Defence Science Journal, Vol. 57, No. 5, September 2007, pp. 721-732

Move Table: An Intelligent Software Tool for Optimal Path Finding and Halt Schedule Generation

Anupam Agrawal, Anugrah Joshi, and M. Radhakrishna

Indian Institute of Information Technology, Allahabad-211 011

ABSTRACT

This study aims to help army officials in taking decisions before war to decide the optimal path for army troops moving between two points in a real world digital terrain, considering factors like traveled distance, terrain type, terrain slope, and road network. There can optionally be one or more enemies (obstacles) located on the terrain which should be avoided. A tile-based A* search strategy with diagonal distance and tie-breaker heuristics is proposed for finding the optimal path between source and destination nodes across a real-world 3-D terrain. A performance comparison (time analysis, search space analysis, and accuracy) has been made between the multiresolution A* search and the proposed tile-based A* search for large-scale digital terrain maps. Different heuristics, which are used by the algorithms to guide these to the goal node, are presented and compared to overcome some of the computational constraints associated with path finding on large digital terrains. Finally, a halt schedule is generated using the optimal path, weather condition, moving time, priority and type of a column, so that the senior military planners can strategically decide in advance the time and locations where the troops have to halt or overtake other troops depending on their priority and also the time of reaching the destination.

Keywords: Digital terrain, classification maps, digital elevation maps, path finding, halt schedule, Move Table software, genetic-fuzzy approach

1. INTRODUCTION

Path finding has many applications ranging from path finding in a network, computer games, and to help robots navigate through an environment. However, there is less research done on path finding in digital terrains represented as digital elevation maps (DEM) and classification maps. Jonsson[1] has done some work on path finding for vehicles in real world digital terrains by focusing on terrain types, which affect a vehicle's speed, and on avoidance of enemy units on the terrain[1]. Wichmann[2] has done some work on the problem of finding an optimal path connecting two points in a digital approximation of a large-scale real world terrain[2].

There are two different path planning problems. The first problem is how to plan a path when the locations of all the enemies (obstacles) are known. The solution to this problem gives the optimal path to follow before the object actually follows the ground. It is referred to as the global path planning case. The planned path is followed in a static environment. Solutions to the global path planning case would provide initial strategies in any planning problem[3]. In the second problem, the locations of the obstacles are not known in advance. Instead, the locations of the obstacle points are known to the object when these are within a sensor range. The object changes its path when it senses the danger areas. It is referred to as the dynamic path

Received 4 July 2006, revised 4 January 2007

planning case. In this case, a path is created for a dynamic environment. This corresponds to the real-world situation of a robot with only a limited knowledge of its environment navigating through a maze. Solutions to both the cases are different. In the global path planning case, one does not have to worry about the time it takes to formulate the plan. This is because one would generate his solution before following it. The dynamic path planning case is more time critical because one has to constantly update ones plan while executing it. Thus, one has the distinction of creating a real-time planner versus a non-real-time planner. Another distinction between the two cases is decidability. In the dynamic case, depending on how one constructs his algorithm,one may not be able to conclude whether a safe path exists. This is a direct result of the changing environment. However, in the global case, one can construct an algorithm that can determine the existence of a safe path[3].

The success of a given path finding technique depends on the requirements and the assumptions of the environment and the constraints it imposes. There are a wide variety of path finding techniques, and one technique may excel under certain circumstances, but do badly under others. The more factors that intend to incorporate into the environment, the more complex the path finding becomes.

In this study, the scope is limited to the special case of finding paths in Euclidean 3-D space, focusing on movements along a surface that can be projected onto a 2-D grid. Furthermore, it discusses the global path planning case, where the environment is static and the obstacles are not moving. We will develop and analyse different heuristics and variations of the A* algorithm to solve the optimal path problem under a variety of user-defined constraints. The A* algorithm is used as it has an optimal heuristic search. The search space, on which the path finding takes place, can potentially be quite large.

For dealing with large-scale images or raster data sets, a tile-based approach is proposed, in which a raster data set is divided into appropriate number of tiles. Instead of working on the entire raster data set, only those tiles that are needed are loaded and used. It results in the reduction of search space required for path finding process with moderate search time and the resulting solution being still optimal.

Using the output of the above optimal path finding algorithm, the proposed halt schedule algorithm generates a schedule in the form of a report. This report contains the reaching time of different troops, which may start from different locations, at important path points so that there should be no collision among the moving troops. It is helpful for senior military planners to decide in advance the time and locations where the troops have to halt or overtake other troops depending on their priority and time of arrival. This schedule will also generate the time of reaching the destination for different troops via optimal path.

2. RELATED WORK

One of the earliest solutions proposed for path finding is Dijkstra's algorithm which finds the shortest paths to all other nodes in the search space as opposed to finding the shortest path to a single goal node. Dijkstra's algorithm always visits the closest unvisited node from the starting node, and hence, the search is not guided towards the goal node[1, 4].

In contrast, ‘best first search’ always selects the node closest to the goal node. Since one does not know the exact path from the current node to the goal node, the distance to the goal node has to be estimated. This estimate is referred to as the heuristic. Best first search does not keep track of the cost to the current node, and therefore, does not necessarily find an optimal solution[5,6].

In Weighted graph method, space is divided into discrete regions, called cells, and movement is restricted from a particular cell to its neighbours. Neighbouring cells are those that can be directly reached from a particular cell. A directed graph is constructed by taking the cells as graph vertices and the possible movements to its neighboring cells as directed edges between the vertices. A weight function is defined by assigning a cost to every edge, corresponding to the cost of moving along the edge. The space division, the definition of neighbours and the edge cost function can all differ for different

methods in this class. Disadvantage of these methods is that for a real-world digital terrain, search space will be very large[1].

The A* algorithm combines the approaches of Dijkstra's algorithm and best first search. The A* algorithm is guaranteed to find an optimal solution (assuming no negative costs and an admissible heuristic), but because it is guided towards the goal node by the heuristic it will not visit as many nodes as Dijkstra's algorithm would do. Hence, it is preferred over other algorithms.

To reduce the memory and time requirements of path finding in large-scale terrains, size of the search space is reduced by computing a multiresolution representation of the terrain[2]. First a path is found on the lowest-resolution version of the digital terrain (which has the least nodes, and hence, allows a fast search) and then each pair of nodes of the resulting solution are used as start and goal nodes for a search on the next higher-resolution representation of the terrain. This process is repeated until one obtains a path for the original terrain. Sub-sampling factor is used to refer to the reduction of size of a low-resolution terrain when compared to its full resolution version. Low-resolution representations of the digital terrain are computed using a mean or a median operator. The mean operator just averages all the height values of a group of cells and does not work well for groups of cells that contain a ridge or sudden drop-off. Once the path on the sub-sampled terrain map has been found, searches between the path points are performed. Following equations are used for coordinate transformation from the low-resolution image to the higher- resolution image.

$$x_h = x_{h\min} + (x_l - x_{l\min}) * s_x$$

$$y_h = y_{h\min} + (y_l - y_{l\min}) * s_y$$

where, (x_h, y_h) are the coordinates in the high resolution image corresponding to the point (x_l, y_l) in the low-resolution image. The variables s_x and s_y are the sub-sampling factors. $(x_{l\min}, y_{l\min})$, $(x_{l\max}, y_{l\max})$ and $(x_{h\min}, y_{h\min})$, $(x_{h\max}, y_{h\max})$ are the minimum and maximum values of x and y coordinates in the low- and high-resolution representations of the image, respectively.

Dijkstra's algorithm is a greedy algorithm, its search space is very large and also the search is not guided towards the goal node. In case of best first search, the search is guided and its search space is small. But it does not give an optimal solution. In case of A* search, the search space, on which the path finding takes place, can potentially be quite large. This means that steps must be taken to ensure that one does not run out of memory or that the search does not take an excessive amount of time to run in large-scale terrains. In case of multiresolution A* search, the quality of a solution usually decreases with an increasing sub-sampling factor. The greater the sub-sampling factor, the more cells get merged into one cell, thus making it easier to lose fine terrain structures, e.g., a narrow passage between two hills. Also the use of large cells can cause a magnification of bends ('detours') in a path which can not be removed in the final iteration of the algorithm since the way points found during the low-resolution passes are fixed. Large sub-sampling factors decrease the number of way points along the path, thus increasing the distance between these on the original terrain map. This means that one is running the path finding algorithm less often on the original terrain map[2].

3. HEURISTICS

Heuristic is the estimated cost of traveling from current node to the destination. Different heuristics are used for finding the optimal path in different situations. Choosing a heuristic affects the speed and accuracy of the pathfinder. On a grid, there are well-known heuristic functions to use. The Manhattan distance heuristic [Fig. 1(a)] is

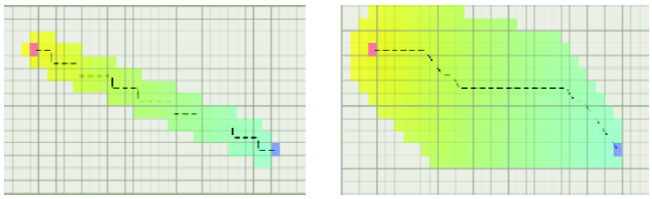
$$h(n) = |x_a - x_b| + |y_a - y_b|$$

where (x_a, y_a) and (x_b, y_b) are the coordinates of the current and goal nodes, respectively. The heuristic is ideal when using 4-adjacency. The Euclidean distance heuristic [Fig. 1(b)] is defined as

$$h(n) = \sqrt{[(x_a - x_b)_2 + (y_a - y_b)^2]}$$

It is admissible, but usually underestimates the actual cost by a significant amount. This means

that we may visit too many nodes unnecessarily, which in turn increases the time it takes to find the goal. In Figs 1(a) and 1(b), the pink square is the starting point, the blue square is the goal, and the shaded areas show what areas the above two heuristics scanned.



(a) (b)

Figure 1. (a) Manhattan distance, and (b) Euclidean distance.

To avoid the expensive square root, Euclidean distance squared heuristic [Fig. 2(a)] is defined as

$$h(n) = [(x_a - x_b)^2 + (y_a - y_b)^2]$$

The diagonal distance heuristic [Fig. 2(b)] is defined as

$$h(n) = |x_a - x_b| + |y_a - y_b| + (\sqrt{2} - 2)\min(|x_a - x_b|, |y_a - y_b|)$$

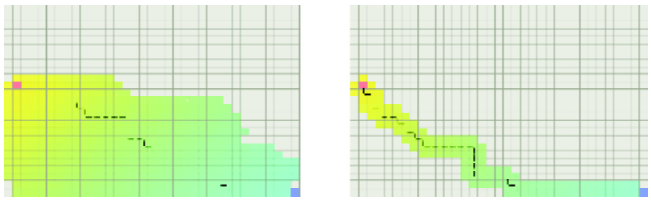
The above heuristic combines aspects of both the Manhattan and Euclidean heuristics and is admissible (unless 16-adjacency is used). It has the advantage of always giving the actual minimum possible cost to the goal if 8-adjacency is used and taking the square root is no longer necessary, thus making it computationally slightly more efficient than the Euclidean distance heuristic.

One thing that can lead to the poor performance of path finding is ties in the heuristic. When several paths have the same *f* value, they are all explored, even though one only needs to explore one of them. To break ties, paths along the straightline from the starting point to the goal are preferred. A vector cross product between the start to goal vector and the current point to goal vector is computed. When these vectors don't line up, the cross product will be larger. The result is that this code will give slight preference to a path that lies along the straightline path from the start to goal. When there are no obstacles, A* not only explore less of the map, the path looks very nice[5] [Fig. 3]. The tie breaker algorithm is as follows:

$$dx_1 = \text{current}.x - \text{goal}.x$$
$$dy_1 = \text{current}.y - \text{goal}.y$$
$$dx_2 = \text{start}.x - \text{goal}.x$$
$$dy_2 = \text{start}.y - \text{goal}.y$$
$$\text{cross} = \text{abs}\ (dx_1 * dy_2 - dx_2 * dy_1)$$
$$\text{heuristic} + = \text{cross} * 0.001$$

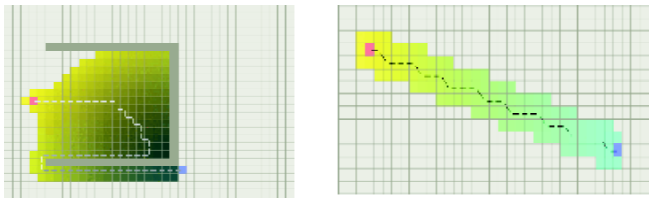
where, cross => vector cross-product between the start to goal vector and the current node to goal vector.

A*'s ability to vary its behaviour based on the heuristic and cost functions can be very useful. The trade–off between speed and accuracy can be exploited to make the pathfinder faster.



(a) (b)

Figure 3. (a) Ties in *f* values, and (b) after applying tie breaker.



(a) (b)

Figure 2. (a) Euclidean distance, and (b) diagonal distance squared

4. PROPOSED METHODOLOGY

This work is divided into two phases: (i) Finding the optimal path, and (ii) halt schedule generation.

First phase will find the optimal path between source and destination nodes in large satellite images and raster data sets using improved A* search algorithm, considering different factors like terrain type, terrain slope, road map, obstacles, and distance.

Input: Original satellite image, height map of the image, classified map of the image, road map of the image, layer representing the obstacles, starting location, and goal location.

Output: Optimal path (a linked list of contiguous *XY* coordinate pairs) from the starting location to the goal location.

4.1 Cost Function: Considering Road Maps

If a road network exists between the two nodes then the optimal path is generated considering only the distance and terrain slope heuristics. Pixel value of a road point is taken to be 0 and for all other points in the road raster map pixel value is taken to be 255.

Cost function is computed as follows:

$$f(v) = g(v) + h(v)$$

where, $f(v)$=> total cost of traveling from source to destination, $g(v)$=> exact cost of traveling from source to node v, and $h(v)$=> estimated cost of traveling from node v to destination.

The details of functions $g(v)$ and $h(v)$ are given below:

$$g(v) = g(u) + w(u,v)$$

where, $g(u)$ is the movement cost from the starting node to u, and $w(u,v)$ which is cost of travelling from node u to v is defined as follows:

$$w(u,v) = \text{pix}(v) + \text{slope}(u,v)$$

where, pix(v) is the pixel value at that point in the road map and slope(u,v) is the slope between u and v.

$$h(v) = h(\text{tie},v) + h(\text{tie},v) * h(\text{slope},v)$$

where, h(tie,v) is diagonal distance tie breaker heuristic and h(slope,v) is average slope difference.

4.2 Cost Function: Without Considering Road Maps

If a road network does not exists between the two nodes then the optimal path is generated considering traveled distance, terrain type, terrain slope and obstacles heuristics.

Cost function is computed as follows:

$$f(v) = g(v) + h(v)$$

$$g(v) = g(u) + w(u,v)$$

$$w(u,v) = d(u,v) + t(u,v) + \text{slope}(u,v) + o(u,v)$$

where, $d(u,v)$ is distance between u and v; $t(u,v)$ is terrain type cost at v; slope(u,v) is slope between u and v; and $o(u,v)$ is obstacle cost, have higher values for more difficult obstacles.

$h(v)$ consists of four parts:

$$h(v) = h(\text{tie},v) + h(\text{tie},v)*\{h(t,v) + h(\text{slope},v)\} + h(o,v)$$

where, h(tie,v) is the diagonal tie breaker heuristic; $h(t,v)$ is the average terrain cost; h(slope,v) is the average slope difference; and $h(o,v)$ is the n/N* cost of obstacles, where n is the number of obstacle points and N is the total number of points.

4.3 Proposed Tile-based Optimal Path Finding Algorithm

For large-scale terrain maps, the entire raster data set is not used for path finding; only required portion of it is used. The tile-based approach divides the whole raster data set into multiple tiles of appropriate size. During path finding, the tiles are managed using an efficient indexing scheme. Initially, the tile with source node is loaded and path points are found considering the location of destination node until path finder reaches the boundary of the current tile. Now the neighbouring tile is loaded using the indexing scheme and path finding process continues on that tile.

During the path finding process, if the pathfinder finds a road point then it follows the road using the cost function given in Section 4.1 until the number of road points traversed (road_count) is greater than a threshold. After that, it leaves the road at a point which is nearest to the destination (having min h value) and then again follows the terrain using the cost function given in Section 4.2 towards the destination.

Linked list RoadList stores all the road points that are neighbours of the current node visited, TerrainList stores all the terrain points that are neighbours of the current node visited, TempList stores all the terrain points that are the neighbours of the current road point, CloseList contains all the currently visited nodes and FinalList contains the nodes in the optimal path.

Function get_min_node(LIST) returns a node with minimum *f* value from the LIST and the function get_min_*h*(LIST) returns a node with minimum *h* value from the LIST. The variable search_count represents the search space and road_count represents the number of road points traversed.

Algorithm

```
TileBased_Astar ( start, end )
  {
    search_count=0;
    road_count=0;
    found = false;
    Load initial tile in which source node lies;
    if start node is a road point then insert it
    into RoadList;
    else insert it into the TerrainList;
    While(found == false) // path not found
     {
      curr = get_min_node(RoadList);
if(curr == null)
       {
        curr1=get_min_node(TerrainList);
        if(curr1==null)
           {
              curr1 = get_min_h(TempList);
             if(curr1== null)
                {
                print("No path found "); exit;
                }
           }
  curr = curr1;
       }
 else
    {
          if (road_count>threshold)
    {
      curr1= get_min_h(TempList);
      if(curr1!=null)
          {curr=curr1;
           clear RoadList;
           clear TempList;
           road_count=0;
          }
      }
      }
      if(curr node's tile != current tile loaded)
      {
          Load curr node's tile;
          Current tile = curr node's tile;
      }
      CloseList = insert_node(CloseList,curr);
      if(curr == end)
      {
           found = true;
           print("Path Found "); break;
      }
if(curr is a road point)
     {
     clear TerrainList;
     opt_path_road(curr); //defined below
     road_count++;  search_count++;
     }
else
    {
    clear RoadList;
    opt_path_terrain(curr);//defined below
    search_count++;
    }
     }// end while
 starting from the curr node  traverse CloseList
 until we get the start node;

 store all the nodes in a list FinalList;
 return FinalList;
}
opt_path_road ( node current )
{
     consider the 8 neighbors of the curr node,
     create a node for each  neighbor, apply cost
     function and calculate f, g and h values;

 if (neighbor does not belong to the current tile)
    {
     Load appropriate tile;
```

```
      Create a node for that neighbor, apply
      cost function and calculate f, g and h values;
      }
  for (int i=0; i<8; i++)
     {
      if(neigh[i] is already present in RoadList as
     node
           temp)
           {
            if ( neigh[i].fval< temp.fval)
            temp = neigh[i];
           }
else
          {
          if(neigh[i] is a road point)
          insert it into RoadList;
          if(neigh[i] is not a road point)
          insert it into TempList;
           }
      }//end for
}
     opt_path_terrain ( node current )
      {
```

Similar to opt_path_road(node current) described above, with neighbor terrain point is inserted into TerrainList and road point is inserted into RoadList.

```
       }
```

4.4 Proposed Halt Schedule Generation Algorithm

The halt schedule is generated using the optimal path returned by the path finding module, weather condition, priority, average size of a column (vehicle) and moving time so that the senior military planners can decide in advance the time and locations where the troops have to halt or overtake other troops depending on their priority and also the time of reaching the destination. The halt schedule is generated in the form of a report and can be shown graphically.

Input: Computed optimal path, troop Id, troop priority, starting date and time, starting location, destination, number of vehicles in a troop, dimension of each vehicle, special instructions (move during day/night), and weather condition.

Output: Output will be generated in the form of a report that will contain Halt Schedule (time and location at which the troops will halt) and reaching time at the destination.

Algorithm

Halt_Schedule_Gen ()

{

(1) One is getting the optimal path from source to destination in which the coordinates of all the points are known and this path is computed using a cost function;

(2) According to the given input, the program will generate the schedule for the first troop in terms of locations, time and distance of halting;

(3) This schedule will then be stored in a table and used for generating other troop schedules;

(4) When the schedule for some other troop is to be generated then it will take the following steps:

   (a) First it will check whether there is a common path or not among the current troop and the troops in the table;

   (b) If yes, then it will check the timing at which the two troops will arrive on that common path calculated on the basis of distance and speed of the vehicles;

   (c) If there is any overlap in time then it will check the priority of the troops. The higher priority troop will be allowed to pass first and the schedule of both the troops will be changed and stored in the table;

   (d) The lower priority troop can either halt before reaching over the common path or the troop will have to be delayed at the starting location itself.

(5) Above steps will be repeated for generating the schedules of other troops also;

(6) When a troop will reach to its destination, then its entry will be removed from the database;

(7) At the end it will generate the report and can be stored in a file;

(8) The halt schedule will also be displayed graphically in the map.

}

5. IMPLEMENTATION

The modules are written in Java Net Beans IDE[7]. The software has been tested on Dehradun, India, and Grand Canyon, Arizona data sets.

5.1 Tools and Software Used

Data Preparation: Adobe PhotoShop, Photo Impact 4.0, PCI Geomatica 9.0

Classification: ERDAS Imagine 8.7

Java Platform and Development Environment (for path finding and halt schedule generation): Net Beans IDE and J2SDK 1.4.2

5.2 Database Used

Halt Schedule Generation: MS Access

Database consists of the following relations:

Path (troop _id, xcoord, ycoord) => stores x and y coordinates of the optimal path for the troop identified by troop_id.

Troop_Detail (troop_id, troop_priority, no_vehicles, avg_leng_vehicle, move_night weather_cond, starting_date, starting_time, avg_speed_troop) => stores the details of the troop and travel condition.

Temp (xcoord, ycoord, troop_id, no_vehicles, reaching_date, reaching_time, troop_priority, avg_speed_troop, avg_len_vehicle) => Temporary table used for generating the schedule and contains those troops that overlap both in path and time with the current troop.

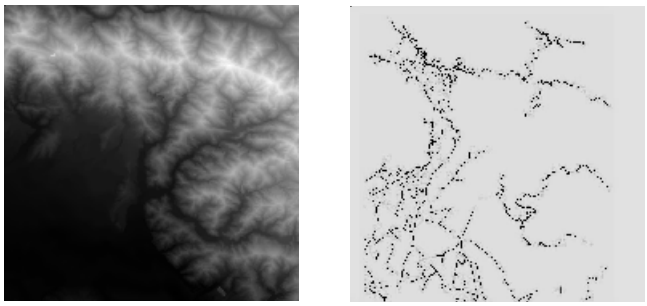
Halt_Schedule (troop _id, xcoord, ycoord, reaching_date, reaching_time) => stores the schedule generated for each point in the optimal path for all the troops.

Final_Schedule (troop_id, halt_xcoord, halt_ycoord, reaching_day, reaching_time, halt_time) => stores the final halt schedule generated.

6. RESULTS AND ANALYSIS

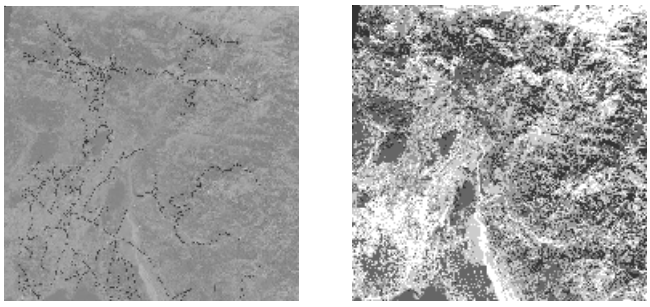
The geocoded IRS-1D LISS III multispectral satellite image of Dehradun (February 2003) and its height map or DEM are used for optimal path finding. The height map has been generated using digitised contours on 1:50,000 scale toposheet (number 53 J/3) [Fig. 4(a)]. Dehradun road map [Fig. 4(b)], digitised from above toposheet, is merged with the original image using Photo Impact package [Fig. 5(a)]. Classified map with 5 classes is generated using ERDAS Imagine 8.7 unsupervised classifier [Fig. 5(b)]. After consulting the toposheet, the classified image pixels were labeled as forest (vegetation), mountains (rocks), residential areas (buildings), dry river path (sand) and unknown class (remaining unlabeled pixels). These classes are assigned different cost values to be used in the cost function to compute the optimal path.

Highest priority is given to the road. If during path finding process, pathfinder finds a road point,



(a) (b)

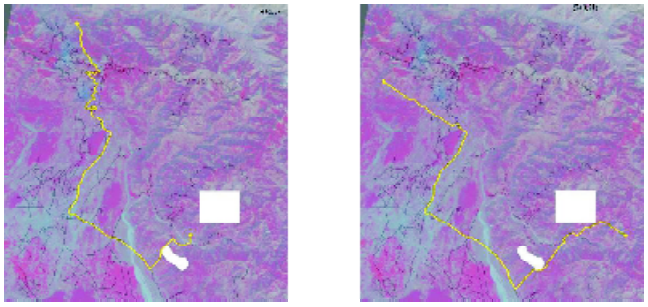
Figure 4. (a) Height map, and (b) road map.



(a) (b)

Figure 5. (a) Merged image, and (b) classified map.

then it follows the road and leaves the road at a point nearest to the destination, after that, terrain is followed. In the terrain, pathfinder avoids any obstacle and moves in the direction of least cost [Figs 6(a)-6(b)]. The white patches in the image show obstacle regions and should be avoided.



(a) (b)

Figure 6. (a), (b) Pathfinder avoiding hills, following road, avoiding obstacles and following the terrain.

In multiresolution A* Search, first a path is found on the lowest resolution version of the terrain (256BX256B) [Fig. 7] and then each pair of nodes of the resulting solution are mapped into the higher

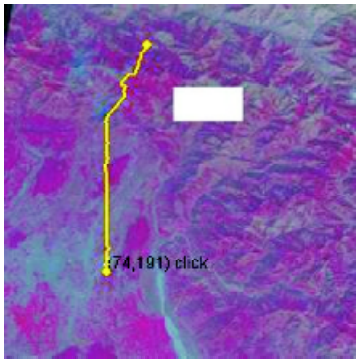


Figure 7. Lower resolution path, not avoiding a narrow obstacle.

resolution representation of the terrain (512B X 512B) as a start and goal node for a search. This process is repeated until a path is obtain in the original terrain [Fig. 8]. But it has the disadvantage of loosing narrow terrain structures, like narrow obstacles resulting in a near-optimal path.

But in the tile-based approach, only those tiles are loaded and used that are needed, resulting in an optimal path [Figs 9(a)-9(d)]. Table 1 presents

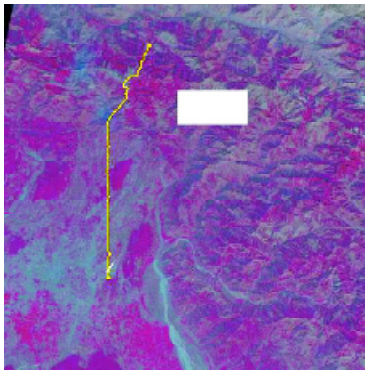
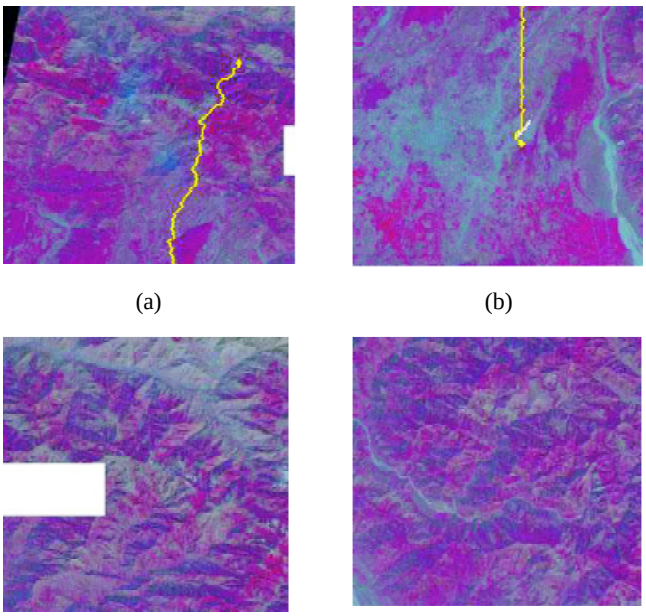


Figure 8. Near-optimal path in high resolution representation (not avoiding the narrow obstacle).

the comparison of results obtained using both the methods.

6.1 Time Analysis

The average time taken by multiresolution approach is much lesser then the tile-based approach as the search space greatly reduces in the lower resolution image. In the higher resolution representation of the image, if the sub-sampling factor is less, then the numbers of nodes between every pair of mapped



(a) (b)

(c) (d)

Figure 9. (a) Tile11, contains source, (b) Tile12, not loaded, (c) Tile21, contains destination, and (d) Tile22, not loaded.

Table 1. Comparison of results from both multiresolution and tile-based approach

	Source	Destination	Time analysis (s)	Search space	Accuracy
Multi resolution (Sub-sampling factor= 2) approach	204, 352	334,160	0.126	421	Near-optimal
	108, 380	30, 138	0.265	595	Near-optimal
	276, 460	406, 264	1.297	1021	Near-optimal
	172, 284	386, 32	0.203	485	Near-optimal
Tile-based approach	204, 352	334,160	1.406	248	Optimal
	108, 380	30, 138	1.14	301	Optimal
	276, 460	406, 264	3.891	747	Optimal
	172, 284	386, 32	1.984	307	Optimal

path points are less, which reduces the total time taken in finding the optimal path (Table 1).

6.2 Search Space Analysis

The search space of the tile-based method is moderate (Table 1). In the tile-based approach only the tiles that are needed are loaded and only the image at a given resolution is used for finding the path points and no mapping of points is involved. But for large-scale raster data sets the search space of the tile-based approach will be larger than the multiresolution approach.

6.3 Accuracy

The tile-based method always returns an optimal path, while the multiresolution method returns a near-optimal path [Table 1]. If there is any narrow obstacle or passage in the original image then it may be possible that the corresponding low-resolution image does not contain it and the path obtained using multiresolution algorithm will not be optimal for the highest resolution image.

7. CONCLUSION

An improved solution to the problem of optimal path finding has been found. Evaluation of a test implementation has shown that it satisfies the objectives. One possible optimisation for real world digital terrain maps that are known to contain roads would be to first search paths from the start to the nearest road and from the nearest road to the goal. This has been implemented here.

The proposed tile-based A* approach helps in finding the optimal path with moderate search space and search time for large real world digital terrain maps. Furthermore, this approach gives more optimal path (avoiding narrow obstacles) as compared to the multiresolution A* approach.

The halt schedule generation module generates the halt schedule consisting of locations where the troops have to halt to avoid collision with other troops. A minimum time difference is maintained between the two troops following a common optimal path.

During the various stages of the method, many approximations have been made. There is a very low probability that the worst case behaviour occurs in the real world, so the practical results seem to be satisfactory. The memory usage and speed objectives have been met and implementation complexity is relatively low.

8. FUTURE SCOPE

There are a number of issues that could be further investigated. Here are a few thoughts that might be worthwhile to examine in greater depth:

8.1 Dealing with Moving Obstacles

One of the shortcomings of the algorithm is its inability to deal with dynamic obstacles. If any input parameter changes a complete new search must be done. Thus, moving obstacles are not handled in the proposed approach. It is probably one area

where a more effective algorithm could be most advantageous for performance[8].

8.2 Alternative Search Techniques

A genetic-fuzzy combination can be used in path finding where the performance of a fuzzy logic controller is improved by using a genetic algorithm[9]. The use of fuzzy logic technique helps in determining imprecise but obstacle-free paths and the use of a genetic algorithm helps in mining an optimal set of rules that can be used in finding the optimal path. Hence, in genetic-fuzzy approach, a genetic algorithm is used to find optimised Fuzzy logic controller which is used on-line, to solve the path planning problem for real world digital terrain.

ACKNOWLEDGEMENTS

The authors are thankful to WARDEC, Ministry of Defence, New Delhi, officials for discussing and explaining the requirements related with the Move Table software tool development. The authors express their sincere gratitude to Director, Indian Institute of Information Technology, Allahabad, for providing excellent facilities and environment for research.

REFERENCES

1. Jonsson, F.M. An optimal pathfinder for vehicles in real world digital terrain maps. The Royal Institute of Science, Sweden, 1997. Master's Thesis.

2. Wichmann, Daniel R. & Wuensche, Burkhard C. Automated route finding on digital terrains. *In* Proceedings of IVCNZ '04, 21-23 November 2004, Akaroa, New Zealand, 2004. pp. 107-12.

3. Stentz, Anthony. Optimal and efficient path planning for partially-known environments. *In* Proceedings of IEEE International Conference on Robotics and Automation, May 1994, San Diego (USA), 1994. pp. 3310-317.

4. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L. & Stein, C. Introduction to algorithms, Ed. 2. MIT Press, 2001.

5. Patel, Amit J. Amit's thoughts on path finding and A-Star, 2003. http://theory.stanford.edu/~amitp/GameProgramming/

6. Russell, Stuart J. & Norvig, Peter. Artificial intelligence: A modern approach, Ed. 2. Pearson Education, 2003.

7. Rodrigues, Lawrence H. Building imaging applications with java technology using AWT imaging, java 2D, and java advanced imaging (JAI). Addison Wesley Professional, 2001.

8. Stentz, Anthony. The focussed D* algorithm for real-time replanning. *In* Proceedings of International Joint Conference on Artificial Intelligence, August 1995, Montreal, CA, 1995. pp. 1652-659.

9. Roy, S.S. & Pratihar, D.K. A genetic-fuzzy approach for optimal path-planning of a robotic manipulator among static obstacles. *Inst. Engg. (India) J.*, May 2003, **84**, 15-22.

Contributors

Mr Anupam Agrawal received his MS (Computer Science) from the J.K. Institute of Applied Physics and Technology, University of Allahabad in 1988 and MTech (Computer Sc & Engg) from the IIT Madras, Chennai, in 1995. He is presently working as Assistant Professor at the Indian Institute of Information Technology, Allahabad. Earlier, he was working as Scientist D at the DEAL, Dehradun. His research interests include: Real-time 3-D graphics, computer vision, artificial intelligence and soft computing, data mining, GIS, remote sensing, and image processing. He has more than 30 research papers to his credit.

Mr M. Radhakrishna received his MSc (Nuclear Physics) from the Andhra University in 1962. Currently, he is Advisor and Professor at the Indian Institute of Information Technology, Allahabad. He is also Technology Advisor to Aptech, Mumbai. His research interests include: Artificial intelligence, automation, cognitive sciences, computer graphics and image processing, modelling and simulation, and computer networks. He has published more than 60 papers.