & Games, Daniela M. Romano and David C. Moffat (Eds.),
at the AISB 2010 convention, 29 March – 1 April 2010, De Montfort University, Leicester, UK

A Hierarchical Task Network Planner for Pathfinding in Real-Time Strategy Games

Munir Naveed[1], **Diane Kitchin**[1] **and Andrew Crampton**[1]

Abstract. In this paper, we propose an automatic mechanism of Hierarchical Task Networks (HTNs) creation for solving the problem of real-time path planning in Real-Time Strategy (RTS) Games. HTNs are created using an abstraction of the game map. A real-time heuristic search approach called Learning Real-Time A* (LRTA) is applied to execute the primitive tasks of the HTNs. The main purpose of using a HTN based real-time path planner is to restrict the real-time search to a shorter part of the problem space while keeping it in the direction of the actual goal position. Results show that the hierarchical approach reduces the suboptimality of the LRTA and speeds up the convergence process.

1 INTRODUCTION

Real-time heuristic search is also known as agent-centered search [1]. This kind of search method interleaves plan making and plan execution. Another notable characteristic of this group of search algorithms is that it keeps the planning process within the neighbourhood of the current state. This means that a real-time heuristic search method looks at the part of the domain world that is directly relevant (i.e. successor states) in its present situation, determines what actions to execute within this part and then executes them. This episode of action selection and execution is done in a fixed time interval that is independent of the size of the domain world. This process continues until the goal state is reached.

HTNs are not built with the objective of achieving a set of goals but to carry out some set of tasks [2]. These tasks can be either primitive or nonprimitive. A nonprimitive task is the abstract task which can be decomposed into sub-tasks (either primitive or non primitive). Primitive tasks are the ground tasks which can be executed using the planning operators. A HTN contains methods to decompose the nonprimitive tasks into subtasks.

One of the motivational examples for real-time pathfinding is Real-Time Strategy (RTS) Games [3]. In RTS games, solving the pathfinding problem becomes a challenging and complex task due to concurrency (i.e. various characters can start the pathfinding task simultaneously) and hard constraints on CPU time (e.g. BioWare Corp. allows 1-3 milliseconds for the pathfinding of all units [3]).

LRTA [4] is an adaptive learning method that satisfies all the essential properties of the real-time heuristic search. It uses a breadth-first approach to find all successor states of a current state in a look-ahead search of a constant depth. In the look-ahead search of depth *d*, it selects a successor state with minimum cost, *d* actions away from the current state. A cost is the sum of two distances; a distance from the current state to the successor state and an estimated distance (also called heuristic value) from the successor state to the goal state. The heuristic value of the current state is modified with that of the selected successor state and the path from the current state to the successor state is executed. This process continues until the goal state or a stopping condition is reached.

LRTA suffers from two main problems: it works with a fixed look-ahead depth and produces poor solution quality due to convergence of the heuristic values into local minima (also known as heuristic depression [5]). Ishida proposed to use A* [6] to rectify the results of LRTA when it is trapped in a heuristic depression.

In this paper, we explore real-time path planning using HTNs and LRTA with the following two contributions. Firstly we demonstrate a method of automatic HTN creation. The main job of HTNs is to use LRTA search for actions selection in a smaller part of the problem search space. We have used a sector-based abstraction of the game map and a simple A* to automatically create the HTNs. The abstraction is built by dividing the game map into large blocks. Our second contribution is to propose an HTN planner by integrating the hierarchical information of the game map with LRTA. We also provide an empirical evaluation of the proposed hierarchical approach.

We use Open Real-Time Strategy (ORTS) [7] environment as a platform to explore real-time path planning in RTS games. It is an open source tool that provides an RTS game programming environment to explore AI (Artificial Intelligence) in RTS games. It also provides options to create new game definitions and user-friendly widgets (interfaces) for human interaction with the game AI during the game play.

The rest of the paper is organised as follows. In section 2 we discuss the algorithmic details of a procedure to build the game hierarchies. The third section describes the details of the automatic HTN creation. Section 4 discusses the proposed HTN planning model for path finding with LRTA. In section 5 we describe the existing relevant work. The section "Experimental Setup" provides the details about the experimental design of the model and evaluation parameters. Section 7 gives the experimental outcomes and their analysis. The final section concludes the work.

2 GAME MAP HIERARCHIES

To build the game map hierarchies, we use a sector-based partitioning of the map. In this partition, a game map is divided into distinct large blocks. Each block is called a zone. Figure 1 shows the partitioning of a game map into four blocks. A zone is represented by a structure with three coordinates and two

[1] Dept. of Informatics, Univ. of Huddersfield, HD1 3DH, UK.
Email: {**m.naveed**, **d.kitchin**, **a.crampton**}@**hud.ac.uk.**

strategic parameters. Figure 2 shows a C++ like representation of a zone. Three coordinates are used to represent the geographical area covered by a zone. These coordinates are the start, end and center points of a rectangular area covered by a zone as shown in figure 3.

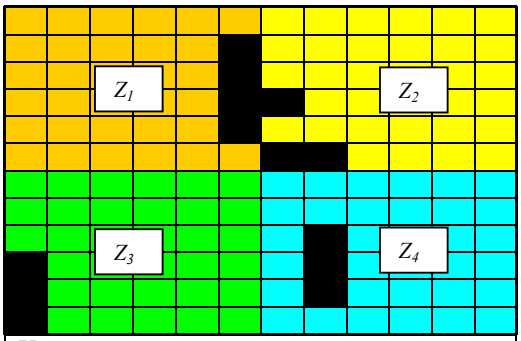

Figure 1. Partitioning a game map into four zones.

```
Struct Zone
{
   startPoint(int x_o, int y_o);
   endPoint(int x_e, int y_e);
   centerPoint(int x_m, y_m);
  double obsDensity[4];
  double enemyDensity[4];
  Region region[4];
 int index;
}
```

Figure 2. A structure of zone representation.

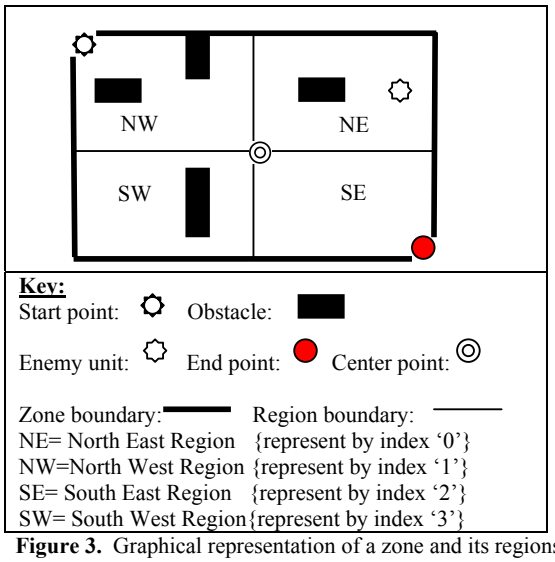

Figure 3. Graphical representation of a zone and its regions.

The center of a zone is moved to the nearest empty cell (point) if it lies on an obstacle. A zone is divided into four sub blocks called regions. The boundaries of each region are calculated using three coordinates of a zone. After creation of four regions, the strategic parameters are calculated. The strategic parameter o*bsDensity* represents the amount of obstacles that a region contains while *enemyDensity* is used to represent the strength of the enemy units in a region. Equation (1) shows the calculation of o*bsDensity* in a region *R* and *enemyDensity* is measured using equation (2). The algorithmic details of the construction of the game map hierarchies are given in figure 4.

$$obsDensity(R) = \frac{\alpha}{\eta} \quad (1)$$

$$enemyDensity(R) = \frac{\beta}{\eta} \quad (2)$$

In above equations, α represents the number of cells covered by the obstacles in a region, β represents the number of cells covered by the enemy units in a region and η is the total number of cells in a region.

```
/*Map is a structure representing 2D game map. N is a
positive whole number such that N>1 and represents number
of zones in a map.  */

Build_mapHierarchies( Map &map, int N)
{
   1-Zone allZones[N]=CreateZones(map, N);
   2- AdjustzoneCenters(map, allZones); /*if theoretical
      center lies on an obstacle*/
   3- CalculateRegionBoundarys(allZones);
   4-CalculateCenterofRegions(allZones)
   5- CalculateObstacleDensity(allzones);
   6-CalculateEnemyDensity(allzones)
}
```

Figure 4. Procedure of building map hierarchies.

3 CONSTRUCTION OF HTN

To solve the problem of moving a game object (a worker in our case) from one location to another location on an actual game map, we propose a solution by dividing the main problem into tasks and sub tasks. We assume the main task (of moving an object between two locations) is the planning of a route from one zone to another zone and the sub-task (and primitive task) is to find a path at the regional level. The primitive task is executed by using a variation of LRTA. In the presence of hierarchical information of a game map, we propose to create the task tree automatically (and initially offline) and then convert it in the form of an ordered task network. A representation scheme of the main task, given in figure 5, is used to create main tasks for path planning between the zones. According to this scheme, a main task of moving an object from zone A to zone B is different than moving the same object from zone B to zone A.

```
Struct MainTask
{
  /*Precondition parameters*/
   Zone sZone; // zone of start position
   Zone tZone;// zone of target location
   Boolean min_obst;/* path with regions of minimum
                       obstacle density. */
    Boolean noenemy;  /* path with regions of minimum
                       enemy density*/
   /*Subtasks*/
    Region *list_of_regions;
   /*For future use in heuristic function*/
    int eucDistance;
    int actualDistance;
 }

Struct TaskNetwork
{
   int index_sZone; /* index (zone number) of the zone of the
                       start position*/
   int index_tZone; /* index (zone number) of the zone of the
                      target position*/
    MainTask *list_of_tasks;
}
```

Figure 5. Main task and task network representation.

To create a list of all regions within a main task, we have used a modified version of a simple A* algorithm (implemented by Michael Buro and Sami Wagiaalla) which is a part of the free ORTS software. Algorithmic details of the creation of the Hierarchical Tasks Networks are given in figure 6.

```
TaskNetwork htn[N][N]; // A lookup table
HTN CreateHTN(Map &map, Zone* allZones, int N )
{
For i=0:N
     For j=0:N
     1- Set the htn zone indices
      /* htn[i][j].sZone=allzone[i].index;
        htn[i][j].tZone=allzone[j].index;*/

     2- Create a new Main Task for path finding with
         min_obst =false
     3- Set the precondtions
     4- Calculate list of regions
     5- Create a new Main Task for path finding with
         min_obst =true
    6- Set the Preconditions
    7- Calculate list of regions
    8- Create a list of main task
    9-Attach the list to htn array
  End For
 End For
}
```

Figure 6. Procedure of automatic HTN creation.

4 HTN PLANNING

Like other HTN Planners e.g. [8], our proposed HTN Planning model, called HTN-LRTA, also performs both planning activities (i.e. task decomposition) and instantiation of operators (i.e. finding of a sequence of actions to execute the primitive task). HTN-LRTA decomposes a main task into subtasks in a sequential way rather than a recursive one. However, the selection of a main task (non-primitive task) relevant to the current planning problem is performed in a random access way as the HTN structure is in the form of a lookup table. Once a primitive task is found, then LRTA is applied to execute this task. An algorithmic detail is given in figure 7.

HTN-LRTA has one method, called *Decomp,* which is a simple forward search with a linkedlist traversing mechanism. The procedural details of *Decomp* (in a C++ style) are given in figure 8. To implement LRTA in ORTS, we have modified a simple A* with a look-ahead of 1-ply.

```
/*HTN-LRTA path planner*/
// P1 is the start position and P2 is the goal position
// d is the depth of look-ahead task in LRTA

HTN_LRTA( P1, P2, htn, Map  &V, d)
{
 1-Zone z1=Zoneof(P1);
 2- Zone z2=Zoneof(P2);
 3- MainTask T=htn[z1.index][z2.index].list_of_regions.
 4-Regionlist l1=Decomp(T, z1, z2, true)
 If l1 is Null
     6- Print "Target out of Reach"
     7- Return
 Else
  While l1→next!=NULL
      8- P1= LRTA(P1, l1→Center, V, d )
      If P1=l1→Center
        9- l1=l1→next
      End if
   End While
 End if
}
```

Figure 7. HTN-LRTA procedure.

```
/* z1 is the zone of the start position, z2 is the zone of target
position, mp is the minimum obstacle path and bool
represents boolean data type*/

Region* Decomp(MainTask *T, Zone z1, Zone z2, bool mp)
{
 1-Region *list=Null;
 If T=Null
     2- Return list;
 Else
  /*use pre-conditions to select the relevant subtask*/
  While T→next!=Null
    If T→sZone=z1 && T→tZone=z2 && T→ min_obst =mp
         3- list=T→list_or_regions
    End if
        4- T=T→next;
  End While
 End if
 5- Return list;
}
```

Figure 8. HTN Method

5 RELATED WORK

The use of game map abstraction for speeding up the pathfinding process has been explored in Hierarchical Path-finding A* (HPA*) [9] and Partial Refinement A* (PRA*) [10]. In both approaches, A* is applied to make a path plan on a high-level abstract graph. This abstract path is then refined to a path with coordinates of the actual map. However, they both use different abstraction and refinement methods. HPA* divides the game map into large blocks and connects the neighbouring blocks using entrance points. The set *s* of entrance points, along a border line (say) *l*, of a block is identified using four conditions: (i) each point of *s* lies on *l*, (ii) for each member of *s*, there is a symmetrical point (on the boundary of a neighbouring block) that is adjacent to the corresponding point of *s*, (iii) all points of *s* are obstacle free and (iv) *s* is extended along *l* as long as all the previous three conditions are met.

The optimal paths between all entrance points of a block are computed offline using A* and cached in a look-up table for their use in the refinement task during an online path search. These entrance points of each block are also used to build an abstract graph of the game map. The abstract graph is built around transitions between the blocks. The number of transitions between two blocks is dependent on the number of entrance points between two neighbouring blocks e.g. there are two transitions if the set's size is 6, one transition if the set is of size 3 and no transition if the set is empty. During an online search, both start and target positions are inserted (temporarily) into the abstract graph by attaching them to the corresponding borders of their blocks. A* search is applied on the new abstract graph for computing an abstract path. In the refinement process, each block crossing is replaced by the pre-computed paths (stored in the look-up table). After refinement, HPA* also applies path smoothing (as a post-processing step) to improve the solution quality (i.e. path length). The results show that the abstraction provides a huge reduction in the number of nodes explored during the path planning, especially when the solution length is large. However, for short paths, A* performs faster than HPA*. At higher levels of abstraction, the integration of start and target positions with the abstract graph takes more time than inserting them at a low level of an abstract graph.

Our work has one similarity with HPA* i.e. the use of block (rectangular) shaped regions for the creation of game map hierarchies. However, we use a different form of block representation in HTN-LRTA than that of HPA*.

PRA* [10] uses an abstraction approach that is based on two local features of the game map i.e. cliques and orphans. A clique is a group of nodes which are connected to each other. In this work, a maximum 4-clique is used. An orphan is a node that is connected to only one node. The abstraction process moves through the game map and finds all the cliques. Then each clique is represented by one abstract node. An orphan node is attached to the abstract node which it neighbours. PRA* uses A* with a modified heuristic function i.e. it uses the *octile distance* to estimate the distance between two abstract nodes. It dynamically selects an abstraction level to start the search from. PRA* refinement process breaks the abstract path into parts (of fixed length) and keeps refining the parts by finding a path in a low level abstraction using A* until it finds any node that is abstracted by the target state. The representation scheme of PRA* is also applicable for regions of irregular shape.

Minimal-Memory (MM) abstraction [11] is a hybrid of PRA* and HPA*. An MM abstraction is built by dividing the game map into large sectors and then each sector is divided into regions. A region in the MM abstraction is an obstacle free area of the sector, so a sector can have more than one region. Each region in the abstraction is represented by a single point e.g. the center point of the region. The results of [11] show that the amount of memory required to store the abstract maps decreases as the sector size increases. The MM abstraction and refinement also finds optimal paths with the larger sector sizes. This approach has been used in the commercial game Dragon Age™ (BioWare Corp.). Our abstraction building scheme has two similarities with MM abstraction. First, it uses blocks of regular shape (i.e. rectangular) to partition a game map. Second, our method uses the center point of the regions for the online path planning. However, we use a different representation scheme and a path planning mechanism.

LRTA offers many design choices such as how to find the successors states (of a current state), how to modify the heuristic values, and which successor states' values should be updated [12]. This means there are many variations of LRTA in the literature. It is not possible to discuss all of them in this paper, therefore, we mention here the variations of LRTA that have been explored with the abstraction of game maps.

PR LRTS [3] is a hybrid of LRTS [13] and the abstraction scheme of PRA*. LRTS (Learning Real-time Search) is a variation of LRTA. LRTS makes three modifications to the simple LRTA: (i) it uses a weighted heuristic function, (ii) it introduces backtracking in LRTA by putting an upper bound on the learning amount, and (iii) it applies a max-min learning rule (to modify the heuristic value of the current state). For these three modifications, LRTS introduces three control parameters which are look-ahead depth, optimality weight and learning quota. In fact, LRTS is a combination of three other variations of LRTA which are SLA* [14], weighted LRTA [15] and γ-Trap [16].

In PR LRTS, the path planning process starts by applying LRTS at a fixed level of the abstract graph. The partial path found by LRTS is refined by Local Repair A* [17]. This refined abstract path is used to create a sub goal and a corridor for the path search (or refinement) in the next iteration. At ground level, the procedure performs execution instead of refinement. The purpose of this corridor is to keep the LRTS or A* search limited to the states which are part of the corridor. However, if the corridor is empty then search is applied on the entire graph.

The results of PR LRTS [3] are obtained on different pathfinding problems (generated over three maps) and are compared with A*, LRTA and LRTS. The results are compared using five performance measures which are convergence planning, convergence memory, first-move lag, convergence travel and suboptimality. The convergence planning shows the planning effort (in terms of CPU time and number of states touched) during the convergence process. The convergence memory is the total number of heuristic values that are stored during convergence process and first-move lag is the planning effort to make the first move. The convergence travel is the total cost of traversing all edges during the convergence process while suboptimality is the difference (in percentage) between the calculated path and the shortest-path (calculated by running the A*). The results show that A* performs the best in convergence travel and convergence memory while LRTA shows the best

performance in the first-move lag. PR LRTS (at level 3 and 2 of abstraction) with look-ahead depth of 1-ply shows the best performance in convergence planning. Suboptimality also increases with increasing the abstraction level, therefore, A* and LRTA (without abstraction) produce fewer suboptimal results than others. In our work, we apply LRTA at ground level rather than on top level hierarchies of the game map.

A variation of PR LRTS [18] uses a dynamic approach to select the look-ahead depth during the planning process. This approach uses the decision tree to dynamically determine look-ahead depth for solving a given problem.

A hybrid of an HTN-based planner SHOP [8] and the FF (Fast Forward) planner [19] has been proposed by McCluskey et al.[20] as *HyHTN*. The results of hybrid HTN are compared with two planners, SHOP and EMS [21]. The results show that *HyHTN* is faster than EMS and takes almost same time as SHOP does to solve the benchmark problems.

6 EXPERIMENTAL SETUP

Experiments are performed using the ORTS game programming environment. We have modified an existing game of ORTS called game-1 to create a testbed for our experiments. We changed the game-1 script file and its server definition such that it has only one unit with one base center and three workers. In the modified game-1, a base center and three workers are the only Non Player Characters (NPCs). We applied our model in game-1 to move the workers from one location to another on the game map. A 2D display of this game world is shown in figure 9. Base center is a static object of the game and is used as a reference point for the creation of hierarchies.

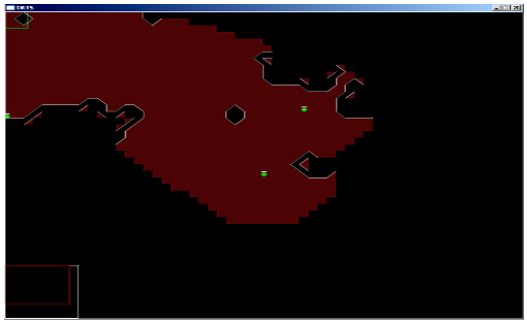

Figure 9. A map used in the modified version of game-1.

In the modified definition of the game-1, a game world has the static obstacles (i.e. hills or cliffs) only. The default position of three workers and a base center is set on the top right corner of the map for each simulation. To implement the game map hierarchies and HTN planning procedure, we have used "Sample AI project" which is a part of the ORTS game programming environment.

Three maps of 64X64 tiles are used to generate 35 pathfinding problems where each worker is given one problem to solve in a simulation. Three evaluation methods are used to compare the performance of HTN-LRTA and LRTA. These three evaluation methods are suboptimality, convergence memory and convergence cost. The suboptimality is measured using equation (3). This measure is taken from [9]. In equation (3), l_m is the length of the path found by a method and l_o is the length of the optimal path. The optimal path is calculated by using A*.

$$suboptimality = (l_m - l_o / l_o) \times 100 \quad (3)$$

The convergence memory is the total number of heuristic values saved during the planning process and the convergence cost is the total number of nodes touched during the planning process.

7 RESULTS AND ANALYSIS

The results are obtained by running ORTS game-1 simulations on three different computers of 3.0 GHz clock speed. In all simulations, both ORTS server and client applications are run on the same computer i.e. with localhost connectivity. The hierarchies of three maps are created with four different zone sizes which are 3, 6, 9 and 12. Average suboptimalities of HTN-LRTA with respect to these zone sizes are shown in figure 10.

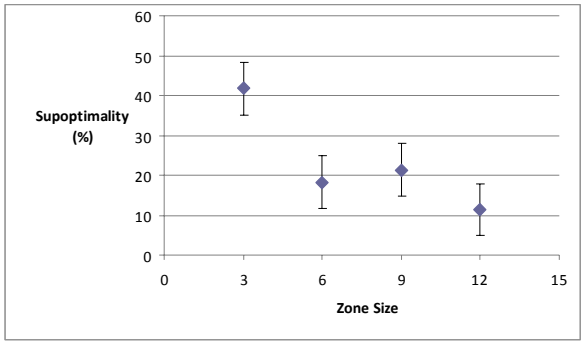

Figure 10. Suboptimality profile of HTN-LRTA with zone sizes.

The sub-optimality of HTN-LRTA is dependent on the zone size and it finds short paths with higher zone sizes. Figure 10 also reveals that HTN-LRTA finds better paths with zone size of even numbers than the paths with odd ones.

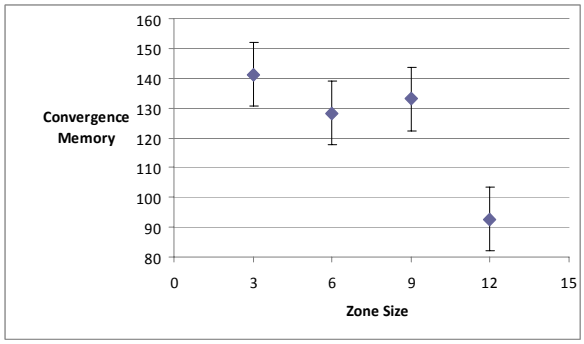

Figure 11. HTN-LRTA convergence memory Vs zone size.

HTN-LRTA consumes a very small amount of memory during the online path planning if a game map is partitioned with a larger zone size. Figure 11 shows that the memory consumption of the hierarchical task network planner remains

high when using the game map abstraction with smaller zone sizes.

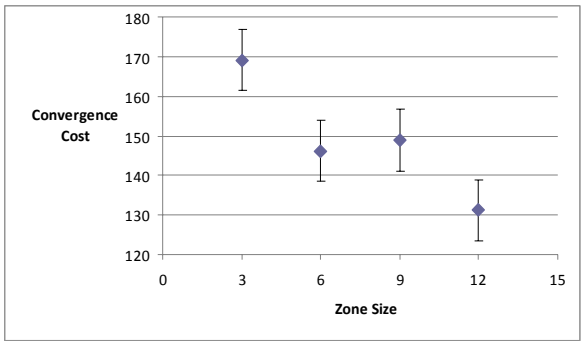

Figure 12. HTN-LRTA convergence cost Vs zone size.

Figure 12 shows that HTN-LRTA explores a small number of states if a game map abstraction is created with a high zone size. However, with small zone sizes, HTN-LRTA consumes high convergence cost. The convergence cost is directly related to the CPU usage, so HTN-LRTA with large zone size finds a path using less CPU cycles as compared to its planning with a smaller zone size. In HTN-LRTA with a higher zone size, the adjacent regions have a shorter distance between their centers than the adjacent regions of the smaller zone sizes. As LRTA gives better performance in the problems where the actual path between the start and goal states is short [12], therefore, with large zone sizes HTN-LRTA gives better performance than with small zone sizes.

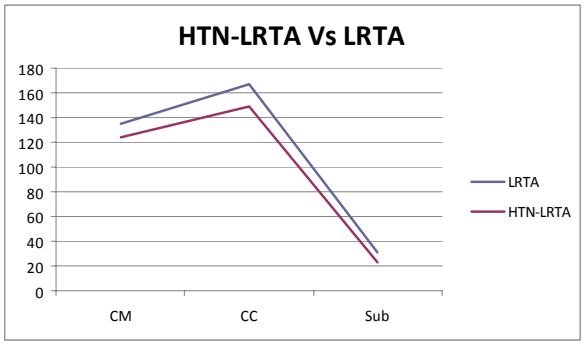

Figure 13. A comparison of HTN-LRTA and LRTA w.r.t. convergence memory (CM), Convergence Cost (CC) and sub-optimality (Sub).

The average results of HTN-LRTA over all four zone sizes are compared to the average results of LRTA for all 35 problems. This comparison is shown in figure 13. These results show a clear advantage of HTN-LRTA over LRTA with respect to all three validation parameters. On average, HTN-LRTA consumes 8% less memory than LRTA and it makes approximately 11% less search efforts (i.e. convergence cost) than LRTA. HTN-LRTA also improves the quality of the solutions and provides 25% less sub-optimal results than LRTA.

8 CONCLUSIONS

A preliminary work is presented in the direction of using an automatic hierarchical task network approach in the domain of real-time strategy games. A detailed description of the proposed model is provided and the novelty of the work has been highlighted. The results show that the effectiveness of the automatic creation of HTN depends on a user-defined parameter i.e. number of zones. The number of zones directly influences the processing time, memory consumption and the quality of the solution. The proposed hierarchical approach achieves a better performance than the LRTA on average.

As a future work, the hierarchical model can be modified to automatically determine an appropriate number of zones for a given map using an evolutionary approach. We also aim at using the hierarchical information for tuning the heuristic function of the path planner. In future experiments, the proposed model would be studied with the dynamic settings of the game i.e. the enemies and other movable obstacles in a game map. The future work also includes an empirical analysis of the proposed hierarchical approach with respect to the existing relevant approaches.

REFERENCES

[1] S. Koenig. Agent-Centered Search. *Artificial Intelligence Magazine*, 22(4): 109-131 (2001).

[2] M. Ghallab, D. Nau and P. Traverso. *Automated Planning and Practice*. Morgan Kaufmann Publishers, Elsevier (2004).

[3] V. Bulitko, N. Sturtevant, J. Lu and T. Yau. Graph Abstraction in Real-time Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)*, 30(1): 51-100 (2007).

[4] R. Korf. Real-Time Heuristic Search. *Artificial Intelligence*, 42(2-3): 189-211 (1990).

[5] T. Ishida. Moving Target Search with Intelligence. In: *Procs. 10th National Conference on Artificial Intelligence (Problem Solving: Search and Expert Systems) AAAI Press*, pp. 525-532 (1992).

[6] P.E. Hart, N.J. Nilsson and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2): 100-107 (1968).

[7] M. Buro. Real-Time Strategy Games: A New AI Research Challenge. In: *Procs. International Joint Conference on AI*, (2003).

[8] D. Nau, Y. Cao, A. Lotem and H. Muñoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In: *Procs. 16th International Joint Conference on Artificial Intelligence,* 2:968-973 (1999).

[9] A. Botea, M. Müller and J. Schaeffer. Near Optimal Hierarchical Path-Finding. *Journal of Game Development*, 1(1): 7-28 (2004).

[10] N.R. Sturtevant and M. Buro. Partial Pathfinding Using Map Abstraction and Refinement. In: *Procs. 20th National Conference on Artificial Intelligence AAAI Press*, pp. 1392-1397 (2005).

[11] N.R. Sturtevant. Memory-Efficient Abstraction for Pathfinding. In: *Procs. AIIDE-2007*, June 6-8, Stanford, California,USA, pp.31-37 (2007).

[12] S. Koenig. A Comparison of Fast Search Methods for Real-Time Situated Agents. In: *Procs. 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2: 864-871 (2004).

[13] V. Bulitko and G. Lee. Learning in Real-Time Search: a Unifying Framework. *Journal of Artificial Intelligence Research (JAIR)*, 25(1): 119-157, (2006).

[14] L-Y. Shue and R. Zamani. An Admissible heuristic search algorithm. *Lecture Notes in Computer Science (LNCS)*, *(procs. 7th International Symposium on Methodologies for Intelligent Systems)*, 689(1993): 69-75 (1993).

[15] M. Shimbo and T. Ishida. Controlling the learning process of real-time heuristic search. *Artificial Intelligence*, 146(1): 1-41 (2003).

[16] V. Bulitko. Learning for Adaptive Real-Time Search. *Tech Report. http://arxiv.org/abs/cs/0407016, Computer Science Research Repository (CoRR) (*2004*)*, accessed date: 14th Nov, 2009.

[17] B. Stout. Smart Moves: Intelligent Pathfinding. *Game Developer Magazine* (1996).

[18] V. Bulitko, M. Luštrek, J. Schaeffer, Y. Björnsson and S. Sigmundarson. Dynamic Control in Real-Time Heuristic Search. *Journal of Artificial Intelligence Research (JAIR),* 32(1): 419-452 (2008).

[19] J. Hoffmann. A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-Climbing Algorithm. *Lecture Notes in Computer Science (procs. 12th International Symposium on foundations of Intelligent Systems)*, 1932: 216-227 (2000).

[20] T.L. McCluskey, D. Liu and R.M. Simpson. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In: *Procs. ICAPS-2003*, Trento, Italy, pp. 92-101 (2003).

[21] T.L. McCluskey. Object Transition Sequences: A New Form of Abstraction for HTN Planners. In: *Procs. 5th International Conference on Artificial Intelligence Planning Systems (AIPS-2000)*, pp. 216-225 (2000).