

Automated Route Finding on Digital Terrains

D. R. Wichmann and B. C. Wünsche

Graphics Group, Dept. of Computer Science, University of Auckland, Private Bag 92019, Auckland,
New Zealand.

daniel.wichmann@gmail.com, burkhard@cs.auckland.ac.nz

Abstract

Path planning is an important problem in Artificial Intelligence and Robotics. In this paper we examine the problem of finding an optimal path for a road connecting two points in a digital approximation of a real terrain. Additional constraints, such as terrain gradients, will be introduced in order to achieve more realistic roads. A performance comparison will be made between the standard A* algorithm and variations of it. Different heuristics, which are used by the algorithms to guide them to the goal node, are presented and compared. To overcome some of the computational constraints associated with route finding on large digital terrains we introduce multi-resolution searching. We will show that multi-resolution searching greatly reduces the time associated with the route finding process while still resulting in near optimal solutions.

Keywords: Digital terrains, route finding, path planning, terrain visualisation

1 Introduction

Path finding has many applications reaching from computer games to helping robots navigate through an environment. However, the literature offers little research on route finding on terrains represented as digital elevation maps (DEM). F. Markus Jönsson [1] previously looked at 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. We present a tool for finding plausible roads in a digital terrain; however, the principles presented can also be used for routes other than roads (e.g. railways, walking tracks).

The literature describes many algorithms for finding the shortest path between two points. One of the earliest solutions proposed is Dijkstra's algorithm [2] 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. In contrast Best First Search [3] always selects the node that is closest to the goal node. Since we do 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. The A* search algorithm was first introduced in 1968 [4] and is still widely used

today, especially in the interactive entertainment industry. 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. This reduces both memory and time requirements. Amit J. Patel [6] presents a good comparison between the A* algorithm, Dijkstra's algorithm and Best First Search.

In this paper we develop and analyse different heuristics and variations of the A* algorithm in order to solve the shortest path problem under a variety of user defined constraints. The main reason for using the A* algorithm is that it is optimal and a heuristic search. It also performs better than other search strategies in many cases [6]. The search space, on which the road finding takes place, can potentially be quite large. This means that we must take steps to ensure that we do not run out of memory or that the search takes an excessive amount of time to run. Even a relatively small terrain map of 300 by 300 pixels has a search space of 90,000 nodes. Two possible approaches to reduce these problems are a reduction in the size of the search space by using multi-resolution terrains and modifications to the search algorithm.

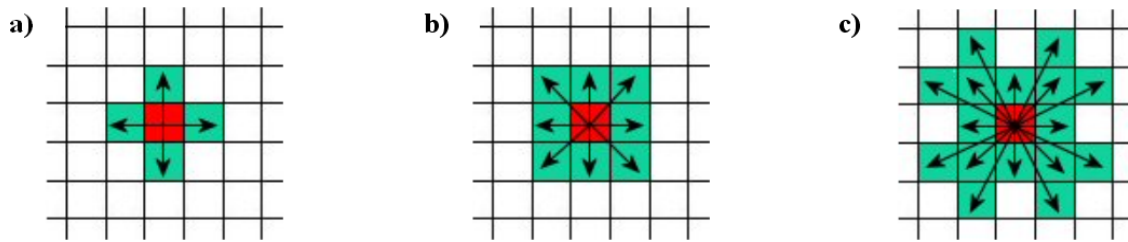


Figure 1: Different adjacency definitions for route finding: (a) 4-adjacency, (b) 8-adjacency, (c) 16-adjacency.

2 Problem Space

2.1 Terrain Setup

The sampled digital terrain image is an approximation to the real terrain. The higher the resolution of the digital terrain image, the more realistic the representation of the real terrain and the more accurate the path finding will be. However, there exists an upper limit on the resolution after which the road will not be any more accurate and will unnecessarily increase the running time of the path finding algorithm. In order to visualize the results we represent the terrain as a grey scale image with black and white representing the lowest and highest altitudes, respectively.

2.2 Adjacencies

To find a path from a starting node to a goal node, we must define a way in which successor nodes can be selected. While a real terrain allows stepping in any direction for a digital terrain 4-adjacency and 8-adjacency (see figure 1 (a) and (b), respectively) are the most common approaches.

Our application additionally implements 16-adjacency (see Figure 1 (c)) which allows even greater freedom of movement as we can now move north-north-east, north-east-east, etc. Using 16-adjacency, we can also achieve a smoother looking road by reducing the sharpness of the turns. To reduce sharp turns even further we can penalize sharp turns, as described in section 3.3.2. If we use the distance between nodes as our cost function for the ‘standard costs’ then

neighbouring nodes that can be reached using 4-adjacency cost 1, the ones that can be reached by the diagonal steps of 8-adjacency have a cost of $\sqrt{2}$ and nodes that can only be reached by the additional steps of 16-adjacency have a cost of $\sqrt{5}$. There are of course other cost functions that can be chosen, rather than just using the distance between two nodes. Section 3.3 introduces additional parameters that will influence the cost function.

3 The A* Algorithm

The A* algorithm finds the shortest path from a start node to a goal node by using a heuristic which estimates the cost to reach the goal node from the current node. The heuristic estimate for node n is usually referred to as $h(n)$. The A* algorithm also keeps track of the cost $g(n)$ needed to get to the current node from the start node. The total cost of a node is hence $f(n) = h(n) + g(n)$.

3.1 Heuristics

The heuristic function $h(n)$ guides the search towards the goal node. If the heuristic function is admissible (meaning it never overestimates the minimum cost to the goal), then A* is guaranteed to find the cheapest path. It is preferable to use a heuristic that underestimates the minimum cost as little as possible, as this will result in fewer nodes to be examined. An ideal heuristic will always return the actual minimum cost possible to reach the goal.

The Manhattan distance heuristic is illustrated in figure 2 (a) and is defined as $h(n) = |x_a - x_b| + |y_a - y_b|$

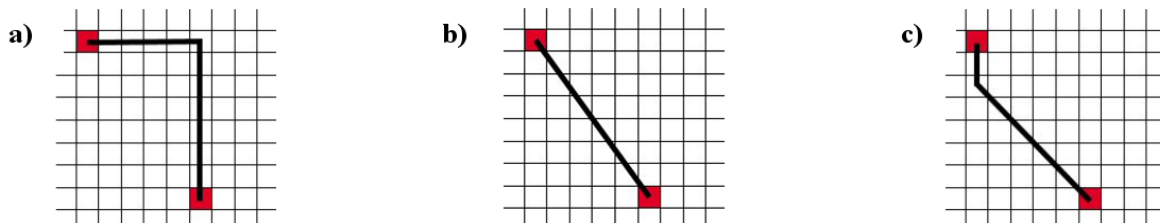


Figure 2: The Manhattan distance (a), Euclidian distance (b) and Diagonal Path Distance (c).

where (x_a, y_a) and (x_b, y_b) are the coordinates of the current start and goal node, respectively. The heuristic is ideal when using 4-adjacency.

The Euclidean heuristic (see figure 2 (b)) is defined as $h(n) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$ and 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 road.

The diagonal distance heuristic is defined as $h(n) = |x_a - x_b| + |y_a - y_b| + (\sqrt{2} - 2) \min(|x_a - x_b|, |y_a - y_b|)$ [5] and is illustrated in figure 2 (c). The 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. We found that using the Manhattan distance heuristic is about 40% faster than the other two heuristics with only a small increase in the cost of the solution.

3.2 User Defined Constraints

We introduce gradient penalties since realistic roads generally have a maximum gradient resulting from safety considerations and vehicle limitations. The Cornwall County Council, for instance, limits the maximum gradient for traditional surfaced roads to 10% [7]. The gradient between 2 nodes is calculated as $\theta = \tan^{-1}(\Delta h / \Delta s)$ where Δh and Δs are the height and ground distance, respectively, between the two nodes. The gradient penalties are cost multipliers with a user defined cost factor. We also introduce direction change penalties in order to keep the resulting roads straight since unnecessary turns would obstruct the traffic flow. The City of Hamilton in Montana, USA, for example has a regulation that says that a curve must have a minimum radius of 249 feet in order to provide a design speed of 30mph [8]. The cost factors can be again specified by the user in order to allow for the creation of different types of road (motorway, country road etc.).

3.3 Variations of the A* Algorithm

The A* will always select the cheapest node from all nodes not previously visited. For large terrains this process becomes inefficient and several modifications have been suggested which minimise the search space and improve the memory and time requirements of the algorithm.

The *Beam Search* limits the size of the list of unvisited nodes ("beam width"). Once the limit has been reached the node with the highest cost is dropped from the list if unvisited nodes to make room

for a new node. The major shortcoming of Beam Search is that it is not optimal and not complete, meaning it may not necessarily find the shortest path, even if the heuristic is admissible.

*Iterative Deepening A** [9] performs a series of searches, where each search has a maximum cut-off value, i.e. a limit on the $f(n)$ value. The cut-off value is increased with each iteration. The iterations continue until the goal node has been reached or the search space has been exhausted. Initially the cut-off value should be the $h(n)$ value of the start node. A difficulty that arises, is finding an appropriate step size for increasing the cut-off value with each iteration.

Searching in one direction (unidirectional search) involves searching a single search tree. An alternative is to perform a *bidirectional search* that searches two smaller trees instead. One search starts from the start node searching forwards to the goal node, the other one is searching backwards from the goal node to the start node. Because the search trees grow exponentially, the search space created by two small search trees is generally less than the size of a single large tree assuming that the bidirectional search meets in the middle. However, Pohl [10] notes that if there is more than one path from the start node to the goal node, then the two search fronts seldom meet in the middle. A variation of the bidirectional search is the *retargeting* approach, first suggested by Pohl and Politowsky [11]. The retargeting bidirectional search does not perform the forward and backward searches 'simultaneously', but switches between them.

4 Multi-Resolution A* Search

As an alternative approach to reduce the memory and time requirements of route finding we reduce the size of the search space by computing a multi-resolution representation of the terrain. We first find a route on the lowest resolution version of the terrain (which has the least nodes and hence allows a fast search) and then use each pair of nodes of the resulting solution as a start and goal node for a search on the next higher resolution representation of the terrain. This process is repeated until we obtain a route for the original terrain. In the following discussion we use the term sub-sampling factor to refer to the reduction of size of a low-resolution terrain when compared to its full resolution version.

We compute low resolution representations of the digital terrain by using a mean and 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. For example if we had a 5 by 5 group of cells where the left half (first three columns) had height values of 10 and the other half (last two columns) had height values of 140, then the single cell that will represent this group of cells will have a value of 62. As a result

<i>Sub-sampling factor</i>	<i>Time taken</i>	<i>Number of nodes visited</i>	<i>Total Cost of Path</i>
No sub-sampling	2 min 32 s	70788	297.10
3	992 ms	12581	423.76
5	281 ms	8687	369.98
10	266 ms	8869	315.41
20	6 s	27122	362.50

Table 1: Typical results for multi-resolution route planning with different sampling factors.

steep gradients in the original terrain become ‘smudged’ which makes taking that path more appealing due to the lower gradient penalty. Once the path on the sub-sampled terrain map has been found, and we perform the searches between the road points, we are now forced to traverse the steep gradient on the original terrain.

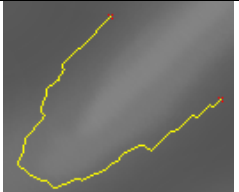
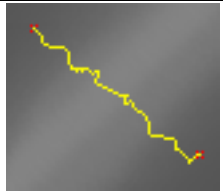
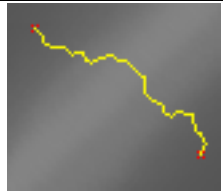
The median operator picks the middle value (median) of all the height values in the current group of cells; this requires the height values to be sorted. Using the example mentioned above, the median operator would be 10, i.e. the median operator preserves the steep gradient in this hypothetical data set. More advanced sub-sampling operators could be used to improve the outcome.

5 Results

5.1 Multi-Resolution Route Planning

We have compared different parameter settings for multi-resolution route planning using a variety of different terrains. A typical result is shown in table 1.

In general we found that as the sub-sampling factor increases, the smoothness of the road decreases. The roads found using sub-sampling factors of 5 and 10 produced reasonable roads, while the quality of the roads found using sub-sampling factors of 3 and 20 produced less satisfying roads. Also interesting are the differences in the running times of the various sub-sampling factors. Using sub-sampling factors of 5 and 10 proved to be the quickest, whereas using the sub-sampling 3 took nearly a second and the sub-sampling factor 20 took longest about 6s. The reason for the factors 5 and 10 being the quickest is that those sub-sampling factors represent equilibrium of large scale and small scale searches. When sub-sampling with a very small factor (e.g. 3) the search between the way points is very fast but we still have to search a reasonable large low-resolution terrain. When using a high sub-sampling factor (e.g. 20) the low-resolution terrain is much smaller but the way points are further apart which results in a slower secondary search. Note that the point of equilibrium depends on the size of the terrain, i.e. for a very large

<i>Sub-sampling Type</i>	No sub-sampling	Single sub-sampling	Single sub-sampling
<i>Image</i>			
<i>Factor(s)</i>	None	5	10
<i>Time</i>	12 m 34 s	1 s	9 s
<i>Nodes visited</i>	146,772	20,882	36,534
<i>Total Cost</i>	534.71	614.07	580.65

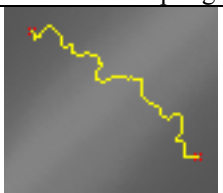
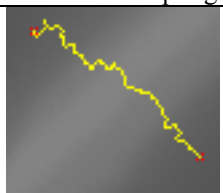
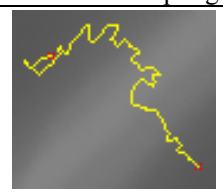
<i>Sub-sampling Type</i>	Multi sub-sampling	Multi sub-sampling	Multi sub-sampling
<i>Image</i>			
<i>Factor(s)</i>	5, 10	3, 5, 10	3, 5, 10, 15
<i>Time</i>	1 s	484 ms	437 ms
<i>Nodes visited</i>	21,029	18,695	21,546
<i>Total Cost</i>	641.63	717.28	1040.13

Table 2: Comparison of multi-resolution route planning with multiple levels of sub-sampling.

terrain a higher sub-sampling factor is necessary to minimise the running time. In summary we found that using large sub-sampling factors reduces the size of the search space but it has several disadvantages: 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 route 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 road, thus increasing the distance between them on the original terrain map. This means that we are running the path finding algorithm less often on the original terrain map. However, it is usually less expensive to run the path finding algorithm more often but with a shorter distance between the each start and goal node.

We also examined the performance of the algorithm when using multiple levels of sub-sampling. In general we found that this decreases the running time but it also reduces the quality of the solution found. For small terrains two levels of sub-sampling seem to be sufficient whereas for very large terrains more levels can be necessary. Also we found that sub-sampling factors which are close together (e.g. 3, 5, 10, 15) should be avoided. A typical example for a small terrain is shown in table 2. Note that in all cases

the resulting solutions must be smoothed before using them as a plausible route.

5.2 Comparison of Different Algorithms

We compared different variations of the A* algorithms. A typical (unsmoothed) result of is shown in table 3.

It can be seen that both the A* algorithm and the Iterative Deepening A* algorithm produce an optimal route (minimal cost). The only efficient variation of the A* algorithm was in our experiments the retargeting search which produced however poor roads having the largest total cost. Using multi-resolution terrains we obtain the best performance while still getting solutions of an acceptable quality. We also found that the results obtained with an A* search with sub-sampling are similar to those obtained using a Beam Search with sub-sampling (see the above example). This is due to the fact that we usually reach our current goal before the width of the beam is exceeded.

6 Conclusion

Finding a path between two points on a digital terrain map can take a considerable amount of time and memory using the standard A* algorithm. We have defined additional constraints, so-called gradient penalties and direction change penalties, in order to

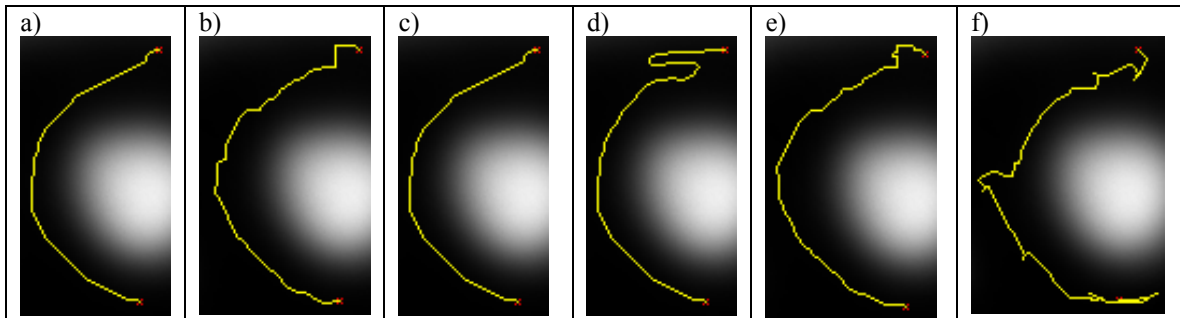


Image	Algorithm (& sampling factor if applicable)	Time taken	Number of nodes visited	Total Cost of Path
a)	A* (with no sub-sampling)	14 s	21,720	222.47
b)	A* (with sub-sampling factor 5)	187 ms	7,431	428.99
c)	Iterative Deepening A* (no sub-sampling, 10% $f(n)$ increase with each iteration)	24 s	51,728	222.47
d)	Beam Search (no sub-sampling, beam width = 300, beam cutoff = 100)	23 s	27,190	294.66
e)	Beam Search (sub-sampling factor 5, beam width = 300, beam cutoff = 100)	203 ms	7,983	447.05
f)	Retargeting Search (no sub-sampling, each search front visits 100 nodes at a time)	218 ms	13,934	470.36

Table 3: Comparison of results obtained with different variations of the A* algorithm.

increase the realism of the solutions. Additional constraints can be added in order to allow for the creation of bridges and tunnels. We have analysed several variations of the A* algorithm in order to address the time and memory issues, but we found that the only variation that significantly reduced the running time of the road finding was the retargeting approach which produces low quality solutions. As an alternative we presented multi-resolution route finding methods which result in a large reduction of the search space and hence yield a dramatic improvement of running times and memory requirements. The road finding takes place on the sub-sampled terrain and after the road is found, all the points along the road are transformed back to the corresponding locations on the original terrain. We then find the road between each consecutive pair of the road points. We found that the optimal sub-sampling factor depends on the size of the terrain and represents equilibrium of high-resolution local searches and low-resolution global searches. However, the quality of a solution usually decreases with an increasing sub-sampling factor. The choice of the sub-sampling filter is important and we found that a median filter works best since it maintains gullies and ridges. We have also introduced multi-resolution sub-sampling which uses intermediate resolutions between the high and low resolutions. The motivation behind multi-resolution sub-sampling is that we can increase the performance when using large sub-sampling factors so that the distances between the way points is reduced by using an intermediate resolution before going back to the high resolution terrain. The performance gain from using multi-resolution sub-sampling over single sub-sampling is usually not very significant but the loss in the quality of the road is significant. Thus the use of multi-resolution sub-sampling is only appropriate on large high resolution terrains.

7 References

- [1] F. M. Jönsson, An optimal pathfinder for vehicles in real-world digital terrain maps, (1997) URL: <http://www.student.nada.kth.se/~f93-maj/pathfinder/>.
- [2] E. W. Dijkstra, A note on two problems in connection with graphs, *Numerische Mathematik*, vol. 1, (1959) 269-271.
- [3] S. Russell and P. Norvig, Best First Search, *Artificial Intelligence – A modern approach*, Prentice Hall, (1995) 94-97.
- [4] P. Hart, N. Nilson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, July, (1968) 100-107.
- [5] D. R. Wichmann, "Automated Route Finding on Digital Terrains", COMPSCI 780 Project Report, Graphics Group, Dept. of Computer Science, University of Auckland, New Zealand, February 2004.
- [6] "Amit's Thoughts on Path-Finding and A-Star", Introduction, URL: <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#S2>.
- [7] Cornwall County Council, "Vertical Design", URL: <http://www.cornwall.gov.uk/environment/design/section5/des53.htm>, February 2004.
- [8] City of Hamilton, Section 16.32.090 Streets and roads, URL: http://www.cityofhamilton.net/codes/Title_16/32/090.html, 1999.
- [9] R. E. Korf, "Iterative-Deepening A*: An Optimal Admissible Tree Search". Proceedings of the International Joint Conference on Artificial Intelligence, Los Altos, California, Morgan Kaufmann, (1985) 1034-1036.
- [10] I. Pohl, "Bi-directional search", *Machine Intelligence*, vol. 6 (1971) 127-140.
- [11] G. Politowski and I. Pohl, "D-node retargeting in bidirectional heuristic search", *Proc. AAAI-84*, Austin, Texas, (1984) 274-277.