

Article

# Using the Hierarchical Pathfinding A\* Algorithm in GIS to Find Paths through Rasters with Nonuniform Traversal Cost

Harri Antikainen

Department of Geography, University of Oulu, P.O. Box 3000, 90014 Oulu, Finland;

E-Mail: harri.antikainen@oulu.fi; Tel.: +358-294-481708

*Received: 16 August 2013; in revised form: 22 September 2013 / Accepted: 7 October 2013 /*

*Published: 17 October 2013*

---

**Abstract:** A fair amount of research has been carried out on pathfinding problems in the context of transportation networks, whereas pathfinding in off-network space has received far less interest. In geographic information systems (GIS), the latter is usually associated with the cost surface method, which allows optimum paths to be calculated through rasters in which the value of each cell depicts the cost of traversal through that cell. One of the problems with this method is computational expense, which may be very high with large rasters. In this study, a pathfinding method called Hierarchical Pathfinding A\* (HPA\*), based on an abstraction strategy, is investigated as an alternative to the traditional approach. The aim of this study is to enhance the method to make it more suitable for calculating paths over cost rasters with nonuniform traversal cost. The method is implemented in GIS and tested with actual data. The results indicate that by taking into account the information embedded in the cost raster, paths of relatively good quality can be calculated while effecting significant savings in computational effort compared to the traditional, nonhierarchical approach.

**Keywords:** off-network pathfinding; cost rasters; hierarchical abstraction

---

## 1. Introduction

Pathfinding through transportation networks has become an established technology both in scientific as well as commercial applications. This is manifested in the vast amount of studies considering accessibility, traffic simulation and site selection problems, as well as in the ever-increasing popularity of automotive navigation, trip planning and fleet management systems. The

demand for pathfinding solutions also exists outside the domain of transportation networks, including applications related to movement and navigation in cross-country environments [1–3] and applications involving the alignment and construction of linear structures such as roads, pipelines or power transmission lines [4–8].

Regardless of the application domain, pathfinding is fundamentally based on graph theory and its concepts. A graph, typically denoted as

$$G = (N, E) \quad (1)$$

consists of a set of nodes (or vertices, points)  $N$  and a set of edges (or arcs, lines, links)  $E$ . An edge

$$e = (i, j) \in E \quad (2)$$

connects nodes  $i$  and  $j$ , and has an associated cost  $c_{ij}$ . The cost represents the impedance of travel, such as geographical distance or time, between  $i$  and  $j$ . An optimum, least-cost path between any two nodes of the graph can be calculated using a graph search algorithm [9,10].

Graph theory lends itself very well to representing real-world transportation networks as one-dimensional graphs, where the junctions of roads serve as nodes, and the roads are the edges of the graph. Edge costs can be easily derived from the physical length of the road segments, or from the estimated travel times along different roads. Unfortunately, casting off-network space into a similar representation is not equally straightforward. First, the suitability of different types of terrain for travel is often application specific, and the identification of appropriate traversal cost values may require laborious knowledge modeling. Secondly, and more importantly, movement in off-network space takes place in two dimensions rather than just one, as effectively is the case in transportation networks.

## 2. GIS and Cost Rasters

In the context of geographical information systems (GIS), pathfinding can be performed using either the vector or the raster data structure, both having their respective strengths and weaknesses. In the vector data structure, each object is composed of a series of coordinate pairs. This data structure is very useful for finding paths in situations where the environment consists of entities with well-defined boundaries. However, most often the task of determining paths through off-network space is based on the idea of cost surface analysis where space is represented by discrete and regular cells specifically associated with the raster data structure. The main advantages of the raster structure are that, due to their regular composition, rasters are easy to handle and process, and are also well suited for representing continuous spaces [11–13].

A raster-based representation of space can be transformed into a graph by treating the center point of each cell as a node, and connecting each such point to the center points of other cells. Since there are an infinite number of movement options in two-dimensional space, there are also multiple ways to approximate connectivity. The most commonly used connectivity pattern is to link a cell to its four orthogonal and four diagonal neighbors, amounting to eight movement angles [14]. The neighborhood can, of course, be extended beyond the contiguous neighbors, allowing for more directions of movement but resulting in an increasingly dense graph [15,16]. If the cells of the raster are assigned with cost values (defining the cost of movement per distance unit through the cell), the cost of each

step connecting the centers of two cells can be calculated according to the stretch of distance traveled through the underlying cells [17].

The raster graph allows the same algorithms used commonly in transportation networks to be utilized in calculating paths through off-network space. Typically the classic algorithm originally proposed by Dijkstra [18], or a variation of it, is used for this purpose. Dijkstra's algorithm is well-known in the literature and can be found in almost any book on algorithms (see, for example, [19]).

Dijkstra's algorithm essentially solves the single-source least-cost paths problem on a weighted graph  $G$ ; in other words, it produces a tree of least-cost paths from a start node  $s$  to all other nodes of  $G$ . The algorithm is initialized by assigning a tentative cost value  $c$  to all nodes: zero for  $s$ , and infinity for all other nodes. At the initial stage all nodes are also marked "open", denoting that they have not yet been visited by the algorithm. The algorithm then repeatedly selects and "expands" the node with the smallest  $c$  among the "open" nodes. This means that the algorithm scans all unvisited neighbors of  $i$ , and checks if the combined value of  $c_i$  and the cost of the edge connecting  $i$  and its neighbor  $j$  is smaller than the previously recorded tentative value  $c_j$ . If this is the case,  $c_j$  is updated with the new, smaller value. Once all the neighbors of  $i$  have been processed,  $i$  is marked "closed" and thus never visited again by the algorithm. The process is repeated until all nodes reachable from  $s$  have been visited, or if a path needs to be found between  $s$  and the destination node  $d$  only (rather than from  $s$  to all other nodes), it is sufficient to terminate the search as soon as  $d$  is reached. The path between the two nodes can be determined simply by backtracking the sequence of moves from  $d$  to  $s$ , which only requires that a pointer to the previous node be assigned to each node by the algorithm.

The running time of Dijkstra's algorithm, in its most basic form, is

$$O(N^2 + E) \quad (3)$$

where  $N$  and  $E$  stand for the number of nodes and edges, respectively. However, when the algorithm is implemented with a priority queue for fast retrieval of the currently cheapest node, the running time is reduced to

$$O((N + E) \log N) \quad (4)$$

assuming that the queue is implemented as a binary heap.

### 3. Strategies to Improve Raster-Based Pathfinding

There are two major drawbacks of raster-based pathfinding: one is the distortion inherently present in raster-based paths, and the other is the high computational effort needed to calculate the paths, especially when large rasters are concerned. The distortion, provoked by the neighborhood pattern used to create the raster graph, manifests itself as elongation and deviation error. The elongation error is due to the raster-based path unavoidably proceeding from one cell center to the center of one of the adjacent cells. Accordingly, there is a discrepancy between distance measured in the raster space and continuous space: the former tends to be greater than the latter. The deviation error, in turn, is the distance of the raster path from the corresponding continuous-space path. Considering the hypothetical case of a uniform cost raster, the deviation error is maximized when the path consists of an equal number of moves in orthogonal and diagonal directions, and when the orthogonal moves are made first followed by the diagonal moves [15].

The use of larger neighborhoods is one conventional strategy to decrease the distortion in paths induced by the raster structure. Instead of the conventional eight-connected pattern, larger patterns embracing 16, 32 or even more neighbors have been used [14–16,20]. Even though distortion can be decreased in this manner, it comes with the cost of increasingly dense graphs and associated computational expense. Another way of addressing the problem is to correct the calculated paths by replacing the distorted segments of the path with straight lines, whenever possible [20,21]. This results in more intuition, and also more useful paths.

The other major drawback of raster-based pathfinding—high computational cost—is a general challenge concerning all kinds of graph search problems. However, the problem is aggravated for raster-based graphs due to their high density. A raster typically needs to have a relatively high resolution in order to provide an adequately accurate representation of the environment. As each cell is treated as a node, this translates into a large number of nodes in the graph. In contrast, for graphs representing transportation networks, a relatively small number of nodes are needed to represent the network, each node being typically connected to two or three nodes only.

Although Dijkstra’s algorithm is rarely used in its basic form without employing more efficient internal data structures, finding a path with Dijkstra’s algorithm is nevertheless computationally demanding, especially with large graphs. The computational efficiency of pathfinding can be improved with heuristic strategies, which can be classified into three categories: limiting the search area, decomposing the search problem, and using an “abstraction” problem-solving strategy [22]. The idea behind limiting the search area is to make use of some knowledge about the start and destination locations to constrain the search to a least-cost path within a certain area. This addresses one of the main disadvantages of Dijkstra’s algorithm, that is, that it propagates the search in all directions instead of the actual direction of the destination. The A\* algorithm, originally proposed by Hart *et al.* [23], is typically used to address this problem. Dijkstra and A\* are effectively similar, with the exception that the latter performs an informed search based on a heuristic evaluation function. The function is denoted as

$$f(n) = g(n) + h(n) \quad (5)$$

where  $g(n)$  gives the path cost from  $s$  to  $n$ , and  $h(n)$  is the estimated cost to get from  $n$  to  $d$  [24]. If the heuristic estimate employed by the algorithm is admissible, signifying that the distance from  $n$  to  $d$  is never overestimated, the algorithm is guaranteed to find an optimum path [25]. A\* is considered to be the best choice in its class of algorithms: no other optimal algorithm is guaranteed to expand fewer nodes than A\*. However, this does not always guarantee fast search, because for most problems, the number of nodes within the search space is still exponential in the length of the path [24].

Both Dijkstra and A\* are deterministic algorithms that follow specific rules in their operation, and thus their behavior is completely predictable [26]. As an alternative to these methods, stochastic methods employing probabilistic search rules have been used to solve pathfinding problems. For example, ant colony optimization (ACO), inspired by the behavior of ants [27], has been used to find optimal paths on terrain images [28] and in indoor environments [29]. Genetic algorithms (GA), drawing on the principles of natural selection [30], have also been used to address shortest path [31], corridor location [32] and re-routing problems [33]. According to a performance evaluation of Dijkstra, A\* and GA [26], the use of a stochastic search method can be a more efficient choice

compared to the deterministic methods, especially for large-scale pathfinding problems. However, due to the tractability and predictability of the deterministic methods, it is likely that especially A\* will remain a popular choice for pathfinding, provided that techniques improving its performance can be utilized.

Bidirectional search, which belongs to the category of techniques aimed at decomposing the search problem, is one available option for limiting the search effort of the A\* algorithm. In this technique, two searches are run simultaneously: one from  $s$  towards  $d$ , and one in the opposite direction. An optimum path is found once these two searches meet each other somewhere in the middle. The rationale for employing bidirectional rather than unidirectional search is that whereas a search tree grows exponentially by its depth, the combined number of processed nodes by two smaller searches can be, at least in principle, less than the number of nodes processed by a single large search [34–36]. The challenge with bidirectional search is that there is no guarantee an optimum path between  $s$  and  $d$  has been found when the two search frontiers meet. The search frontiers may therefore have to significantly overlap each other, resulting in performance inferior to conventional unidirectional search. While several solutions have been proposed to provide improved efficiency compared to more conventional bidirectional search [36–38], their utility has not been extensively tested with real-world datasets [22].

Although the graph search process can be trimmed using the techniques described above, they do not always provide a satisfactory solution to the computational challenge posed by large graphs. Therefore, instead of attempting to modify the search algorithm itself, an alternative way to seek improved performance is to decrease the size of the graph. This may involve creating an abstraction of the problem, thereby retaining only the essential features of the problem and discarding the irrelevant details. For rasters and raster graphs, there are several standard techniques that may be used to create an abstraction. The most straightforward way is to reduce the resolution of the raster by means of resampling, and then use the coarse raster to construct the raster graph. While this may indeed result in a dramatic reduction in the size of the raster graph and computation time, its obvious drawback is the loss of information and detail. One problem in particular is related to obstacles and narrow passageways surrounded by obstacles, which may be lost completely at coarsened resolutions. As a consequence, the pathfinding procedure may fail to find any path (even if one exists in reality), or may propose a path through an inaccessible region. This problem can be addressed by transforming the raster into a quadtree representation [39] in which a group of cells is subdivided, step by step, into four smaller groups until each quadrant corresponds to a uniform block of cells. Compared to using the original raster for pathfinding, a quadtree-based search for an optimum path may be substantially faster, as the number of cells (nodes) is potentially considerably lower in the quadtree representation [40]. However, this approach may yield suboptimal results when the cells are large, requiring the postprocessing of paths, the use of “framed” quadtrees, or alternative node placement strategies [16,20,41].

A commonly applied solution for creating abstractions, especially in the context of transportation networks, is to exploit hierarchies. One option is to utilize the natural hierarchies present in transportation networks. By favoring primary roads and disregarding irrelevant local details, computation time can be reduced significantly. A hierarchy may also refer to a series of successive abstractions of the network that are specifically created for pathfinding purposes. There are many mechanisms for creating such abstractions, as explained in a review by Sturtevant and Jansen [42].

While certain techniques, such as clique-based hierarchy, are best suited for general graphs where the topological relationships of the nodes are explicitly defined, the so-called sector-based hierarchy is particularly useful for raster graphs, where the topological relationships are implicit and portray a regular structure.

Hierarchical Pathfinding A\* (HPA\*), proposed by Botea *et al.* [43] and further improved by Jansen and Buro [44] and Harabor and Botea [45], is a pathfinding method that incorporates two heuristic strategies (the A\* algorithm and an abstraction strategy) to improve the computational performance of raster-based pathfinding. The fundamental idea of creating an abstraction in the HPA\* method is based on dividing a raster into a set of *blocks* of equal size (or, as Li *et al.* [46] propose, into irregular-sized blocks based on decision tree). This initial phase is illustrated in Figure 1a. Then, for each border between two adjacent blocks, the obstacle-free segments along the border are identified (Figure 1b). These segments are the *entrances* between the blocks. Movement through an entrance is only allowed via *transitions*, which are defined for each entrance according to a predetermined transition placement scheme. A transition consists of two nodes, each placed on one side of the border between two adjacent blocks. An edge is drawn between these border nodes, connecting the blocks together (Figure 1c). Edges of this type are called “inter-edges”. Another type of edges, “intra-edges”, are defined between all pairs of border nodes within each block by calculating the least-cost paths between them using eight-connectivity (Figure 1d). The border nodes, together with the two kinds of edges, constitute the graph at the level of the block partition. Obviously, before using the graph to calculate a path between locations *s* and *d* (which may be located anywhere on the raster) these locations need to be temporarily connected to the graph. This is done by calculating least-cost paths from *s* to the border nodes of the block containing *s* and repeating the same procedure for *d* as well (Figure 1e). Finally, a path between *s* and *d* can be calculated on the graph using the A\* algorithm (Figure 1f).

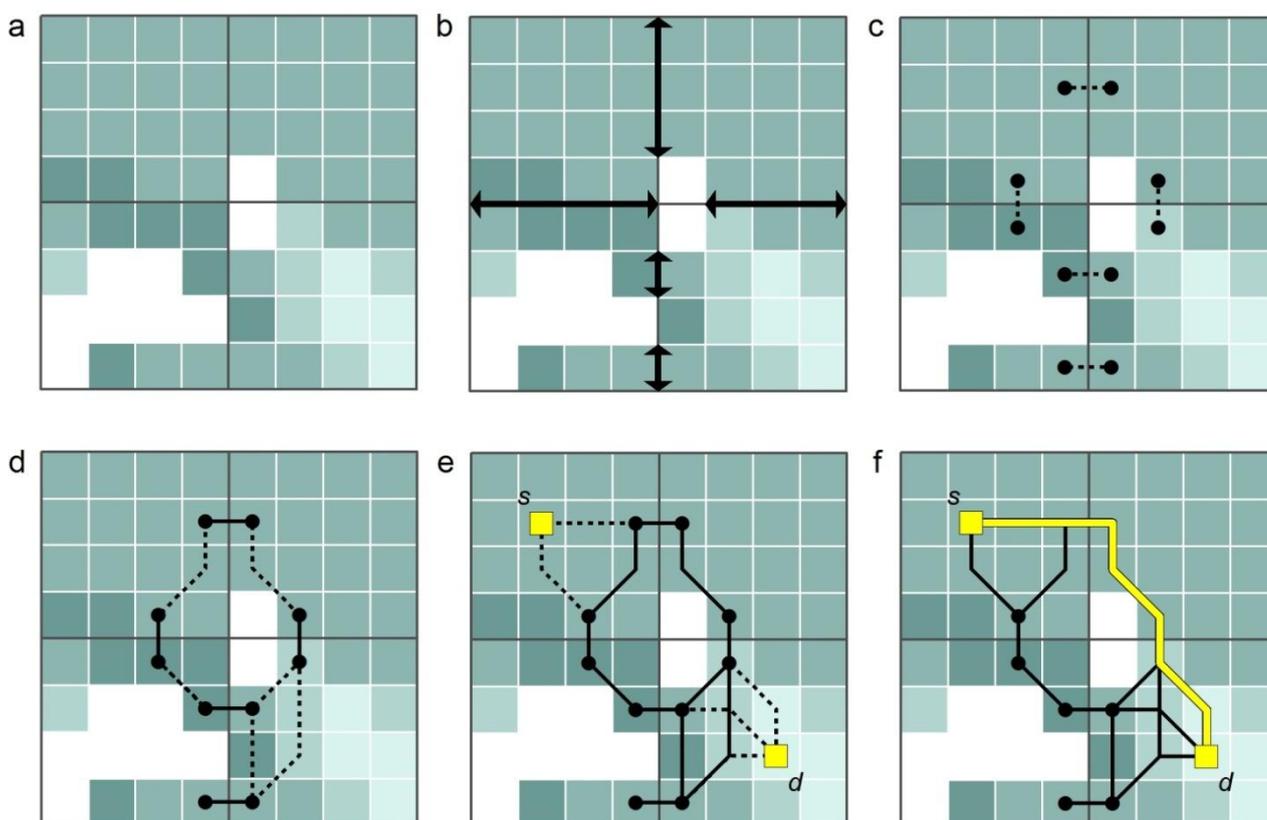
The utility of HPA\* is that a great deal of computation can be done in the preprocessing stage (phases a–d in Figure 1), making the actual pathfinding task much faster. When a path is requested between locations *s* and *d*, all that is needed is to temporarily connect them to the precalculated graph by making small Dijkstra searches on the original cost raster within the blocks containing *s* and *d*, and then calculate a path between them on the graph using A\*.

The overall computational effort of the pathfinding phase is determined by the block size. A large block size translates to a light-weight graph with a relatively small number of nodes. This saves time in the pathfinding phase, but the drawback of large block size is the increased effort needed to connect *s* and *d* to the graph. Conversely, small block size reduces the time needed to connect the locations to the graph, but the graph itself will contain more nodes, thus increasing computational cost. In order to address this problem of trade-off, a multi-level graph can be constructed. The benefit of a multi-level graph is that it allows paths to be calculated quickly by taking advantage of high abstraction levels, while providing a base level of small blocks that prevents the need for making large searches on the original raster to connect *s* and *d* to the graph.

In addition to block size, the placement pattern of transitions is an important parameter affecting the quality of the paths produced by the method. In their study, Botea *et al.* [43] suggest placing the transition in the middle of an entrance, if the width of the entrance is less than six cells. Otherwise, two transitions are defined, one on each end of the entrance. It is important to realize that this placement pattern is intended to be used for binary rasters consisting only of free space with uniform passage cost

and impassable obstacles. For rasters of this kind, the positioning of the transitions is not a highly critical factor, as the calculated paths can be corrected later by replacing the unnecessarily curved parts of the paths with straight shortcut lines, provided that the shortcuts do not intersect any obstacles. However, most real-world datasets involve a wide spectrum of traversability conditions which cannot be reduced to two classes. In such a case, a transition placement strategy based on the presumption of a binary division of space may not be very useful. This is true firstly because without taking into account the underlying traversal cost, the transition may end up being located at a position rarely used by any truly optimal path. Secondly, due to the varying cost of the cells, the paths generally cannot be improved by replacing curved parts of the path with straight shortcuts in the postprocessing phase. In other words, the transitions should be placed at the most appropriate locations in the first place, as the alignment of the path cannot be efficiently improved later.

**Figure 1.** The scheme of the HPA\* method: (a) dividing the cost raster into blocks, (b) identifying the entrances along the block borders, (c) determining a transition (inter-edge) to represent each entrance, (d) connecting the transition nodes within each block (intra-edges), (e) connecting *s* (start) and *d* (destination) temporarily to the graph, and (f) finding a path between *s* and *d*.



Although the HPA\* method, and raster-based pathfinding in general, has been widely studied in the literature, most of the research has been carried out in the fields of robotics, operations research and computer games. In these fields, the emphasis is usually on obstacle avoidance problems in restricted environments, whereas in geography and GIS the pathfinding typically involves varied traversal cost

and large geographical areas. However, the standard raster-based pathfinding method, regardless of its limitations, is usually used in GIS without considering any alternative methods.

Motivated by the apparent lack of research, the intended contribution of this study is to propose an improvement to the HPA\* algorithm which allows the transitions to be placed in a more informed manner than has been proposed in previous studies. The entire proposed algorithm is then implemented in GIS and tested and evaluated with actual data. The contribution of this study should be of particular interest for geography and GIS, and related application areas where paths need to be optimized in terms of varying traversal cost.

#### 4. Implemented Method

Experimental implementation of the HPA\* method was made for this study using the C# programming language and the ArcObjects components of the ArcGIS software. The basic HPA\* algorithm, along with its pseudocode, has been described in detail in [43] and will not be reiterated here. The major parts of the implemented method, and the contribution made by this work, are described below.

##### 4.1. Transition Placement Strategies

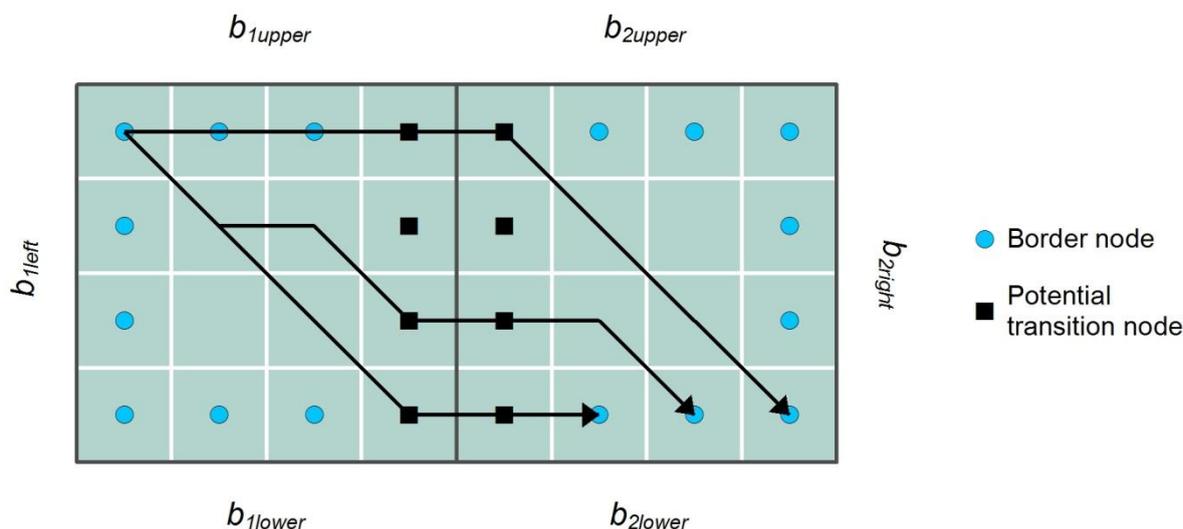
As described above, transitions play an important role in the HPA\* method. Since access from one block to another is only allowed through transitions, and since paths between any start and destination must “share” them, transitions greatly affect the quality of paths. Three different strategies for placing a transition on an entrance in the HPA\* method are considered in this study. These strategies are based on the idea of placing a transition either in the middle of an entrance, at the position of lowest cost, or at the position of best accessibility and will be referred to as one-letter acronyms, as M, C, and A, respectively.

The M strategy is directly adopted from Botea *et al.* [43]. Despite being simplistic, it has certain advantages. The first advantage is its tractability: no complicated assessment of cell costs needs to be performed. Secondly, a transition placed in the middle of an entrance can be relatively efficient in minimizing the potential error, especially when the underlying cost is uniform over the whole width of the entrance.

The flaw of the M strategy lies in the fact that optimal paths tend to favor areas of low traversal cost, particularly corridors of low cost. The M strategy does not take this into account, and therefore there is a high chance that the location of the transition deviates from locations where the truly optimum paths cross the block border. In contrast, the C strategy aims at placing the transition at the position of the lowest traversal cost along an entrance. This procedure is not significantly more complicated than the M method as only the costs of the cells on the two sides of the entrance need to be evaluated. This simplicity is, again, also its disadvantage. The position of lowest transition cost along the border may only be an isolated case or part of a local cluster of low-cost cells rather than a larger area or corridor of low traversal cost. Particularly problematic are local clusters of low-cost cells near either end of an entrance, as this may force the paths to make unnecessary twists and thus impair path quality.

The limitations of the M and C strategies can be addressed by the third transition placement strategy presented here, namely the strategy based on determining the location of best accessibility (A). The objective of the A strategy is to more accurately predict where paths passing from a block to an adjacent block will cross an entrance between the two. Here, accessibility generally refers to the overall cost of movement between a transition candidate on the border between two blocks and the other borders of the blocks. There are different ways of assessing the accessibility of a transition candidate. One of the most straightforward options would be to calculate least-cost paths from both nodes comprising the transitions to the border cells of their respective blocks and then use this information to determine an accessibility score. The problem with this approach is that locations in the middle of the block border tend to naturally receive high accessibility scores, and as a consequence the transition placement pattern of A resembles that of M, resulting in a fundamentally similar abstraction. Therefore, a different strategy needs to be adopted to determine accessibility scores. The idea is to estimate where least-cost paths passing from one block to another most likely cross the border between the two. To determine this, least-cost paths are calculated between the cells of “opposite” block edges for each pair of adjacent blocks sharing a common border. This is illustrated in Figure 2 for a case of horizontally adjacent blocks,  $B_1$  and  $B_2$ , with the size of  $4 \times 4$  cells. The procedure is performed between all pairs of cells along borders  $b_{1upper}$  and  $b_{2lower}$ ,  $b_{1lower}$  and  $b_{2upper}$ , and  $b_{1left}$  and  $b_{2right}$ , excluding the cells assigned the role of a potential transition node. Once all the paths connecting the cells at the opposite borders of the blocks are calculated, the accessibility score is determined for each potential transition node by summing up the number of paths crossing each node. The transition will be created between those nodes that have the highest combined accessibility score. If two or more potential transitions along the same entrance happen to have an identical score, the centermost of these candidates serves as the transition.

**Figure 2.** The principle used to determine accessibility scores. For illustration purposes, paths are shown only between the border cell at the top left corner of the first block and cells along the opposite, lower border of the neighboring block. The calculated score indicates the potential transitions which will most likely be shared by different paths, and the transition is placed at the position of the highest score (in the case of a tie, the centermost candidate is chosen).



## 4.2. Hierarchical Abstraction

In addition to the placement of transitions, the operation of the method is mostly dependent on the choice of block size and the number of hierarchical levels above the base level. The implemented method receives both of these parameters as user input. First, the raster is divided into blocks according to the specified base block size (e.g.,  $20 \times 20$  cells). Using this division, transitions are placed on the entrances between adjacent blocks and the graph is constructed.

If the hierarchy consists only of the base level  $L_1$ , this is all that needs to be done in the preprocessing phase. When more than one hierarchical level is requested, the levels are constructed according to the quadtree structure such that each block at a higher level consists of four blocks at one level below. For example, with a base block size of  $20 \times 20$  cells and the number of levels set to three, the block size of the levels  $L_1$ ,  $L_2$  and  $L_3$  would be  $20 \times 20$ ,  $40 \times 40$ , and  $80 \times 80$  cells, respectively. The higher levels are always constructed on top of the graph defined at the base level, sharing the same nodes and edges. However, the intra-edges of larger blocks are constructed by concatenating the lower-level edges to constitute direct connections between their border nodes. This procedure guarantees that while generating higher-level abstractions, path quality is not affected by the number of hierarchical levels defined above the base level. It is worth noting that the quadtree structure used here is by no means the only conceivable option to construct the succession of hierarchies. The quadtree structure was chosen since it is well-known in the literature and easy to implement and manage, but there are no compelling reasons to restrict the hierarchical structure specifically to this structure.

Besides the base block size and the number of levels, the user may also determine any number of transitions for an entrance by defining the maximum number of cells that can be represented by a single transition. For each entrance, the width of the entrance is divided by this value and the ceiling of the quotient is used to divide the entrance into segments of equal width. A transition is then placed separately for each segment, according to the chosen transition placement strategy.

## 4.3. Graph Search

Analogous to finding a route through a large road network, the most reasonable way of finding the path with the HPA\* method is to utilize the highest level of the hierarchy to the maximum extent possible. This allows the method to discard most of the nodes and edges at the lower levels, completing the calculation more quickly. On the basis of this principle, the pathfinding procedure has been implemented as follows.

The algorithm constructs a temporary graph  $G_t$  which contains only those parts of the original hierarchy of graphs that are necessary for finding a path between two given locations  $s$  and  $d$ . First, the highest level of the hierarchy serves as the basis for  $G_t$  (here the highest level  $L_l$  is included in the temporary graph completely, but in the case of very large areas and graphs, the extent to which the highest level abstraction is included in  $G_t$  should be delimited according to some criteria). Secondly, the block containing  $s$  at level  $L_l$  ( $B_o$ ) is identified. The subgraphs associated with the four blocks from level  $L_{l-1}$ , which are contained in  $B_o$ , are added to  $G_t$ . The process is successively continued downwards throughout the hierarchy until the base level ( $L_1$ ) is reached. At the base level,  $s$  is connected to  $G_t$  by calculating least-cost paths using Dijkstra's algorithm through the original cost

raster to the border nodes of the base level block containing  $s$ . The same procedure is of course performed for  $d$  as well. This effectively means that only two small searches on the original cost raster are needed to connect  $s$  and  $d$  to the base level  $L_1$ , and from that level, they are connected to successively higher levels of the hierarchy.

Once the graph structure  $G_t$  has been completed, calculating a path between  $s$  and  $d$  is a simple matter of running the A\* algorithm on  $G_t$ . In this implementation, the function of the estimated distance that is used to guide the A\* search in the general direction of the destination node is based on the product of the Euclidean distance between  $n$  and  $d$ , and the lowest unit cost value  $c_{min}$  present in the cost raster. The graph search can be done both unidirectionally and bidirectionally; however, since initial tests indicated that no improvement was gained by using the bidirectional search technique, only unidirectional search was included in the final implementation. The algorithm was implemented with a priority queue using a binary heap.

## 5. Test Case

The testing of the implemented method was done with a mosaicked Landsat image which has been classified with maximum likelihood classification and assigned cost values taken from [3]. Three smaller cost rasters were selected arbitrarily from an area enclosed between 25.5°–27.3° E and 63.9°–65.7° N (in central Finland). Each one of these three rasters consists of 500 × 500 cells at a resolution of 30 m, thus covering an area of 225 km<sup>2</sup>. Twenty-five points were chosen for each raster to represent those locations between which paths were to be calculated. The selection of these points was not completely random: first, only the centers of cells were considered as potential points. Secondly, a minimum distance of 2,000 m was required between any two points. This was done to ensure that each block contains no more than one point, so as to allow proper testing of transition placement. For each of the three rasters, paths were calculated between each pair of points. As the traversal cost is assumed to be isotropic, it was sufficient to calculate the paths in one direction only. Hence, the number of paths per cost raster is

$$m(m - 1)/2 \quad (6)$$

where  $m$  is the number of locations. With twenty-five locations, the number of paths is thus 300. The paths were first calculated with the conventional, nonhierarchical method, providing the true optimum paths in terms of the raster structure. These serve as the benchmark which the paths obtained with the HPA\* method can be compared to.

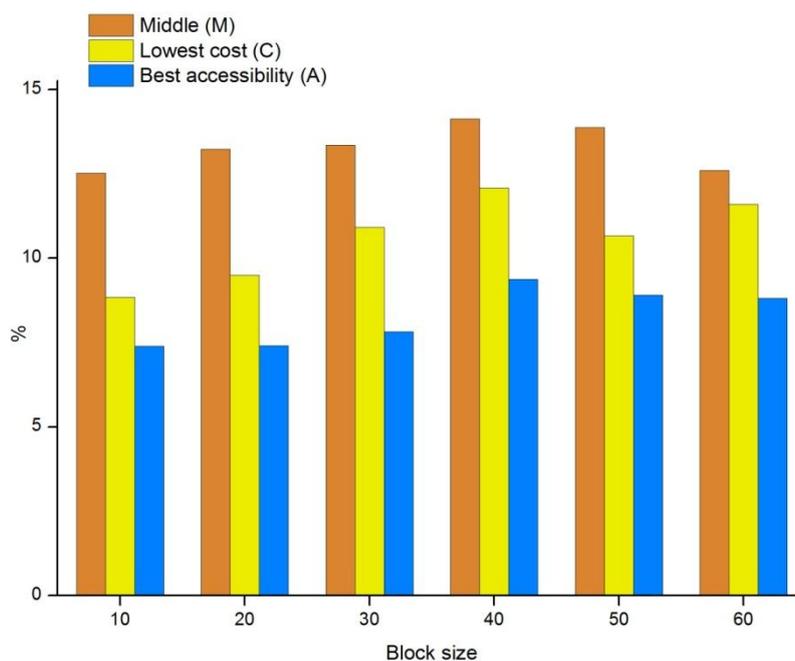
The objective of the test case is to represent results for three different transition placement strategies of the HPA\* method together with six different base block sizes (10 × 10, 20 × 20, 30 × 30, 40 × 40, 50 × 50 and 60 × 60 cells), and assess the path quality (measured as the difference compared to paths calculated using the traditional, non-hierarchical approach). Although using a higher number of transitions can improve path quality, in this study the method was only tested with the aim of minimizing the size of the graph, which amounts to representing an entrance by a single transition. In addition to path quality, computational expense was also evaluated in terms of the number of nodes processed in the actual pathfinding stage and nodes processed when connecting locations  $s$  and  $d$  into  $G_t$ . Finally, the effect of increasing the number of abstraction levels on computational effort was investigated.

## 6. Results

### 6.1. Path Quality

The quality of the paths calculated on the test rasters using the HPA\* method with three different transition placement options and six different block sizes is summarized in Figure 3. Path quality associated with each combination is expressed as error compared to the true optimum paths calculated with the traditional, nonhierarchical method. In other words, the shorter the bar, the smaller the difference, and the better the method in question can be considered to perform.

**Figure 3.** The error in paths calculated with the HPA\* method using three different transition placement options and six different block sizes, compared to true optimum paths calculated with the conventional nonhierarchical method.



The error depicted in Figure 3 ranges from about 7% up to 14%, depending mostly on the transition placement method, and to a lesser degree, on the chosen block size. As expected, the error is greatest with the transition placement option M (middle); both C (lowest cost) and A (best accessibility) options provide better results. The A method can provide up to a 20% reduction in error compared to C, and a 50% reduction compared to M. To put the observed deviations into a relevant perspective, they can be compared to the maximum elongation error of 8.24% [15] inherently associated with the conventionally used eight-connected neighborhood pattern. As it appears, A is the only transition placement technique with an error consistently smaller than this. Of course, the error incurred by the HPA\* method is added to the elongation error already present in raster-based paths, but it is important to realize that the added error of the A strategy is in fact less than the elongation error commonly regarded as acceptable.

The effect of the block size on path quality is dependent on the chosen transition placement strategy. As the results indicate, M and A appear to be less sensitive to block size than C. This might be attributable to the high chance of badly positioned transitions when the C strategy is used, which is

amplified by large block sizes and wide entrances. This is an important aspect, as block size is a key factor determining the computational expense of the method.

## 6.2. Computational Performance

There are two aspects concerning the investigated method: preprocessing and the actual pathfinding phase. While the latter is obviously the more critical one, the performance of the former is also important in real-world applications.

It is necessary to emphasize, though, that in this study the HPA\* method has been implemented for experimental purposes so as to enable comparison between methods rather than to achieve high performance in absolute terms. For example, calculating a path directly on one of the test rasters without constructing a hierarchical abstraction first may take up to half an hour with the implemented method. This is a very long time considering that performing the exact same calculation using the built-in cost surface functions of state-of-the-art GIS software, such as ArcGIS, takes only a few seconds. From this perspective, it is quite remarkable that the implemented HPA\* method manages to calculate a path in just about 10 seconds or less (including the construction of the temporary graph and connecting the start and destinations points to it) once the preprocessing is done.

The preprocessing of any of test rasters used in this study takes about ten minutes using a desktop computer with a 3.30 GHz processor and 8 GB of memory. This is true of the M and C strategies, for which the preprocessing time is, in principle, independent of the base block size chosen. In contrast, with the A strategy, the corresponding time is several hours, and it increases with block size. By introducing more hierarchical levels, the preprocessing time is increased slightly for all strategy options and the path calculation time becomes respectively shorter.

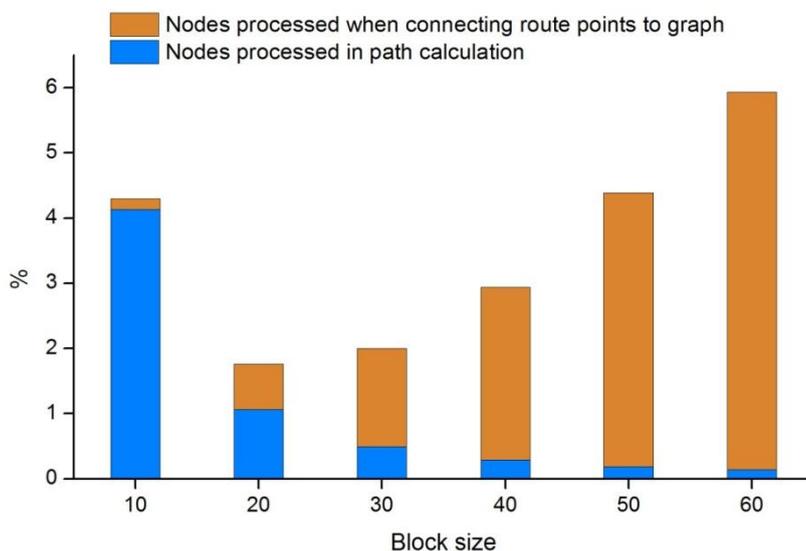
Due to the fact that the computational effort needed to calculate paths is always dependent on the properties of the software implementations and hardware used to carry out the calculations, it is not very useful to assess absolute processing times to evaluate performance. Instead, a better way is to assess the number of nodes processed by the different methods, since the computational effort of pathfinding is more or less directly dependent on the number of nodes that must be visited during the graph search.

In terms of the number of processed nodes, the HPA\* method can deliver up to a more than 95% reduction when compared to the traditional, nonhierarchical approach (Figure 4). While this reduction can be achieved irrespective of the transition placement strategy, the exact amount of potential improvement depends on the block size. On the other hand, due to the nature of the complexity function of the graph search, the resultant reduction in computation time may be even higher than what is indicated by the reduction in the number of nodes.

Figure 4 also illustrates how the total number of processed nodes is shared between nodes processed in connecting  $s$  and  $d$  to the graph, and those processed in performing the calculation of the actual path. Evidently, the computational cost of connecting  $s$  and  $d$  to the graph is low with small block sizes because the number of cells contained by a single block is small. However, as the block size increases, the computational cost grows rapidly. The exact opposite is true for the computational cost of the pathfinding phase. With a small block size, the graph consists of a larger number of nodes and edges than is the case with larger block sizes. When the HPA\* method is used with a single block level

only, the optimum block size in terms of computational cost is the one that minimizes the overall number of processed nodes.

**Figure 4.** The total number of processed nodes in the HPA\* method with six different block sizes. The bars represent the percentage of processed nodes compared to the traditional, nonhierarchical method, separately for nodes processed in the actual pathfinding process and nodes processed to temporarily connect  $s$  and  $d$  to the graph.



The computational cost of connecting  $s$  and  $d$  to the graph is almost completely dependent on the block size, and thus highly predictable. However, the computational cost of the actual pathfinding phase is more dependent on the data, which determines the density of transitions along the block borders. If a raster is divided into  $c \times r$  blocks, the minimum number of transitions is

$$r(c - 1) + c(r - 1) \quad (7)$$

As each transition consists of two nodes, the number of nodes is then

$$2(2cr - r - c) \quad (8)$$

This is true when each border between adjacent blocks encompasses only a single entrance, which is always the case for obstacle-free rasters. On the other extreme, the number of transitions and nodes are maximized when obstacle cells divide each border into a maximum possible number of separate entrances. In such a case, the number of nodes is

$$2(2cr - r - c) \left\lceil \frac{w}{2} \right\rceil \quad (9)$$

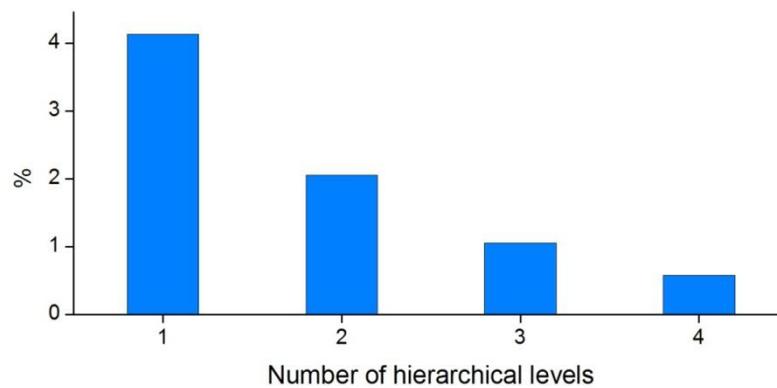
where  $w$  is the block width in cells.

### 6.3. Multiple-Level Hierarchies

Instead of attempting to determine a single optimal block size, it is more useful to take advantage of the ability of HPA\* to work with several levels of abstraction. The use of a small base block size allows  $s$  and  $d$  be connected to the graph quickly, while higher level abstractions are used to quickly

find a path between them. Figure 5 shows how the number of processed nodes is dependent on the number of hierarchical levels used. With the data used in this study and a base block size of  $10 \times 10$  cells, the number of processed nodes is just slightly over four percent of that of the traditional nonhierarchical method. To put it another way, HPA\* delivers a more than 95% reduction in the number of processed nodes.

**Figure 5.** The percentage of processed nodes in graph search performed using the HPA\* method with different number of hierarchical levels compared to the traditional, nonhierarchical method. The base block size is  $10 \times 10$  cells.



When more levels of abstraction are introduced, the number of processed nodes is further reduced. With a three-level hierarchy, the number of processed nodes is reduced to one-fourth of that of the 10-cell base level. As the path quality remains intact regardless of the number of hierarchical levels, the only drawback of using more hierarchies is the overhead incurred by the handling of several levels, which affects the preprocessing phase primarily.

## 7. Discussion

While the HPA\* method can provide a considerable computational advantage for pathfinding on large cost rasters, it also involves certain disadvantages that need to be considered. The main criticism of the method is the deviation of the paths from the true optimum, which is an unavoidable result of the abstraction. Even when this error is not very large, it adds to the error inherently present in raster-based paths. The error could be decreased by allowing more transitions per entrance, and thereby gain a more accurate representation of the movement options. Again, this comes at the price of increased computational cost in the pathfinding phase, which may be prohibitive in applications requiring immediate response.

An additional source of error stems from the fact that the transitions only allow block borders to be crossed in an orthogonal fashion. However, this is only a matter of implementation, as diagonal transitions could be taken account of, for example, by employing a node placement pattern in which nodes are placed on cell borders rather than cell centers [16,20]. By using this procedure, a transition would consist of only a single node located on the border between two blocks, offering more flexibility as to the directions in which the border can be crossed. This would also provide computational advantage by decreasing the number of nodes in the abstraction. However, the counterweight of this option would be a more complex preprocessing phase.

There are some additional aspects concerning the described method, or its implementation, that could be done differently. In this case, as well as in the implementation of the original authors of the HPA\* method, the higher level abstractions are built on the base level abstraction. There are two important advantages to this. Firstly, the construction of the higher level abstractions can be done quickly, and secondly, the results always remain the same irrespective of the number of hierarchical levels used. However, it could be possible to define transitions (and the intra-edges) independently for each level of abstraction. This would provide more flexibility in the placement of transitions, which in turn could improve path quality, especially at very high levels of abstraction. In fact, this procedure was tested at an early phase of the study with fairly promising results but was subsequently abandoned due to impractically long preprocessing time. An additional issue would be the consistency of the results, since the results would always depend on the number of abstraction levels used.

Another modification of the method might be to retain only a single transition per entrance at all levels. In the implemented method, all the transitions defined at the base level are transferred to higher levels as long as they correspond to the boundaries of the higher level blocks. However, this exceeds what is needed to ensure connectivity at higher levels. By redefining entrances at each level, and selecting only a single transition from the lower level to represent the entrance at the higher level, the size of the graph could be reduced significantly. An experiment with this procedure was done as well. The computational effort was indeed reduced, but as a negative effect, the error increased to an unacceptably high level of 15%–40%. A path calculated in this way could be refined in a postprocessing phase by successively recalculating the path at lower level abstractions using the crossed higher level blocks as a “mask” delimiting the search at the lower level. While the path can be refined in this manner, the computational expense incurred by the postprocessing phase cancels out any advantage gained by the sparcification of the graph.

## 8. Conclusions

The aim of the study was to demonstrate and evaluate the HPA\* method for pathfinding on cost rasters with nonuniform cost, and to propose an improvement in order to better approximate optimum paths on rasters of this kind. An experimental tool was implemented in GIS and was tested with data based on satellite imagery. A particular objective was to test whether a transition placement strategy based on the notion of accessibility is capable of producing better paths than the more rudimentary techniques.

The results indicated that the transition placement method based on accessibility score may indeed be a preferable technique compared to placing the transition either in the middle of the entrance or the position of lowest traversal cost. The results are in keeping with the original hypothesis that transition placement based on accessibility improves path quality, especially when the number of transitions is minimized. In fact, the advantage of this strategy is two-fold. First, according to the results, it uniformly provides *better* results than the other tested alternatives. Second, it also provides more *consistent* results, being less sensitive to the choice of block size. The downside of the proposed accessibility-based strategy compared to the other techniques is the extensive preprocessing phase, which may potentially make the technique impractical. Nevertheless, the preprocessing needs to be done only once as long as the original cost raster remains unchanged, and even if changes occur, the

preprocessing needs to be done only for the blocks affected by the changes. It would also be possible to modify the way that the accessibility score is calculated and thereby limit the number of paths considered, thus decreasing the computational expense but also degrading the quality of the results to some degree.

Overall, the HPA\* method is suitable for GIS applications in which off-network pathfinding has to be done with small computational resources and in a limited amount of time, or when paths need to be repeatedly calculated in a certain area, such as in off-road navigation and route planning applications. Although the method has been studied elsewhere, it is noteworthy that most of the research into the subject takes place in fields other than geography, such as in robotics and computer games, and that the methods developed in these fields may not always be ideally suited to the geographical context. This study represents an attempt to bring some of the knowledge existing in other fields to the domain of geography and GIS, and to extend it to better suit a wider variety of purposes.

### Acknowledgments

The research presented in this article was funded by the Academy of Finland.

### Conflict of Interest

The authors declare no conflict of interest.

### References

1. Balstrøm, T. On identifying the most time-saving walking route in a trackless mountainous terrain. *Geogr. Tidsskr.* **2002**, *102*, 51–58.
2. Rees, W.G. Least-cost paths in mountainous terrain. *Comput. Geosci.* **2004**, *30*, 203–209.
3. Store, R.; Antikainen, H. Using GIS-based multicriteria evaluation and path optimization for effective forest field inventory. *Comput. Environ. Urban* **2010**, *34*, 153–161.
4. Feldman, S.C.; Pelletier, R.E.; Walser, E.; Smoot, J.C.; Ahl, D. A prototype for pipeline routing using remotely sensed data and geographic information system analysis. *Remote Sens. Environ.* **1995**, *53*, 123–131.
5. Xiang, W.-N. A GIS based method for trail alignment planning. *Landsc. Urban Plan.* **1996**, *35*, 11–23.
6. Collischonn, W.; Pilar, J.V. A direction dependent least-cost-path algorithm for roads and canals. *Int. J. Geogr. Inf. Sci.* **2000**, *14*, 397–406.
7. Yu, C.; Lee, J.; Munro-Stasiuk, M.J. Extensions to least-cost path algorithms for roadway planning. *Int. J. Geogr. Inf. Sci.* **2003**, *17*, 361–376.
8. Bagli, S.; Geneletti, D.; Orsi, F. Routeing of power lines through least-cost path analysis and multicriteria evaluation to minimise environmental impacts. *Environ. Impact Asses.* **2011**, *31*, 234–239.
9. Diestel, R. *Graph Theory*, 2nd ed.; Springer-Verlag: New York, NY, USA, 2000; p. 2.
10. Miller, H.J.; Shaw, S.-L. GIS-T Data Models. In *Geographic Information Systems for Transportation*; Oxford University Press: New York, NY, USA, 2001; pp. 54–55.

11. Douglas, D.H. Least-cost path in GIS using an accumulated cost surface and slopelines. *Cartogr. Int. J. Geogr. Inf. Geovis.* **1994**, *31*, 37–51.
12. Goodchild, M.F. Geographic information systems and disaggregate transportation modeling. *Geogr. Syst.* **1998**, *5*, 19–44.
13. De Smith, M.J.; Goodchild, M.F.; Longley, P.A. *Geospatial Analysis*, 2nd ed.; Winchelsea Press: Leicester, UK, 2007; pp. 149–152.
14. Xu, J.; Lathrop, R.G., Jr. Improving simulation accuracy of spread phenomena in a raster-based geographic information system. *Int. J. Geogr. Inf. Syst.* **1995**, *9*, 153–168.
15. Goodchild, M.F. An evaluation of lattice solutions to the problem of corridor location. *Environ. Plan. A* **1977**, *9*, 727–738.
16. Van Bemmelen, J.; Quak, W.; van Hekken, M.; van Oosterom, P. Vector vs. Raster-Based Algorithms for Cross Country Movement Planning. In Proceedings of the 11th International Symposium on Computer-Assisted Cartography, Minneapolis, MN, USA, 31 October–1 November 1993; pp. 304–317.
17. Bolstad, B. Raster Analyses. In *GIS Fundamentals*; Eider Press: White Bear Lake, MN, USA, 2002; pp. 283–285.
18. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271.
19. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L. Single-Source Shortest Paths. In *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 1990; pp. 527–532.
20. Antikainen, H. Comparison of different strategies for determining raster-based least-cost paths with a minimum amount of distortion. *Trans. GIS* **2013**, *17*, 96–108.
21. Tomlin, D. Propagating radial waves of travel cost in a grid. *Int. J. Geogr. Inf. Sci.* **2010**, *24*, 1391–1413.
22. Fu, L.; Sun, D.; Rilett, R.L. Heuristic shortest path algorithms for transportation applications: State of the art. *Comput. Oper. Res.* **2006**, *33*, 3324–3343.
23. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107.
24. Russell, S.; Norvig, P. Informed Search and Exploration. In *Artificial Intelligence*, 2nd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 2003; pp. 97–101.
25. Pearl, J. Formal Properties of Heuristic Methods. In *Heuristics*; Addison-Wesley: Reading, MA, USA, 1984; pp. 77–78.
26. Soltani, A.R.; Tawfik, H.; Goulermas, J.Y.; Fernando, T. Path planning in construction sites: Performance evaluation of the Dijkstra, A\*, and GA search algorithms. *Adv. Eng. Inform.* **2002**, *16*, 291–303.
27. Dorigo, M.; Stützle, T. The Ant Colony Optimization Metaheuristic: Algorithms, Applications and Advances. In *Handbook of Metaheuristics*; Glover, F., Kochenberger, G.A., Eds.; Kluwer Academic Publishers: Secaucus, NJ, USA, 2002; pp. 251–285.
28. Rishiwal, V.; Yadav, M.; Arya, K.V. Finding optimal paths on terrain maps using ant colony algorithm. *Int. J. Comput. Theory Eng.* **2010**, *2*, 1793–8201.
29. Jaén, J.; Mocholí J.A.; Catalá A.; Navarro, E. Digital ants as the best cicerones for museum visitors. *Appl. Soft Comput.* **2011**, *11*, 111–119.

30. Goldberg, D.E. A Gentle Introduction to Genetic Algorithms. In *Genetic Algorithms in Search, Optimization, and Machine Learning*; Addison-Wesley: Boston, MA, USA, 1989; pp. 1–2.
31. Ahn, C.W.; Ramakrishna, R.S. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Trans. Evol. Comput.* **2002**, *6*, 566–579.
32. Zhang, X.; Armstrong, M.P. Genetic algorithms and the corridor location problem: Multiple objectives and alternative solutions. *Environ. Plan. B* **2008**, *35*, 148–168.
33. Lee, J.; Yang, J. DGA: A fast and scalable re-routing algorithm based on shortest path and genetic algorithms. *Int. J. Comput. Commun.* **2012**, *7*, 482–493.
34. Pohl, I. *Bi-Directional and Heuristic Search in Path Problems*; Technical Report 104; Stanford Linear Accelerator Center: Stanford, CA, USA, 1969.
35. Nilsson, N.J. Search Strategies for AI Production Systems. In *Principles of Artificial Intelligence*; Tioga Publishing: Palo Alto, CA, USA, 1980; pp. 88–90.
36. Kaindl, H.; Kainz, G. Bidirectional heuristic search reconsidered. *J. Artif. Intell. Res.* **1997**, *7*, 283–317.
37. De Champeaux, D.; Sint, L. An improved bidirectional heuristic search algorithm. *J. Assoc. Comput. Mach.* **1977**, *24*, 177–191.
38. Kwa, J.B.H. BS\*: An admissible bidirectional staged heuristic search algorithm. *Artif. Intell.* **1989**, *38*, 95–109.
39. Samet, H. Introduction. In *Applications of Spatial Data Structures*; Addison-Wesley: Reading, MA, USA, 1990; pp. 2–3.
40. Kambhampati, S.; Davis, L.S. Multiresolution path planning for mobile robots. *IEEE J. Robot. Autom.* **1986**, *2*, 135–145.
41. Chen, D.Z.; Szczerba, R.J.; Uhran, J.J., Jr. A framed-quadtrees approach for determining Euclidean shortest paths in a 2-D environment. *IEEE Trans. Robot. Autom.* **1997**, *13*, 668–681.
42. Sturtevant, N.; Jansen, R. An Analysis of Map-Based Abstraction and Refinement. In Proceedings of the 7th International Symposium on Abstraction, Reformulation and Approximation, Whistler, Canada, 18–21 July 2007; pp. 344–358.
43. Botea, A.; Müller, M.; Schaeffer, J. Near optimal hierarchical path-finding. *J. Game Dev.* **2004**, *1*, 7–28.
44. Jansen, M.R.; Buro, M. HPA\* Enhancements. In Proceedings of the 3rd Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, CA, USA, 6–8 June 2007; pp. 84–87.
45. Harabor, D.; Botea, A. Hierarchical Path Planning for Multi-Size Agents in Heterogeneous Environments. In Proceedings of the IEEE Symposium on Computational Intelligence and Games, Perth, Australia, 15–18 December 2008; pp. 258–265.
46. Li, Y.; Su, L.-M.; Li, W.L. Hierarchical Path-Finding Based on Decision Tree. In Proceedings of the 7th International Conference on Rough Sets and Knowledge Technology, Chengdu, China, 17–20 August 2012; pp. 248–256.