An Analysis of Dimdal's[1] "An Optimal Pathfinder for Vehicles in Real-World Terrain Maps"

Submitted in partial fulfillment of the requirements for the
Qualifying Exam for Ph.D. students
in Computer Science at the University of Iowa
by D. Ezra Sidran.

Spring 2005

[1]F. Markus Jönsson has changed his last name to Dimdal and has requested that he be referenced as such in the title of this paper. Dimdal wrote to me: [I]… "changed to a new family name 5 months just before my first child was born - I wanted us to have a name that is shared by no one else here in Sweden, my old family name Jönsson is *very* common here)." Dimdal means "Misty valley" in Swedish. However, to avoid confusion, I am leaving the citation as Jönsson.

0. Introduction

"When game developers look at AI research, they find little work on the problems that interest them, such as nontrivial pathfinding…"
- John Laird (University of Michigan) [1]

"The most buggy aspect of the game [is] enemy pathfinding…"
- Review of *Star Wars: Knights of the Old Republic*, PC [2]

"Various algorithms have been proposed for the determination of the optimum paths in line networks. Moving in space is a far more complex problem, where research has been scarce."
- Stefanakis & Kavouras [3]

For the purpose of this paper 'pathfinding' is defined as the various methods and techniques used to calculate an optimal path from one point to another across a real-world or realistic three dimensional terrain. The need for efficient pathfinding algorithms is well documented – the above quotes are only a minor sampling - and the quest for a "black box" universal pathfinding algorithm is ongoing.

There is also an extraordinary lack of published academic research on the subject. A search of the ACM Digital Library returned only two papers on the topic. [4] [5]

In the almost fifty years since research on pathfinding has begun, numerous methods have been employed to determine the shortest or fastest path. While this paper is only concerned with the weighted graph method proposed by Dijkstra [6] in 1959 and its direct descendents (including the A* algorithm) [7] various other interesting techniques have been suggested including Line Intersection, Lonningdal's "crash-and-turn" algorithm and Burgess' "Gas diffusion as a method of analyzing avenues of approach through digital terrain."[8] However, analysis by Jönsson [9], and many others, concludes that the weighted graph method is the most efficient method; especially considering that digital terrain and elevation maps easily lend themselves to this method of discretization.

1. The Problem Defined

Jönsson's optimal pathing algorithm is designed to work with multiple terrain types and three-dimensional elevation data stored in an M x N matrix. While it is possible to modify this algorithm to work with hexagonal planar meshes, this paper will assume that the terrain and elevation has been discretized into a regular quadratic planar mesh (a grid).

Figure 1. Section of sample map showing elevation contours (from author's "AI Test Bed" program).

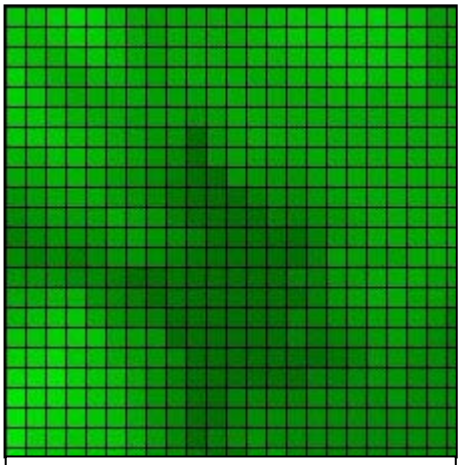


Figure 2. . Section of same map showing elevation, now digitized and displayed in color. (from author's "AI Test Bed" program).

The purpose of the pathfinding algorithm is to determine, first, if it is possible to find a path from the start point to the goal (maps may contain impassable barriers such as rivers, swamps or elevations that have been defined as too steep to cross), and, if so, to determine the optimal path. There are many factors that are evaluated to determine the optimal path (defined as the path with the *least weight*) including the speed in which a path will be traversed and if the path can be observed by enemy units. These factors are discussed below.

In addition to the correctness of the pathfinding algorithm, the speed at which it executes is of paramount importance. Indeed, since one of the most common applications of pathing algorithms is in the commercial computer game industry – and because only a small fraction of the computer's clock cycles can be budgeted to pathing – an algorithm that

quickly returns a valid path is valued more highly than an algorithm that returns an optimal path but exceeds its allotment of clock cycles.

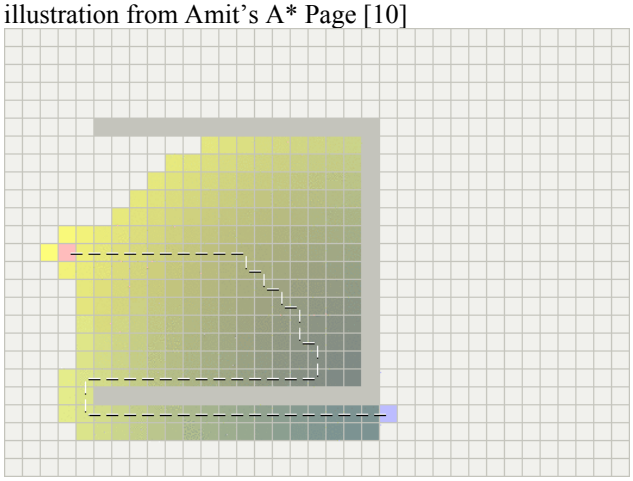
Lastly, we desire a most general pathfinding algorithm that works optimally with few assumptions (or no assumptions) about the terrain, elevation,

Table 1: Effects of a priori information on pathfinding optimization.

A priori information:	Optimization:	Note:
No impassable terrain	Use Best-First-Search (BFS).[2]	BFS is very fast but handles obstacles badly and is *not* guaranteed to find the shortest path if detours need to be made.
Roads	Utilization of road net. Concentrate pathfinding on reaching nearest road, following road to nearest exit point to goal.	Traveling along a road net will usually result in the fastest path.
Map previewed; we are not dealing with a previously "unknown" map. Location of bridges or bottlenecks.	We can use preprocessing methods to determine waypoints, choke-points, bridges, etc.	This is a common optimization done in commercial games.

roads and enemy units. In Table 1 (above) we see that if we possess some a priori information major optimizations can be realized. The commercial game industry often employs the strategy of pre-calculating waypoints, bottlenecks and bridge-crossings to optimize their pathfinding routines. However, we are concerned with pathfinding algorithms that function optimally even without any a priori information.

[2] Best-First-Searches run fast but are extremely vulnerable to 'traps' of impassible terrain as shown in this illustration from Amit's A* Page [10]



2. Summary of What Jönsson's Optimal Pathfinding Does

F. Markus Jönsson's master's thesis, "An Optimal Pathfinder for Vehicles in Real-World Digital Terrain Maps" [9] introduces an algorithm that given a map (represented by terrain and elevation matrices and roads as an array of vectors) and locations of enemy units will plot the fastest, least observable, route from a starting position to a goal position.

Jönsson's has developed a sophisticated h(n) heuristic estimate cost function that accounts for more variables than the traditional A* cost function (described below) and includes modifications for terrain, elevation, roads and enemy line of sight.

Algorithm Overview:

Input:

- An *N* x *M* matrix representing the elevation of a map in a digital format (usually meters above sea level).
- An *N* x *M* matrix representing the terrain of a map in a digital format (usually in sixteen values; see *Figure 3*, below) with an associated table of 'costs' for each terrain.
- An array of polygon chains representing roads vectors.
- An array of units and locations (in XY coordinate pairs) that represent enemy units on the map.
- A *starting* location (XY coordinate pair).
- A *goal* location (XY coordinate pair).

Output:

- A path (an array of contiguous XY coordinate pairs) from the starting location to the goal location. This path will be the path with the lowest weight as determined by slope, terrain and enemy visibility.
- The algorithm returns '0' if a path from the starting location to the goal location cannot be found.

3. A Brief Overview of Related Work

E.W. Dijkstra's 1959 paper *A Note on Two Problems in Connexion with Graphs* [6] inevitably appears as a reference in pathfinding papers. Of the two problems presented by Dijkstra, it is the second one, "Find the path of minimum total length between two given nodes *P* and *Q*," that interests us, and that is the basis of all weighted graph pathfinding algorithms.

The good news about Dijkstra's Algorithm is that it is thorough; if there is a path from the start to the goal it will return the path with the smallest values. The bad news is that it is a naive algorithm and runs in O((V + E) *log* V) which is O(E *log* V) if all vertices are reachable from the source. [11]

The A* algorithm was first presented in 1968 in a paper entitled, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," Hart, Nilsson and Raphael. [7].

Like Dijkstra, Hart, Nilsson and Raphael, thought of their algorithms directed towards sparse weighted graph problems, "of the sort of problem… of cities with roads connecting certain pairs of them." [7]. It would be a number of years later before this algorithm would be applied to real-world maps stored as terrain and elevation matrices.

The A* algorithm is a direct descendent of Dijkstra's Algorithm. Like Dijkstra's, A* also maintains two lists; traditionally called "open" which is usually implemented as a priority queue and "closed" (the list of nodes that have been examined).

However, A* introduces an important new concept, the cost function, and the equation associated with it: "the cost $f(n)$ of an optimal path through node n is estimated by an appropriate evaluation function $f(n)$. We can write $f(n)$ as the sum of two parts: $f(n) = g(n) + h(n)$ where $g(n)$ is the actual cost of an optimal path from s to n, and $h(n)$ is the actual cost of an optimal path from n to a preferred goal node of n.[7] This cost function is described in much greater detail in section 5, below.

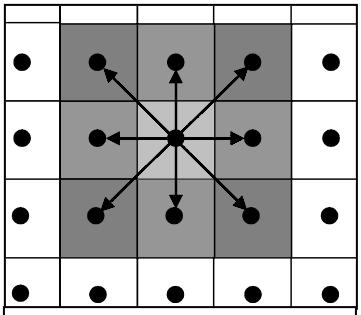
Stefanakis and Kavouras, in their 1995 paper, "On the Determination of the Optimum Path in Space" [3] present a "new approach to the optimum path finding problem which they summarize in five steps:

1. Determination of a finite number of spots in space.
2. Establishment of a network connecting these spots.
3. Formation of the travel cost model.
4. Assignment of accumulated travel cost values to these spots from the point of reference (i.e. the departure or destination spot).
5. Determination of the optimum path(s).

As we shall see, Jönsson will use this approach as the basis for his optimal path algorithm.

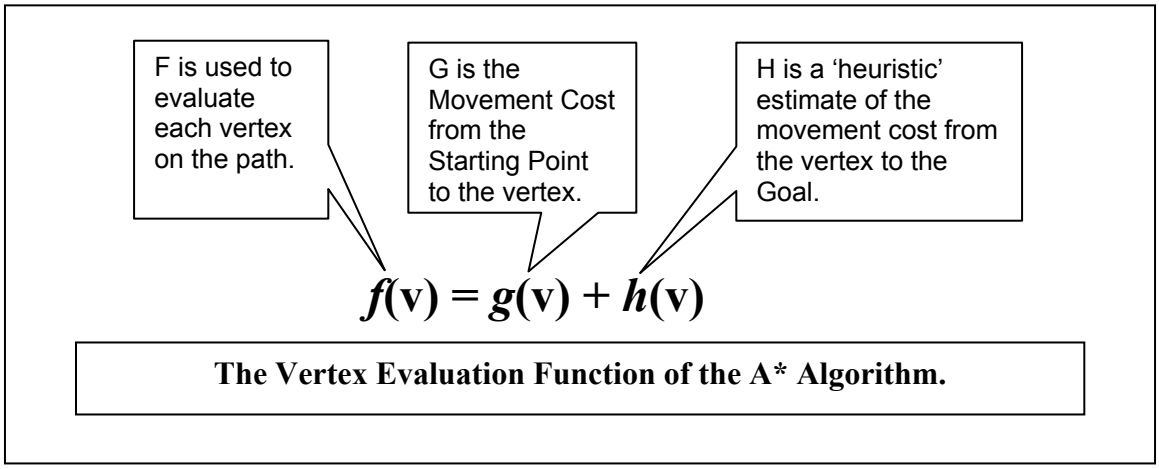
4. Problem Reduction

Jönsson's approach is to, "discretize the problem into essentially a weighted graph search problem." The first step is to restrict "(discretizing) the infinite number of locations of the continuous space into a finite number of points." [9] This is accomplished by converting (discretizing or digitizing, if you prefer) an 'analog' map into a digital "regular quadratic planar mesh" of the sort used by the U. S. Geological Survey (DEM format) or the military (DTED format). Jönsson then restricts movement to the edges between the vertices (the term 'cell' is used interchangeably with 'vertex') along the directed edges. The cells that are destinations of all the edges of a given cell are called its 'neighbors' (see illustration right). By this method Jönsson has converted a map into a directed graph.



A cell and its eight neighbors from Jönsson [9].

Dijkstra's Algorithm and the A* Algorithm were created to find the shortest path in weighted sparse directional graphs.



The Vertex Evaluation Function of the A* Algorithm.

By the 1980's these algorithms were employed in the computer game industry and elsewhere to find paths not in sparse graphs but dense graphs represented by N x M matrices that corresponded to real-world type terrain. Generally the edge weights of Dijkstra's and the A* algorithms

were assumed to simply be the distance between vertices[3]. Jönsson introduces refinements including calculations for terrain and elevation effects on vertex weight.

4.1 Developing a function that accounts for terrain.

Terrain penalties, or the 'cost' of 'entering a node' from an adjacent node have been used in board wargames since at least the eighteenth century [12].

A typical wargame terrain effects chart appears at right. [13] It is interesting to note that traditionally these board wargame terrain effect charts employ a percentage with "rolling flat terrain" = 100% of maximum movement and other terrain types being a percentage of this effect.

For example, in the chart at the right, if a unit travels at 10 miles per hour on "rolling flat terrain" it will travel at 50% or 5 miles per hour on "Rugged Mixed".

Because the A* algorithm selects the path with the lowest f we need to employ a terrain effect that *adds* a penalty. Consequently, we may set the cost of entering a "rolling flat terrain" node at '1' and the cost of entering a "Rugged Mixed" node at 2.

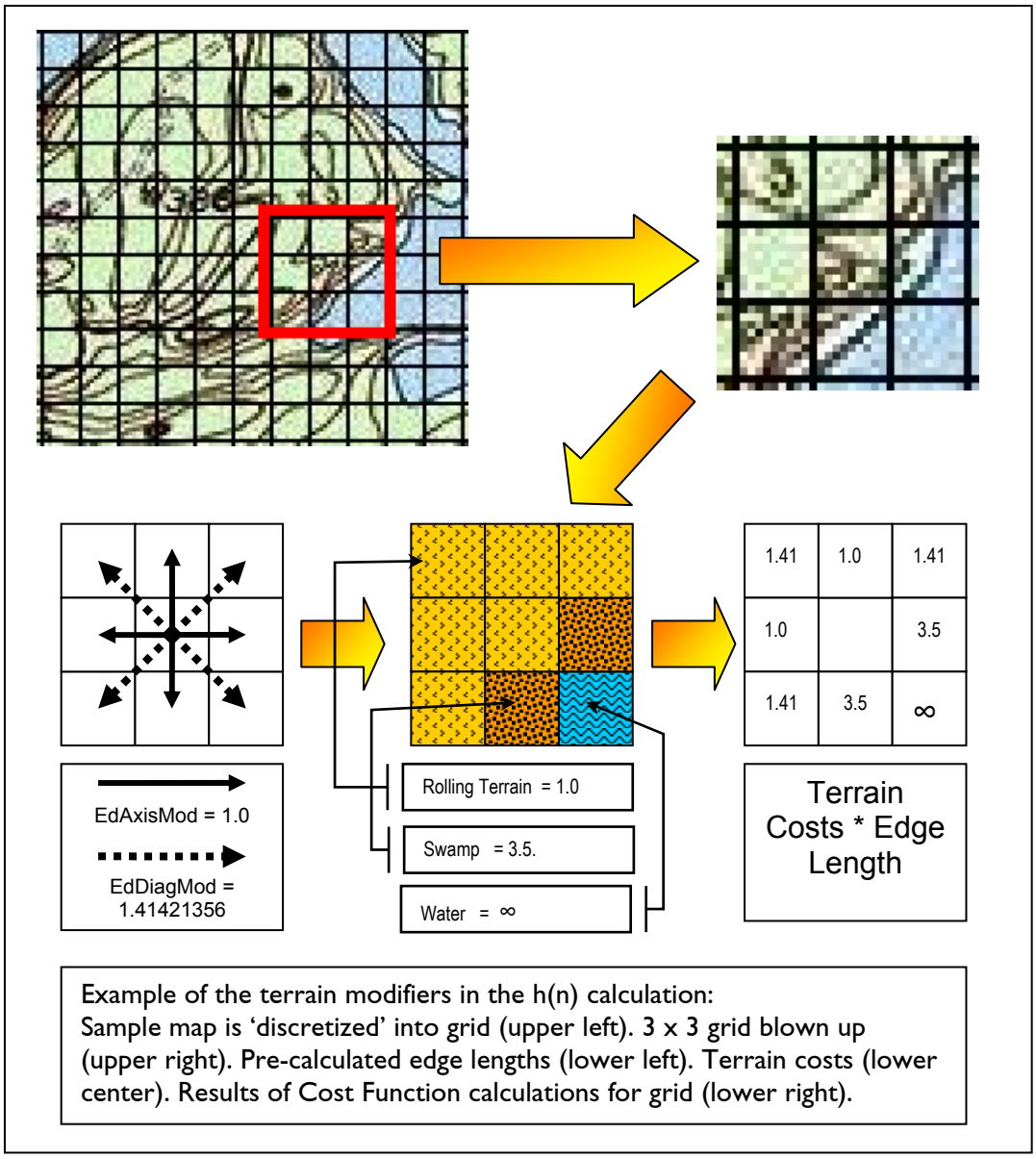
Another interesting observation is that the number of terrain types is frequently below 16. This allows for the terrain value for a node to be economically stored.

TABLE 1.

Terrain Characteristics	Mobility r_m
1. Rugged—Heavily Wooded	0.4
2. Rugged—Mixed (or extra rugged—bare)	0.5
3. Rugged—Bare	0.6
4. Rolling—Heavily Wooded	0.6
5. Rolling—Mixed	0.8
6. Rolling—Bare	1.0
7. Flat—Heavily Wooded	0.7
8. Flat—Mixed	0.9
9. Flat—Bare, hard	1.05
10. Flat Desert	0.95
11. Rolling Dunes	0.3
12. Swamp—jungled	0.3
13. Swamp—mixed or open	0.4
14. Urban	0.7

Figure 3. A Terrain effects on mobility chart from Dupuy[13]. Note: this chart uses a percentage of theoretical 'maximum movement' rather than a penalty cost.

[3] Because our directed graph is a matrix there are, in fact, only two distances between nodes: either a diagonal or an axial edge. These values are pre-calculated and used in the Cost Function.



Example of the terrain modifiers in the h(n) calculation:
Sample map is 'discretized' into grid (upper left). 3 x 3 grid blown up (upper right). Pre-calculated edge lengths (lower left). Terrain costs (lower center). Results of Cost Function calculations for grid (lower right).

Slopes.

Another common variable used in real-world terrain calculations is the difference in elevation (slope) between two cells. Jönsson employs a simple calculation to determine if the slope of the vertex between cells is too great (i.e. greater than the value *dwMaxHeightDiff*) to be traversed and, if so, returns the constant *infinity* which removes the vertex from the set of vertices to be considered for an optimal path. This method is acceptable because Jönsson's algorithm was designed to work with vehicles. However, if Jönsson's algorithm was to be extended to calculate optimal paths for

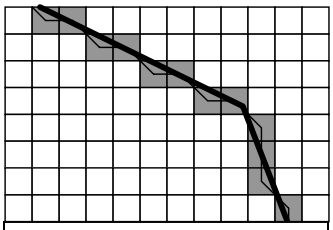
foot-soldiers the slope of the vertex should be calculated and added as a modifier (such as the terrain modifier) to the weight function.

Obstacles.

Jönsson defines an obstacle as an impassable area and simple sets the movement cost for such areas to *infinity*.

Roads.

Jönsson's Optimal Pathfinder accepts as input a representation of roads stored as an array of vector data. This is similar to a method employed by the author in his "UMS II: Nations at War" commercial computer game published in 1997. Jönsson 'reconstitutes' the road vectors using the Bresenham Algorithm and then 'rasterizes' the vector data (right). This data is then stored as a cell attribute and it is used in the calculations for the cost function.

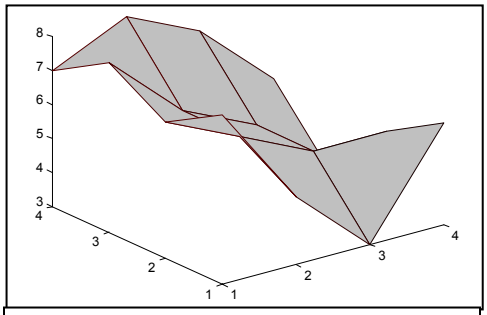


Road 'rasterization' from Jönsson [9].

Enemy line-of-sight.

An important application of optimal pathfinding routines is in military simulations. Consequently, it is very desirable to include the implementation of a three dimensional line-of-sight algorithm that determines what areas are visible to enemy units and apply this information to the optimal pathfinder routine.

Jönsson does not employ a 3D Bresenham algorithm because of the potential of "overt aliasing" and the potential for inaccuracy that this could introduce. Consequently, Jönsson uses a fixed point Digital Difference Algorithm (DDA) which is more accurate because it uses floating point numbers and sub-pixel steps. However, as Jönsson points out, "This calculation can be one of the heaviest computational burdens in… [the optimal pathfinding]… algorithm. Therefore, we don't want to do this if it isn't absolutely necessary." Jönsson leaves a byte in each cell structure to indicate if the visibility for the cell has been previously calculated and, if so, if the cell



Example of bilinear interpolation surface on a 4x4 vertex grid from Jönsson [4].

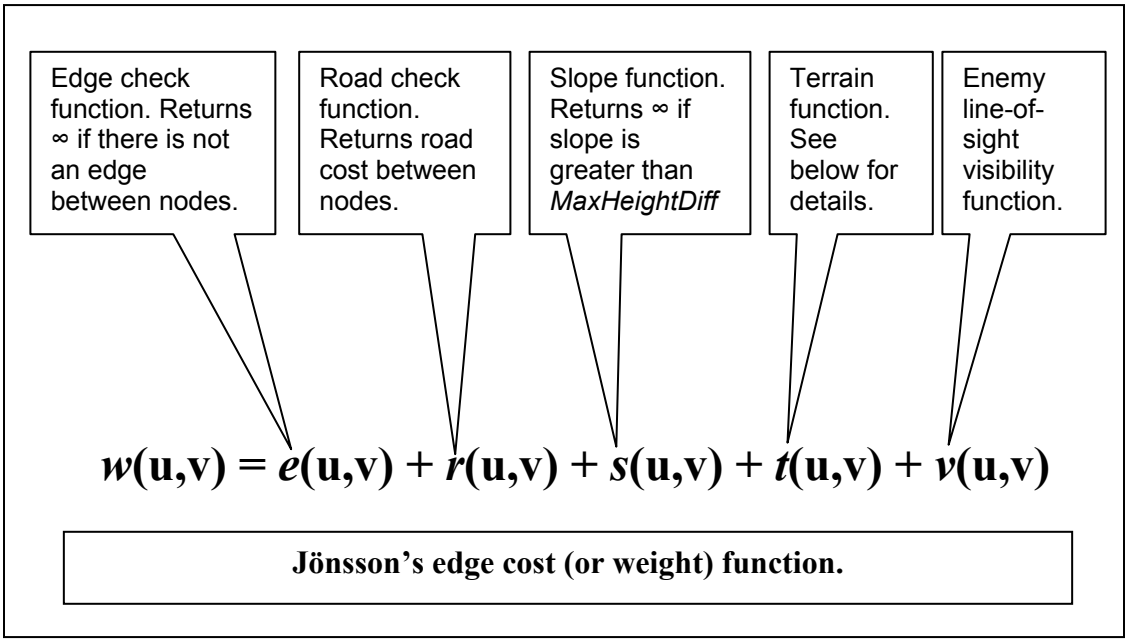
is currently visible from known enemy positions.

4.2 Putting it all together: computing edge cost.

The cost for traversing each edge between two cells is not stored but rather calculated once as needed. Jönsson defines *infinity* as half the maximum value of the data type "so that it can safely be added to any cost value without any risk of overflow." [9] *Infinity* is used to mark inaccessible cells and the edges of the map.

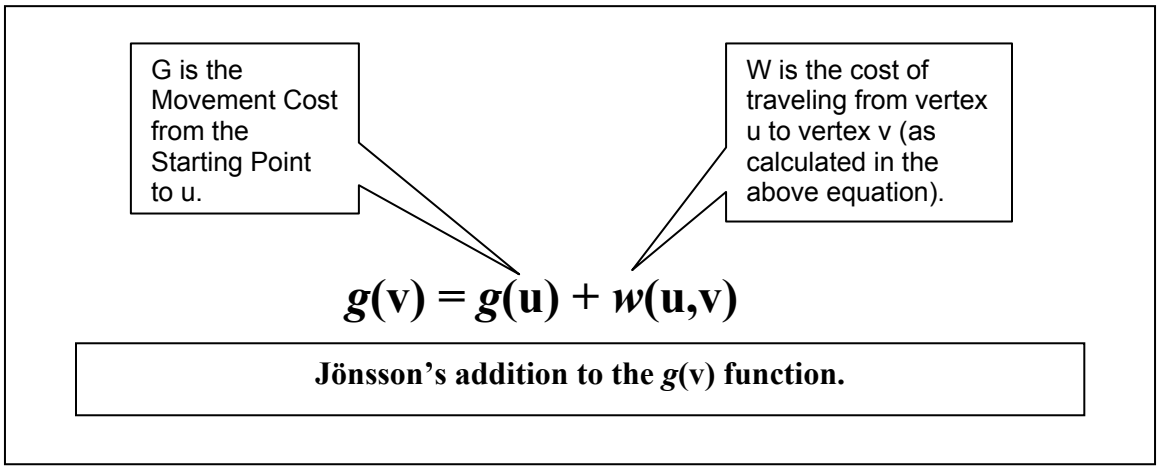
The algorithm for computing the cost function between two adjacent cells is:

1. Check to see if the edge does not return *infinity*. If the edge returns *infinity* this edge cannot be traversed, return *infinity*, else go to step 2.
2. Check to see if both the source and destination cells contain identical *road values*. There are four possible *road values*: no road, light road, medium road, and large road. If either cell contains the 'no road' value go to step 3. If both cells contain the same road value it is assumed that the two cells are connected by the same road; return the cost for that road value, else go to step 3.
3. Calculate the slope of the edge between the two cells. If the slope is greater than a predefined maximum slope value return *infinity*. (Note: Using the slope as a modifier for speed and/or cost might make an interesting addition to this algorithm.) Else go to step 4.
4. Calculate the basic terrain cost (see illustration below).
5. Calculate if the destination cell is visible to known enemy locations.
6. Return cost.



The terrain costs are implemented via a pre-calculated look-up table. This also adds the correct weight for distance (remember there are two different values for distance depending on the direction; see illustration next page).

Looking back at the vertex evaluation function, $f(v) = g(v) + h(v)$ on page 8, the edge cost comes in to play in computing g(v) as follows:



The following section details the h(v) part of the f(v) function and how the A* search proceeds.

5. Searching for the Optimal Path

The key to the A* algorithm is the Heuristic Function: h(v).

Jönsson writes, "Generally the 'distance from u to v' times 'smallest terrain cost' is the best estimate we can do for h[v] when an underestimate must be guaranteed. What is the optimal 'distance' estimate depends on the graph structure. The true Euclidean distance should serve, but it would often be lower than the actual edge lengths along the shortest path… Instead, since the graph structure is uniform we can actually calculate the exact, 'optimal' distance. If we'd only had orthogonal edges, it would be the so-called 'Manhattan' distance: $|\Delta x| + |\Delta y|$. In our case, with diagonal edges as well, it is easily seen to be 'diagonal edge length' * min ($|\Delta x|$, $|\Delta y|$) + 'axial edge length' * $|\Delta x| - |\Delta y|$. An addition[al] bonus compared to using the Euclidean distance is that no costly square root calculation is required."[9]

$$\mathbf{h(v)= ((}\text{Diagonal Edge Length} * \min(dx , dy)) + (\text{Axial Edge Length} * |dx - dy|)) * \text{Minimum Terrain Cost}$$

where dx = |SourceX – DestinationX| *and* dy = |SourceY – DestinationY|

Jönsson's implementation of the A* heuristic.

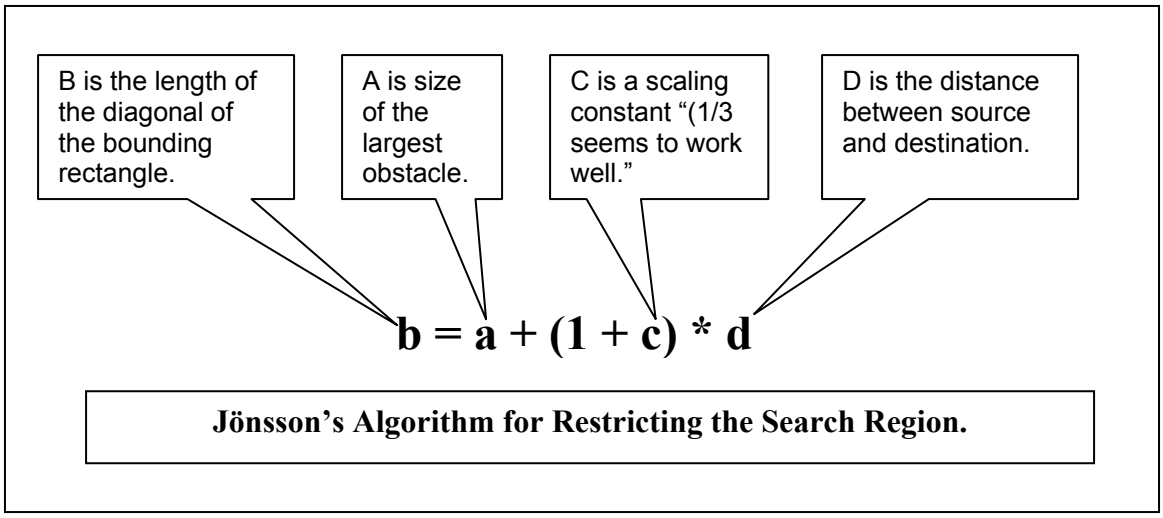
Jönsson uses the Minimum Terrain Cost (i.e. the smallest of all possible terrain costs) as the multiplier because it is essential that h(v) return an underestimate to guarantee that a path will be found. Other ideas, such as an average terrain cost, cannot guarantee that h(v) will return an underestimate.

5.1 Other issues.

Jönsson does not search the entire graph for an optimal path but, rather, restricts the search area to a smaller region within the entire graph.

Jönsson writes, "How can the minimum size of this region be determined without a priori knowledge of the actual optimal path? …Thus, it is necessary to resort to using some kind of heuristic to determine a good region size." The heuristic that Jönsson uses employs an estimate of the largest obstacle (i.e. impassable terrain) and the distance between the source and the destination.

"Practical tests have shown that at small scales… [the largest obstacle estimate]… is much more important while at larger scales… [the distance between the source and the destination]… is dominant." Consequently, Jönsson arrives at the following "compromise heuristic":



Jönsson's Algorithm for Restricting the Search Region.

Lastly, there is the issue of 'ties'[4] – paths that share the lowest 'f' score – in the A* algorithm. In Nilsson's algorithm "Ties among minimal f values are resolved in favor of the deepest node in the search tree." Obviously, this requires each of the paths to be searched to the very end to determine which path possesses the deepest node; and this takes more time. Consequently, A* is frequently implemented with various 'tie-breaking' heuristics.

These tie-breaking heuristics include:

[4] Jönsson does not describe the heuristic he uses for tie-breaking. I sent him an email asking what method of 'tie-breaking he used and he replied, "To be honest, I don't remember any more..." It may be surmised that Jönsson does not employ the "give preference to paths that are along straight lines" method (above) because he later employs a "path smoothing" algorithm which, presumably, would not be necessary if he implemented this method.

- **Chose the node that is closest to the goal.** This can be accomplished by adding a small value (perhaps as small as 0.1% though, in theory, the incremental value should be based on the number of steps of the optimal path) to the f value for every step along the path. This will have the effect of making the path with the most steps having that much of a greater f value. Patel [10]
- **Give preference to paths that are along straight lines.** A different way to break ties is to prefer paths that are along the straight line from the starting point to the goal:

  This code computes the vector cross-product between the start to goal vector and the current point to goal vector. When these vectors don't line up, the cross product will be larger. The result is that this code will give some slight preference to a path that lies along the straight line path from the start to the goal. When there are no obstacles, A* not only explores less of the map, the path looks very nice as well. [10].

Efficient representation of a graph.

Jönsson employs some very impressive optimization techniques to minimize the memory requirements for storing the graph. Rather than using 4 bytes to explicitly store the edge cost plus two bytes to store an index to the destination cell (times 8 for the eight edges from a cell to its neighbors) Jönsson uses only 3 bytes to store the values for each cell (see footnote 21, above) and, instead uses a two dimensional index (*CellRef*) into the array of cells. Then, by numbering, "the edge directions clockwise from 0 to 7 (0 being North, 1 Northeast, 2 East, 3 Southeast, etc.)… it is easy to calculate the index change and store it in an look-up table, *crWalkEdgeDelta[8]*. To walk an edge, with direction *ed*, all we then have to do is 'add' *crWalkEdgeDelta[ed]* to the source cell's *CellRef*."[9]

The Priority Que (using a Fibonacci Heap).

Dijkstra's Algorithm and the A* algorithm both employ a list (frequently called 'Open') which is usually implemented as a priority queue. Jönsson has recognized that a Fibonacci Heap is considerably faster than the common

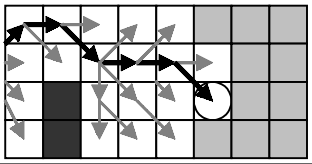
"binary heap" when employed for pathfinding across large real-world terrain (dense) graphs. [14] Binary heaps are faster for small heaps.

The key to the efficiency of a Fibonacci Heap when employed in dense graph pathfinding is that its 'costly' functions such as 'housekeeping' are amortized; therefore the bigger the heap the lower the cost of the occasional (but expensive time-wise) 'housekeeping' operations. [9].

N.B. Employing a Fibonacci Heap may not be practical when implementing Jönsson's Optimal Pathfinder in a commercial Real Time Strategy (RTS) computer game because the 'housekeeping' may cause the game to pause for an unacceptable length of time. Traditionally, RTS games allot a small fraction of the clock cycles for AI and pathfinding. Consequently, implementing a Fibonacci Heap should be reserved for games that use 'phases' or other non time-slice applications.

Reconstructing the Path.

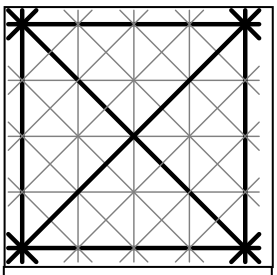
After the goal has been reached, "we can easily use it for backtracking the optimum path from the destination vertex. We start in the destination cell and then walk in the opposite direction to the edge direction stored in the current cell until we reach the source cell. It is then a simple thing to reverse the sequence of the cells visited during the backtracking to produce a 'forward' path from source to destination. The result of a typical search is illustrated (above). The circle is the path's destination. Gray cells never needed to be visited during the algorithm execution. Dark gray is an obstacle. The arrows show the edges of 'currently best known' paths to all examined cells, i.e. the ed[v] values for the vertices at the arrow heads. For the retired cells (not marked) this is also the optimum path to those cells. The thicker dark arrows show the least cost path to the destination."



The result of a typical search. Jönsson.[9]

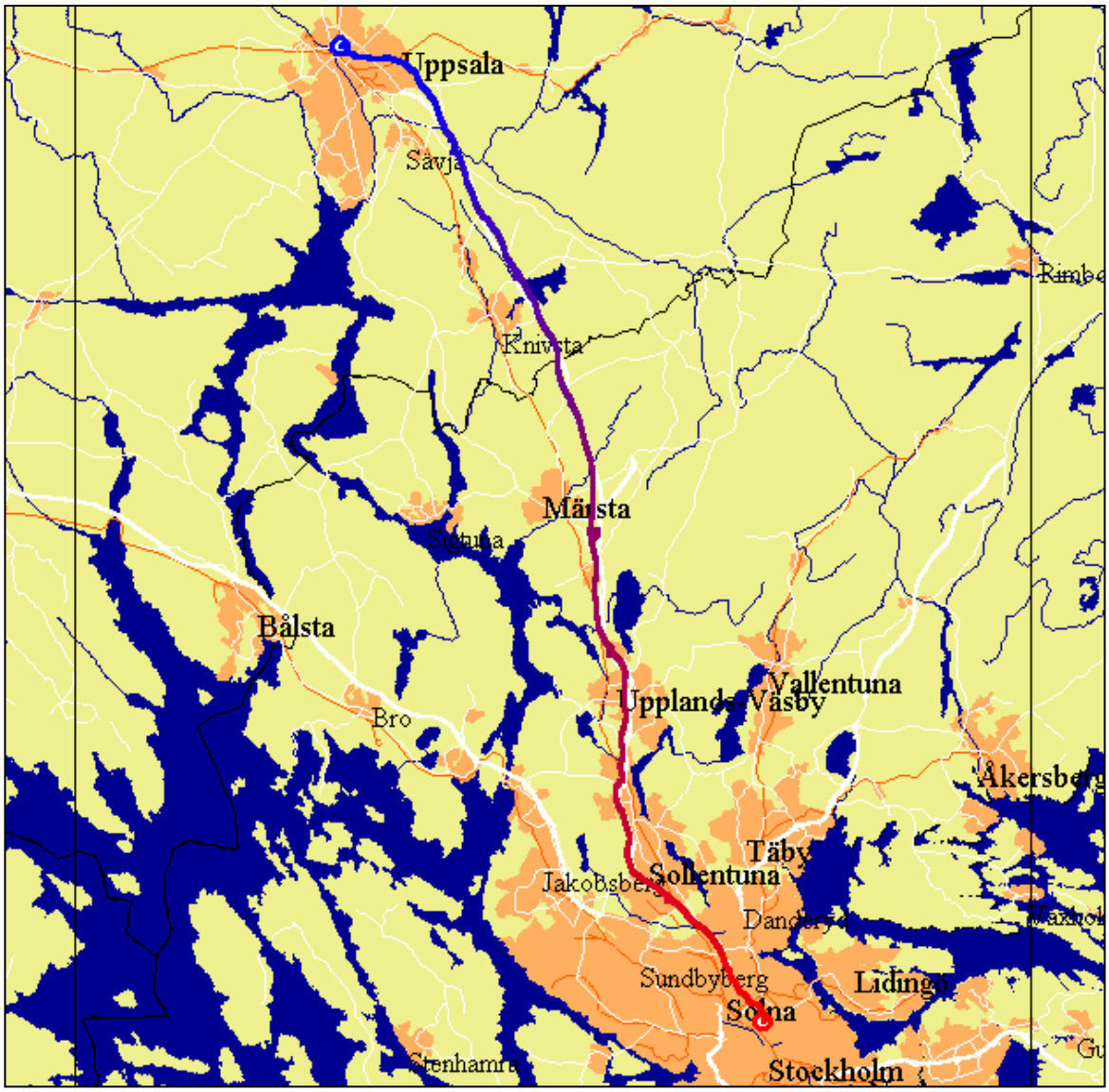
Progressive approximation.

Jönsson's final optimization is progressive approximation: the method of 'subsampling' a graph by 'clumping' 4x4 vertices into one. This method is employed when it is possible to exceed physical memory. There are, however, some issues that need to be resolved when employing subsampling:



16 vertices are subsampled into one from Jönsson [9].

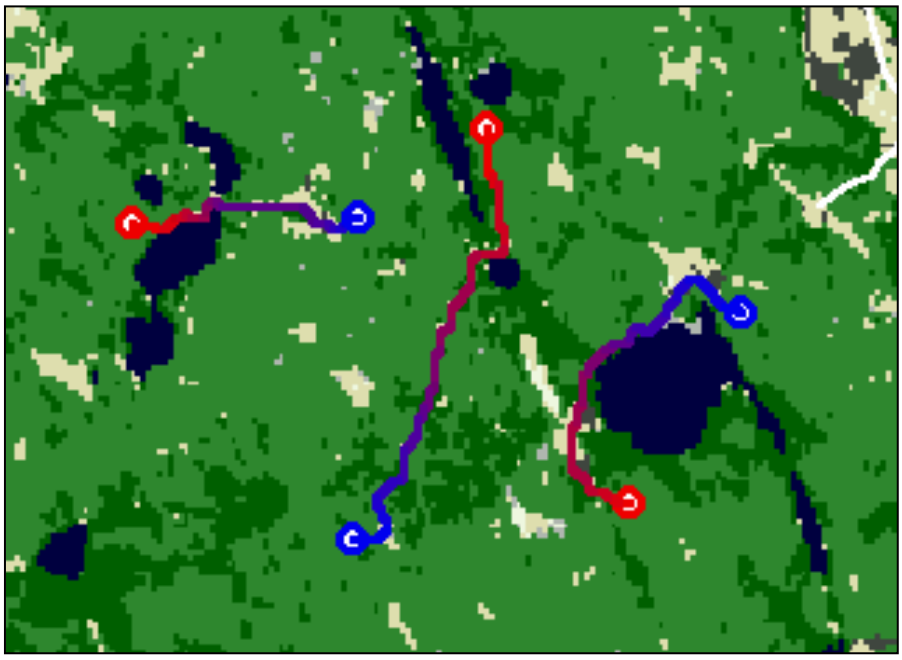
- Elevation. This is the easiest issue. Jönsson used the mean value of the elevation from the 16 cells.
- Terrain. Jönsson chose to "select the most 'populous' terrain type," and if there was, "ambiguity the 'least costly'."
- Roads. The 'fastest' road is selected if more than one road type appears in a 4 x 4 subsampling. Then, "the coarse solution (found on the subsampled graph) is divided into 'segments'… Then the optimum path between the first segment's end points… is found using the A* algorithm on the original, 'fine' graph. This is repeated for the other segments and all the 'fine' segment solutions [are] finally pasted together into a complete solution." [9]



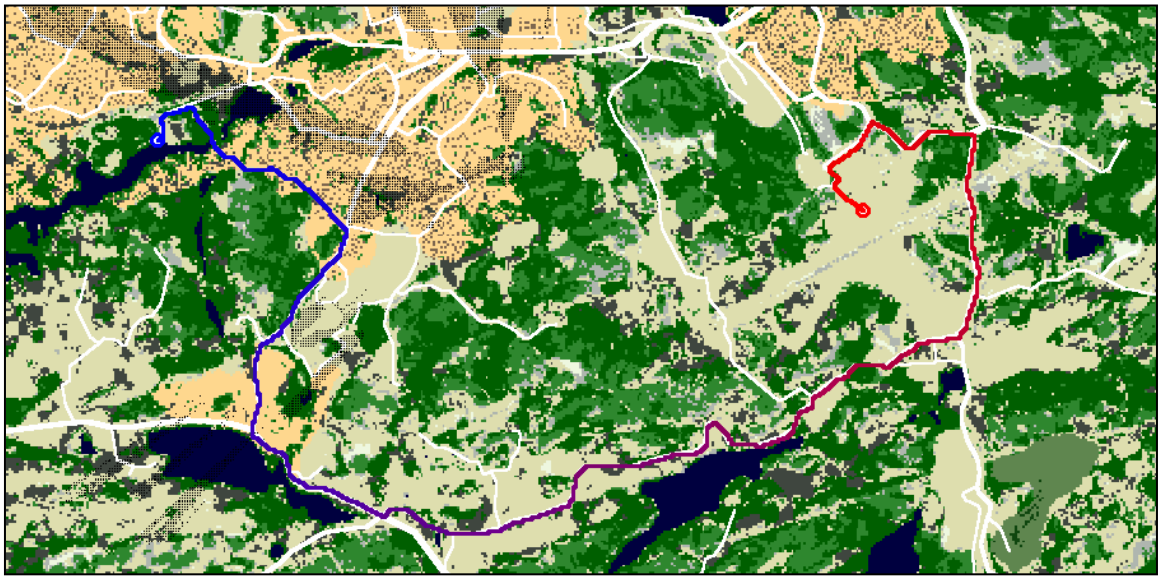
Example from Jönsson which shows a path where the distance between source and destination is approximately 64km (the cell size is 25m). "The complete search graph contained approximately 8.5 million vertices. No enemies were present. Using the normal algorithm, it took 48s to compute on a standard Pentium 133 with 18Mb of free memory… Most of the time was taken up by the virtual memory management thrashing the hard disk. As a comparison, computing a 'coarse' solution on a factor 6 [i. e. a 6 x 6 vertex clumping] subsampled graph took only 2.5s!" [9]

6. Examples and Discussion

Examples of Jönsson's Optimal Pathfinding Algorithm:

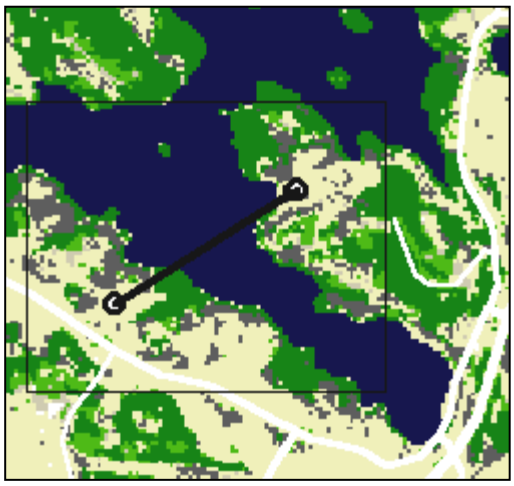


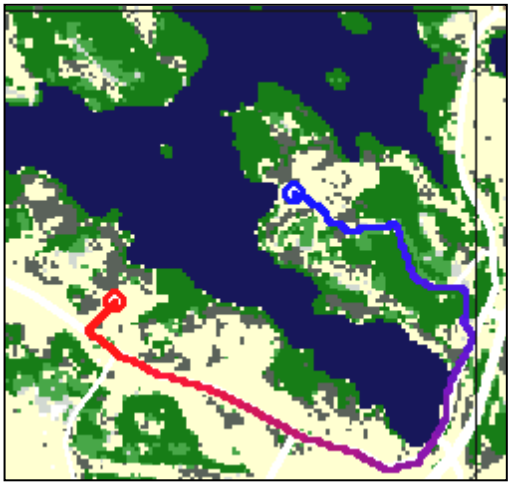
Obstacles in the form of lakes and 'bad' terrain are correctly circumnavigated. Above, the dark green is forest (very low speed traversal) and dark purple is water.



In the above example avoidance of enemy detection has forced the algorithm to take a southern route rather than the more direct route which crosses enemy observed locations (black dots).

Roads have a very low 'cost' in this example. Consequently the optimal path is not the most direct.





The above two images are examples of Jönsson's Algorithm for Restricting the Search Area. The image on the left shows the results of a search rectangle that is too small and, consequently, does not return a solution. In the image in the right the search area has been expanded and a solution has been found. Note: Jönsson's Optimal Path Algorithm does not automatically

expand the search area when a path solution is not found though it would be easy to add this feature.

Future work.
Jönsson acknowledges that the greatest shortcoming of the algorithm is its inability to deal with dynamic, i.e. moving enemy units, scenarios. He suggests that a variant of the progressive approximation method could be used to calculate 'course' solutions.

Another possibility, Jönsson suggests, is to "keep track of a time parameter along with the 'currently best path' parameter for every node during the A* search. This parameter could then be used together with the spatial position as an input to the enemy and other modifier."

Comments.
Jönsson's contribution to the field of optimal real-world terrain pathfinding algorithms is in combining numerous optimizations (some, such as his efficient graph representation are original, while others such as using a Fibonacci Heap instead of a Binary Heap are not) and publishing his results.

As noted previously, most work on optimal real-world pathfinding is done in the commercial computer game industry and, consequently, is not published. Jönsson's paper is certainly the best "all round" work covering this important subject that has been published in an academic setting and, for this reason alone, is significant.

One possible optimization for 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.

Also, adding at least an 'approximation of terrain costs' to the h(v) function would be beneficial.

REFERENCES

[1] Lard, John. *Bridging the Gap Between Developers & Researchers*. November 8, 2000. http://www.gamasutra.com/features/20001108/laird_03.htm.

[2] *Review of Star Wars: Knights of the Old Republic, PC*. http://www.gamersinfo.net/index.php?art/id:27

[3] Stefanakis, E. and Kavouras, M. *On the Determination of the Optimum Path in Space*. (1995). Proceedings of the European Conference on Spatial Information Theory COSIT 95.

[4] Fu, Jensen, Houlette . Human systems modeling: specifying the behavior of computer-generated forces without programming. (2003)

[5] Koenig. A Comparison of Fast Search Methods for Real-Time Situated Agents. (2004).

[6] Dijkstra, E. W., *A Note on Two Problems in Connexion with Graphs*. Numerische Mathematik 1, 269-271 (1959). Note: The title of this paper is frequently "Americanized"; however, I give the title as it appears on the copy of the paper that I have.

[7] Hart, P. E., Nilsson, N.J., Raphael, B., *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions of Systems Science and Cybernetics, Vol. SSC-4, No. 2, 100-107 (1968).

[8] Burgess, R. G., "Realistic Evaluation of Terrain by Intelligent Natural Agents (RETINA)," Master's Thesis, Naval Post Graduate School, Monterey, CA (2003).

[9] Jönsson, M. J., "An Optimal Pathfinder for Vehicles in Real-World Terrain Maps," Master's Thesis, The Royal Institute of Science, School of Engineering Physics, Stockholm, Sweden (2003). N. B. Markus Jönsson has changed his name to Markus Dimdal.

[10] Amit's Thoughts on Path-Finding and A-Star, http://theory.stanford.edu/~amitp/GameProgramming/

[11] *Introduction to Algorithms, Second Edition*. Cormen, T., Leiserson, C., Rivest, R., Stein, C. MIT Press, Cambridge, Massachusetts. 2001.

[12] "A Brief History of Computing & Wargaming", LeGuerrier, Volume 2, Number 2, D. E. Sidran

[13] Numbers, Predictions & War: The Use of History to Evaluate and Predict the Outcome of Armed Conflict; Trevor N. Dupuy; Hero Books; Fairfax, Va.; (1985)

[14] “Fibonacci Heaps.” Van Houdt, B.
http://www.win.ua.ac.be/~vanhoudt/graph/fibonacci.pdf