Journal of Geography and Earth Sciences
June 2022, Vol. 10, No. 1, pp. 1-15
ISSN 2334-2447 (Print) 2334-2455 (Online)
Copyright © The Author(s). All Rights Reserved.
Published by American Research Institute for Policy Development
DOI: 10.15640/jges.v10n1a1
URL: https://doi.org/10.15640/jges.v10n1a1

A New Approach for Computing Anisotropic Cost Surfaces and Least-Cost Paths

C. Romo, J.C. Torres[1], G. Arroyo and A. León

Abstract

Cost surfaces are a useful tool for solving many spatial analysis problems. However, GIS tools are limited when used to compute cost surfaces. This is primarily because they can only deal with isotropic frictions or, at best, they use a very limited definition of anisotropic friction that only considers local information, thereby generating unnatural paths.

This paper proposes a novel algorithm for computing anisotropic cost surfaces from a cell traversing cost which not only incorporates the direction of movement, but also the change of direction. As such we can solve real-world anisotropic problems, such as the design of roads. The paper also describes the implementation of this approach on GRASS GIS.

1 Introduction

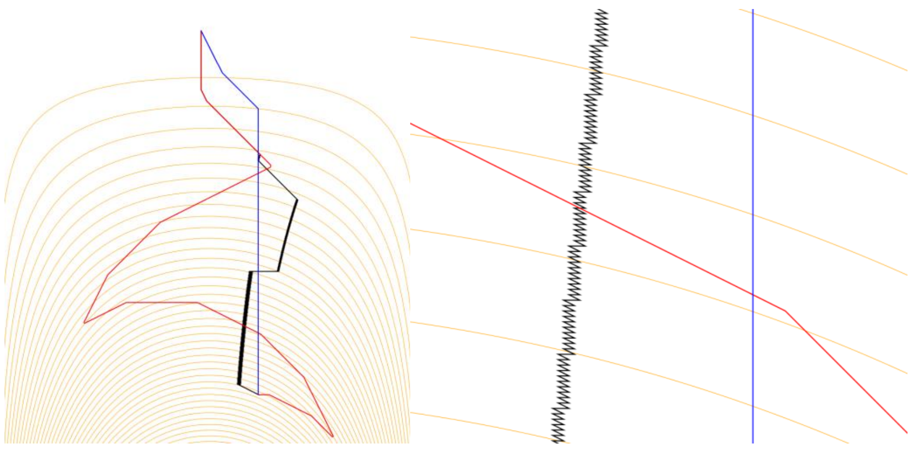
Cost surfaces are widely used to solve spatial analysis problems and compute least-cost paths. A cost surface can be defined as a function that, for a given friction function $f(x)$, assigns the least cost to traverse from an initial region S to any point Q on the working region:

$$AC_S^f(Q) = \min_{\mathrm{path}(S,Q)}[\ \mathrm{cost}(\ \mathrm{path}(S,Q)\ )\ ] = \min \int_S^Q f(x)dx$$

The friction function $f(x)$ indicates the cost of moving at point x. This cost may represent time, distance, economic cost or any other parameter.

Cost surfaces can be computed using Dijkstra's algorithm [1], which takes a graph as input. The edges of this graph are weighted, storing the cost of displacement between two linked nodes. GIS systems usually use a raster map to store frictions. Every cell in the friction map contains the cost of crossing the cell, either in a fixed direction (E-W) or per unit length. The graph derived from the connectivity relationships on the map is used to apply Dijkstra's algorithm.

Therefore, this approach poses two limitations: the discretization of the movement directions and the use of isotropic frictions.



1 Virtual Reality and Computer Graphics Research Group, University of Granada, Spain. Email: jctorres@ugr.es Virtual Reality Lab, University of Granada, Granada, Spain

Figure 1: Comparison of the proposed approach for computing an anisotropic path (red) with the isotropic path (blue) and the previous proposal for anisotropic cost surface algorithms (black). The right-hand image shows a close-up of a section of the path.

Some approach has been used to overcome the limitation of movement directions. Antikainen [2] proposes to use large connectivity pattern between raster cells to improve the quality of the generated paths. Dean [3] proposes to use a Triangulated irregular network to allow unlimited movement directios. Several authors have addressed anisotropic cost problems by proposing very specific computation methods. Dean et al. [4] study the impact of linear hydrological features with high crossing costs on least-cost paths and proposes a method to treat them. Collischonn and Pilar [5] and Yu et al. [6] addressed the design of roads and channels, wherein they proposed an algorithm that computes the path directly from the digital elevation model without representing friction. Saha et al. [7] planned mountain routes using a similar approach to the one proposed by Collischonn [5], but considering more movement directions and adding thematic maps to restrict movement. Cooper [8] modeled mobility in Pre-Columbian Cuba using the *r.walk* function of GRASS GIS, which computes cost surfaces for walking movements [9]. Bevan [10] analyzed different approaches for computing walking distance maps of Crete during the Bronze Age. LeidWanger [11] studied mobility on the oriental Mediterranean Sea to analyze cultural relationships in Ancient Greece. LeidWanger used the ArcGIS Path Distance functions to compute friction from the average wind direction. Nuñez et al. [12] applied cost surfaces to estimate corridors for the movement of species using *r.walk*, and adapted the temperature map using it as an elevation model.

Furthermore, there are problems for which the friction depends not only on the position and movement direction, but also on the change in the direction of movement. Examples of such an instance are a road path, a pipeline or an electric power line. Choi and Nieto [13] and Baek and Choi [14] study the computation of off-road dump truck least-cost paths, using an algorithm derived from the one proposed by Collischonn and Pilar [5]. They compute the friction from the elevation map adding a cost for the change of direction that is a function of the angle between the incoming and the outgoing directions. This approach can be used only when the friction is a linear function of the elevation.

Although one of the best known purposes of cost surfaces is the computation of least cost paths, there are many other problems that have been solved by means of cost surfaces: computing influence areas [15], computing accessibility [16], resource allocations [17], interpolation [18], planning [12], generating routes with minimum pollution exposure [19], among others. To solve these problems it is necessary to compute the cost surface from a friction maps.

None of the methods previously described compute anisotropic cost surfaces from a friction function. Moreover, most of them do not represent friction. For instance some of them derived the friction from an elevation map, thus limiting their application to problems in which friction is a function of elevation.

To solve general anisotropic cost computation problems, we need an anisotropic cost surface generation algorithm that takes an explicit representation of the anisotropic friction as input.

This paper presents a novel and general approach for computing both cost surfaces and least-cost paths for anisotropic problems in which the friction considers the change in the direction of movement. The proposed approach generates natural paths that do not exhibit a macro-scale isotropic behavior (see Figure 1). The main contributions of this paper are:

- A new representation of friction for anisotropic cost computation problems.
- A new algorithm for computing anisotropic cost surfaces.
- A new algorithm for computing least-cost cost paths from anisotropic cost surfaces.

The rest of this paper is structured as follows. Section

2 reviews previous approaches to computing anisotropic cost surfaces. Our approach is presented in Section

3. Section

4 describes the implementation developed for GRASS. Finally, Section 5 presents some case studies and an evaluation of our method.

2 Previous work

IDRISI Software [20] includes the *varcost* function which computes anisotropic cost surfaces. It expresses the anisotropic friction as the maximum friction in each cell (represented as magnitude and direction maps) and a function which describes how friction varies with the direction of movement. This function is the same for all the regions [21].

GRASS includes the *r.walk* function [22], which also computes anisotropic cost surfaces, although it uses an isotropic friction map and an elevation map from which anisotropic cost is computed based on the formula of Langmuir et al. [23].

ArcGIS includes a path distance function [24] that can perform a similar functionality as *r.walk* and *varcost*. Gietl et al. [25] compared these three GIS packages (IDRISI, GRASS and ArcGIS) and presented a comparative case study in a high alpine environment.

To the best of our knowledge, only Valdez et al. [26] and Romo et al. [27] have previously suggested the use of cost surface computation to solve anisotropic problems using friction maps.

Valdez and Dean [26] proposed an anisotropic cost spreading algorithm that used eight unit cost maps as inputs, each representing the friction for one of the eight possible directions of movement from the center of one cell to the center of an adjacent cell.

Romo and Torres [27] used a similar approach to compute anisotropic cost surfaces and least-cost paths with either eight or sixteen unit cost maps, applied them to road design and model dispersion processes. Nevertheless, these approaches evaluated anisotropy at cell level, so they may produce paths consisting of a sequence of zig-zags, as shown in Figure 1. The segments of these paths are usually only one cell long, and therefore, the resulting path exhibits a macro-scale trajectory similar to that of an isotropic path, as shown in Figure 1.

One of the main differences between the above approaches for solving anisotropic cost surfaces is how they represent friction. Friction for isotropic problems is easily represented as a cell traversing cost, as explained in Section

1. However, for anisotropic problems, cost must be associated with the direction of movement. This has been done in different ways:

(1) Assigning every cell a minimum crossing cost and direction, and assuming a fixed distribution of cost for other directions.
(2) Deriving the friction as a function of a digital elevation model, computing the friction using a predefined formula applied to the elevation values.
(3) Assigning every cell a cost of movement to each adjacent cell, which implies using a directional friction map for every possible direction of movement. Note that there is no need to use more cost maps than permitted connections to adjacent cells.

The IDRISI *varcost* function uses the first approach. GRASS *r.walk* function, Collischon et al. [5] and Yu et al. [6] used the second one, while Valdez et al. [26] and Romo et al. [27] used the third approach. The third is more general than the other two. Nevertheless, it may produce paths that are isotropic at a macro scale. This paper presents a new cost surface computation method that takes the third approach and adds a cost for changing the direction of movement in order to represent the friction.

3 Proposed approach

The shortest path problem with turning costs has been well studied on vector maps ([28] and [29]), but it has not been addressed on raster representations.

We propose to represent the friction using a function that depends on the position, the direction of movement and the change in the direction of movement. Let us represent a path in parametric form as $P(t)$, where the parameter t traverses the path from the starting point (t=0). Thus position Q on the path can be identified by the parameter t_Q, a value for which $P(t_Q)=Q$. The friction can then be expressed as:

$$Friction(\mathbf{P}(t)) = f(\mathbf{P}(t), \vec{s}(t), \frac{d\vec{s}(t)}{dt})$$

where $\mathbf{P}(t)$ is the position at point t and $\vec{s}(t)$ is the direction of movement, that is, the direction of vector $\frac{dP(t)}{dt}$, so

$$\vec{s}(t) = \frac{\frac{dP(t)}{dt}}{\|\frac{dP(t)}{dt}\|}$$

The variable *t* can be understood, without loss of generality, as time if the path is considered as the trajectory of a moving object. The derivate $\frac{d\vec{s}(t)}{dt}$ is the change in direction of movement along the path, which is related to the path curvature.

We separated the friction into two components:

$$Friction(\mathbf{P}(t)) = F_a(\mathbf{P}(t), \vec{s}(t)) + F_t(\frac{d\vec{s}(t)}{dt})$$

The first component, the anisotropic friction F_a, encapsulates the dependency on the position and the direction, while the second, F_t, is the contribution of the change in direction of movement, which we shall call the turning friction. In this way, we can use different representations for both components.

While the number of possible directions of movement is limited (usually eight or sixteen), the first component, , can be represented as a set of raster maps, each storing the value of the friction for a given direction, which follows the approach proposed by Valdez [26] and Romo [27]. Each cell on these maps stores the cost of moving to the current cell following a fixed direction of movement.

The turning friction can be global for the whole region or depend on the position. In the former case, the second component of the friction, F_t, is independent of the position, and so it can be represented as a function of the angle (see Figure 2). We have decided to use a global quadratic function of the angle, as it is simple and allows us to solve the most common problems without increasing the complexity of the system.

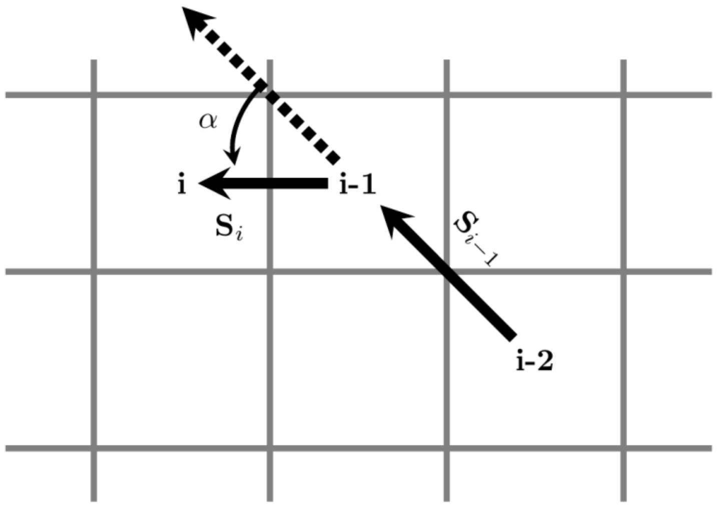


Figure 2: Turning friction is computed as a function of the incoming direction to the current cell, S_i, andthe incoming direction to the previous one, S_{i-1}.

To compute the cost of a path, Equation 1 must be integrated along the path. This integral is computed as the sum of the friction for every cell in the path. The contribution of the i-th cell of the path is

$$cost_i = F_a(i, s_i) + F_t(s_i, s_{i-1}) \tag{5}$$

where $F_a(i,S_i)$ is the cost of moving from the previous cell, $i-1$, to the actual cell, i, which is a function of both the position (i) and direction of movement (S_i). $F_t(S_i,S_{i-1})$ is the turning friction for the last movement, in other words it is a function of the angle between S_i and S_{i-1},where S_i is the incoming direction to the current cell and S_{i-1} is the incoming direction to the previous one (see Figure 2). Note that S_{i-1} is the input direction at cell $i-1$ and S_i is the output direction at cell $i-1$.

To develop a flexible and compact specification of the turning friction, F_t, we used two parameters $k_1$and k_2 to describe it as follows:

$$cost(\alpha) = \begin{cases} 0 \; if \; \alpha = 0 \\ k_1 \; if \; \alpha = 22.5 \\ k_2 \cdot k_1 \; if \; \alpha = 45 \\ k_2 \cdot (cost(\alpha - 22.5) + cost(\alpha - 45)) \; if \; \alpha > 45 \end{cases}$$

where α is the angle between S_i and S_{i-1}, $k_1$the cost added by turning an angle of 22.5º, and k_2 is the ratio at which the cost increases when the angle is increased. The value of 1 for k_2 indicates linear growth. Note that the algorithms are independent of the representation of the turning friction function.

4 Implementation

To compute the inertial anisotropic cost surface we have adapted Dijkstra's algorithm provided by the *r.cost* GRASS GIS function. As explained in Section

3, we used two components to represent the friction. The first corresponds to the non-inertial component, which is stored using one directional friction map per permissible movement direction. Each cell in a given directional friction map contains the cost of moving from said cell following the direction specified by the map, instead of the cost of crossing it.

To simplify the nomenclature for these maps, they are named by sharing the same root and adding a suffix that indicates the direction of movement.

The second component is the inertial friction due to the change of direction of movement, which depends on the angle between the incoming and outgoing directions (α in Figure 2).

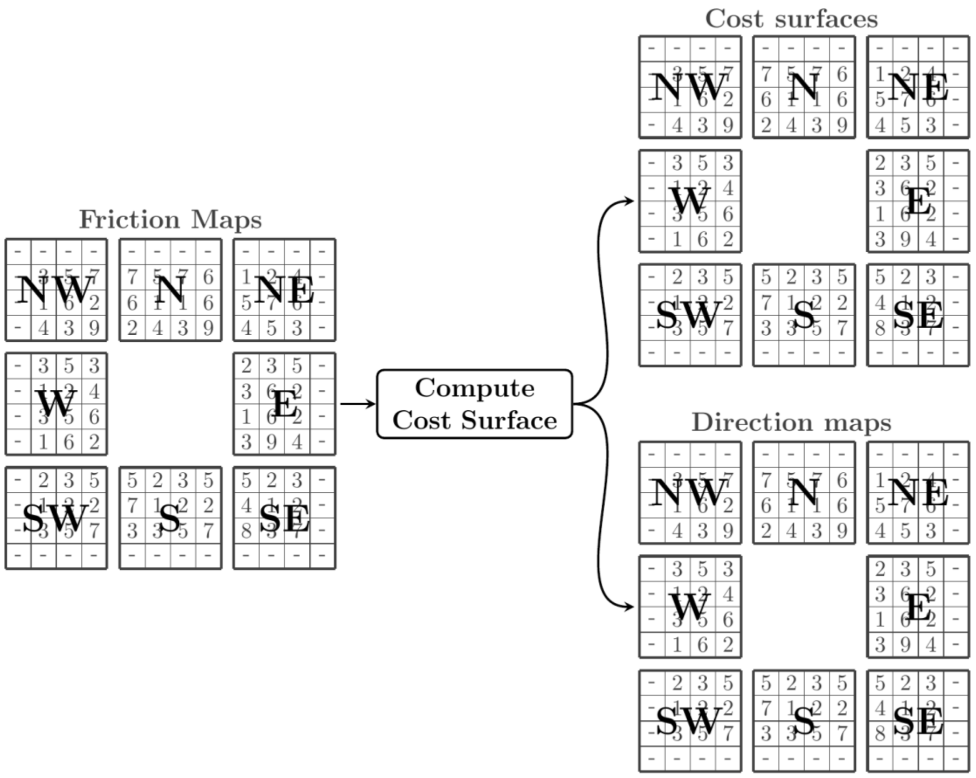


Figure 3: Cost surface computation using the proposed algorithm takes directional friction maps as input (one for every possible movement direction, eight in this example) and generates one cost surface map for every possible incoming direction. Incoming direction output maps store the incoming direction when entering the previous cell.

When using anisotropic frictions, we cannot guarantee that a cell is reached from its neighbor with the lowest cost, so we must generate a map that stores the incoming directions. Moreover, we need to know the incoming direction entering the previous cell to compute the turning cost. Our implementation computes a set of incoming direction maps, *direction map*$_{si}$, one for every possible direction of movement, s_i. Each map stores the incoming direction entering the previous cell when the current cell is reached from the direction indicated by the map.

The proposed algorithm takes as input the initial cells, eight directional friction maps (or sixteen when knight moves are considered), and the coefficients defining the inertial friction as a function of the change of direction of movement The output of the algorithm is a set of eight or sixteen cost surface maps (one for each incoming direction, called *accumulated_map* in the algorithm) and a set of maps, named *direction_maps*. The *direction_maps* contain the incoming direction entering the previous cell for each incoming direction (see Figure 3). The pseudo-code of the cost surface computation is shown in Algorithm 1.

Algorithm 1 r.acost(initial_cells,cost_map[#neighbours],k_1,k_2)

```
1: $heap \leftarrow initialize(initial\_cells)$
2: $accumulated\_map[0,\dots,15] \leftarrow NULL$
3: $directions\_map[0\dots,15] \leftarrow NULL$
4: for each cell in the heap do
5:   $(row, col, min\_cost, in\_dir) \leftarrow get\_cell(heap)$
6:   remove cell from the heap
7:   $current\_cost \leftarrow accumulated\_map[in\_dir][row, col]$
8:   if $min\_cost < current\_cost$ or $current\_cost = NULL$ then
9:     for each adjacent cell do
10:      $nrow \leftarrow$ adjacent cell row
11:      $ncol \leftarrow$ adjacent cell column
12:      $out\_dir \leftarrow$ direction from cell to adjacent cell
13:      $cost \leftarrow cost\_map[out\_dir][row, col]$
14:      $turning\_cost \leftarrow turningCost[angle(out\_dir, in\_dir)]$
15:      $total\_cost \leftarrow min\_cost + cost + turning\_cost$
16:      $current\_cost \leftarrow accumulated\_map[out\_dir][row, col]$
17:      if $total\_cost < current\_cost$ then
18:        $insert\_heap \leftarrow (nrow, ncol, total\_cost, out\_dir)$
19:        $accumulated\_map[out\_dir][nrow, ncol] \leftarrow total\_cost$
20:        $directions\_map[out\_dir][row, col] \leftarrow in\_dir$
21:      end if
22:    end for
23:  end if
24: end for
```

The algorithm uses a heap of cells with a tentative value for their cost, sorted by cost value. For each cell, the heap contains its row and column, its provisional cost and the direction from which the cell was reached.

This heap is initialized with the initial cells (line 1), inserting the row and column, the initial cost of the starting cells and null as previous directions in the heap. We can set the starting direction by inserting only the cell after the initial cell following the starting direction in the heap.

The variable *accumulated_map* is a set of eight or sixteen cost surface maps that will contain the output cost surface for the different incoming directions (line 2). The variable *direction_maps* is the set of maps containing the incoming direction entering the previous cell. The cells of these maps are initialized to null (lines 2 and 3).

The proposed algorithm follows Dijkstra's scheme and iterates taking the cell with the lowest cost off the heap (lines 4 to 7) until the heap is empty. The variable *current_cost* is assigned the previously computed cost of the extracted cell. The function *get_cell(heap)* take off the cell on top of the heap.

The cost obtained from the heap for each extracted cell, *min_cost*, is compared with the current cost on the *accumulated_map* corresponding to the direction of movement (line 8). This is necessary because a lower cost could have has been computed for the cell after this element was inserted in the heap. In this case, the newly extracted cost must be ignored. Otherwise, every neighbor cell is visited (line 9) and the cost of traveling to it from the newly extracted cell is computed. This cost is computed by adding the traversing cost between the two cells according to their relative direction, which is obtained from the input friction map *cost_map[out_dir][row, col]* (line 13) and the cost associated with the change of direction of movement between the directions *out_dir* and *in_dir* (line 14). Here *cost_map[out_dir][row, col]* denotes the cell *[row,col]* of the cost surface for the out-going direction *out_dir* and *turningCost* is the function for computing the friction due to the change of direction of movement (6). The function *turnigCost[angle(out_dir, in_dir)]* computes the turning cost due to the change of direction of movement from the incoming direction, *in_dir*, to the out-going one, *out_dir*. If the cost computed, *total_cost*, is smaller than the cost previously computed for the neighbor cell, current_cost, then the cost is assigned to this neighbor cell and it is inserted in the heap (lines 17 to 20).

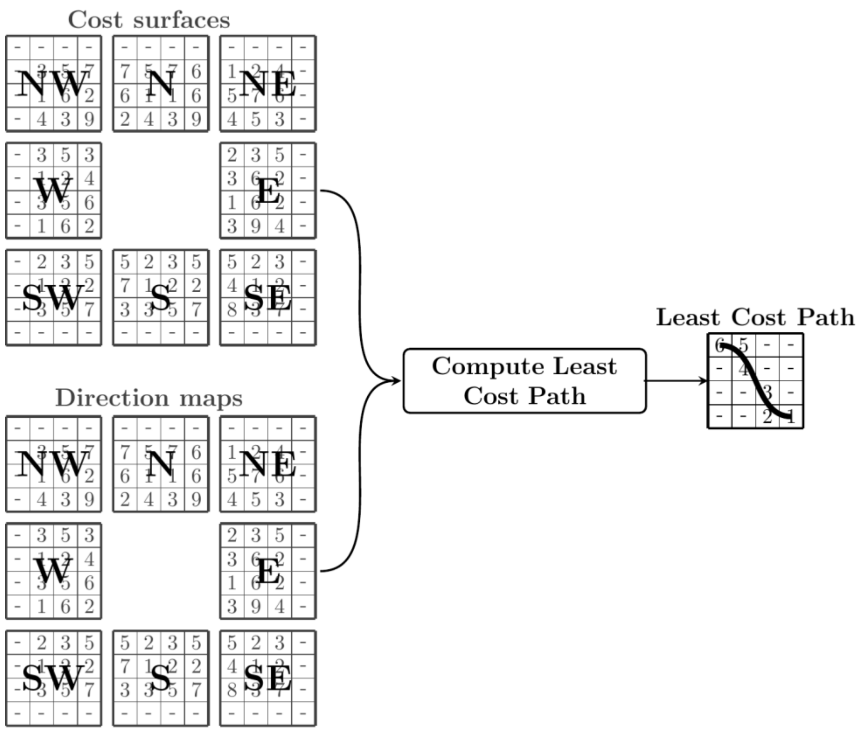


Figure 4: Scheme of the least cost path computation algorithm.

The least cost path computation algorithm takes as an input the destination cell, the set of cost surface maps and the set of incoming directions maps generated by the cost surface algorithm and optionally the incoming direction to the destination cell. The algorithm follows back the link stored at the incoming direction maps (see Figure 4). The pseudo-code for the least cost path algorithm is shown in Algorithm 2.

Algorithm 2 r.adrain(destination_cell, accumulates_map, directions_map, [incoming_direction])

```
1: $(row, col) \leftarrow get(destination\_cell)$
2: if $incoming\_direction$ is given then
3:     $previous\_dir \leftarrow incoming\_direction$
4: else
5:     $mincost \leftarrow accumulated\_map[0][row, col]$
6:     $previous\_dir \leftarrow 0$
7:     for each possible $direction$ do
8:         if $mincost > accumulated\_map[direction][row, col]$ then
9:             $mincost \leftarrow accumulated\_map[direction][row, col]$
10:            $previous\_dir \leftarrow direction$
11:        end if
12:    end for
13: end if
14: while $previous\_dir \neq NULL$ do
15:     $(prev\_row, prev\_col) \leftarrow$ neighbour of $(row, vol)$ at $previus\_direction$
16:     $direction \leftarrow directions\_map[previous\_dir][next\_row, next\_col]$
17:     add line from $(row, col)$ to $(next\_row, next\_col)$ to the output map
18:     $(row, col) \leftarrow (prev\_row, prev\_col)$
19:     $previous\_dir \leftarrow direction$
20: end while
```

The algorithm starts by reading the destination cell and the incoming direction (lines 1 to 3).
If the incoming direction is not provided, it selects the direction with the lowest cost (lines 4 to 12).

Once the incoming direction for the destination cell has been assigned, we returned back along the path until we reach the starting cell (line 14). At each step, we computed the coordinates to the previous cell according to the direction vector, *previous_direction* (line 15). This computation is straightforward. For instance, if the direction of the previous movement (from the cost surface origin) is North, then the next cell (from the path destination) is to the South.

Subsequently, we updated the direction to the one from which we arrived at the previous cell, referring to the corresponding direction map (line 16). We then added a segment to the output path map (line 17).

Finally, the variables were adjusted for the next iteration (line 18 and 19), the algorithm iterates until it reaches a cell for which the *previous_dir* value is null.

The current application accepts several starting points. In this case the algorithm is computed inside a loop for each of them.

It is easy to prove that the computational complexity of the cost surface algorithm is the same as the isotropic one because the loop structure is the same. For the current implementation, using a heap, the complexity is *O(nlog(n))*, being *n* the number of cells. The complexity of the least-cost path computation algorithm is obviously linear, as it depends on the length of the path and can be bounded by *O(n)*, being *n* the number of cells in the path.

5 Results

We performed several tests to assess the quality of the proposed method. The new approach was compared with previous methods (the anisotropic cost surface method proposed by Valdez [26] and Romo [27] and the least cost road computation method proposed by Collischonn [5]. Tests were performed on an Intel® Core i7 personal computer with 32GB of RAM, running GRASS 7.3 on Ubuntu Linux 16.04. This section describes some of these tests.

To allow reproducibility, all the details about the generation of the data sets and running the tests are included in Appendix A.

Although the proposed method generated both cost surfaces and least-cost paths the figures in this section show the generated paths, as it is easier to identify the behavior of the method on single paths than on the complete cost surface.

5.1 Alpujarra mountain road

This test compared roads designed using our algorithm against real mountain roads and those generated using anisotropic friction without an inertial component. To do so, we used real data from an abrupt area in southern Spain. We choose this area because there is a sharp mountain road with which we can compare the results.

The altitude in the area ranges from 508 to 1027 meters. The size of the generated DEM map is 3420 x 4740 cells. Figure 5 shows a 3D view of this model (see Appendix for details).

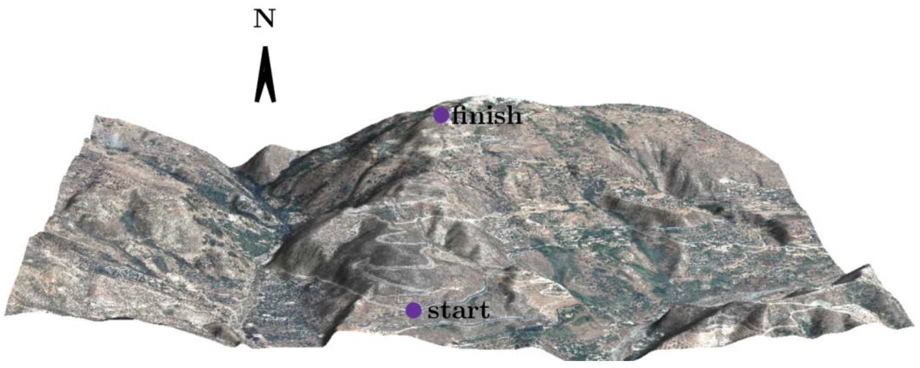


Figure 5: 3D view of the Alpujarra test case.

First, the anisotropic friction maps were generated with raster algebra, using the following expression to compute the cost to move from cell *i* to cell *j*:

$$F_a(i,j) = \begin{cases} null\ if\ slope(i,j) > maxSlope \\ \alpha \cdot slope(i,j)^2 \cdot distance(i,j) + \beta \cdot distance(i,j)\ otherwise \end{cases}$$

where *slope*(*i*,*j*) is the slope of the line from cell *i* to the neighboring cell *j* and *distance*(*i*,*j*) is the distance from cell *i* to cell *j*. Parameter α controls the contribution of the slope to the cost and parameter β the contribution of the distance. In this case study, the following values were used: $a=100$, $\beta=1$ and $maxSlope=0.2$. Note that this cost is computed as a set of maps, one for every possible movement direction.

The anisotropic cost was computed using the coordinates 461621,4085201 as a starting point, which correspond to the road crossing at the bottom of Figure 5. Values 3 and 4 were used for the inertial coefficients *k_1*and *k_2*, respectively.

The shortest path was computed using the anisotropic drain function for the destination point 461755,4086564 and an easterly incoming direction, which corresponds to a high slope trajectory (see Figure 6).

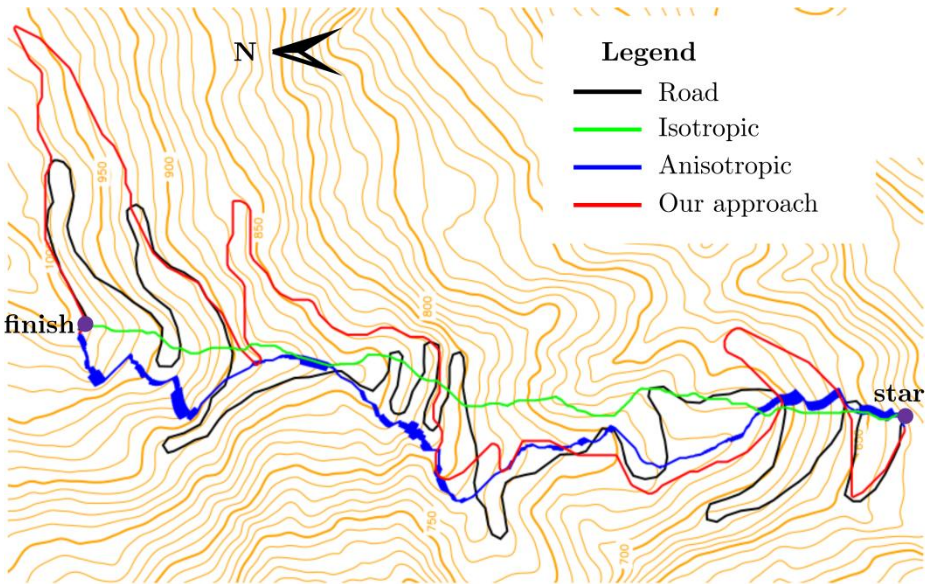


Figure 6: Result for the mountain road case. The real road is drawn in black, the anisotropic path is shown in blue, the isotropic cost in green and the inertial anisotropic path in red.

To make comparisons with the non-inertial anisotropic cost proposed by Valdez [26] and Romo [27], a new cost surface was computed setting the turning friction to zero. The resulting path is shown as a blue line in Figures 6, 7 and 8.

The isotropic path was computed using the *r.cost and* r.drain GRASS commands, using the average of the anisotropic frictions as isotropic friction. Results are shown in Figures 6, 7 and 8 in green. Current roads are shown in black lines while our inertial anisotropic path is in red.

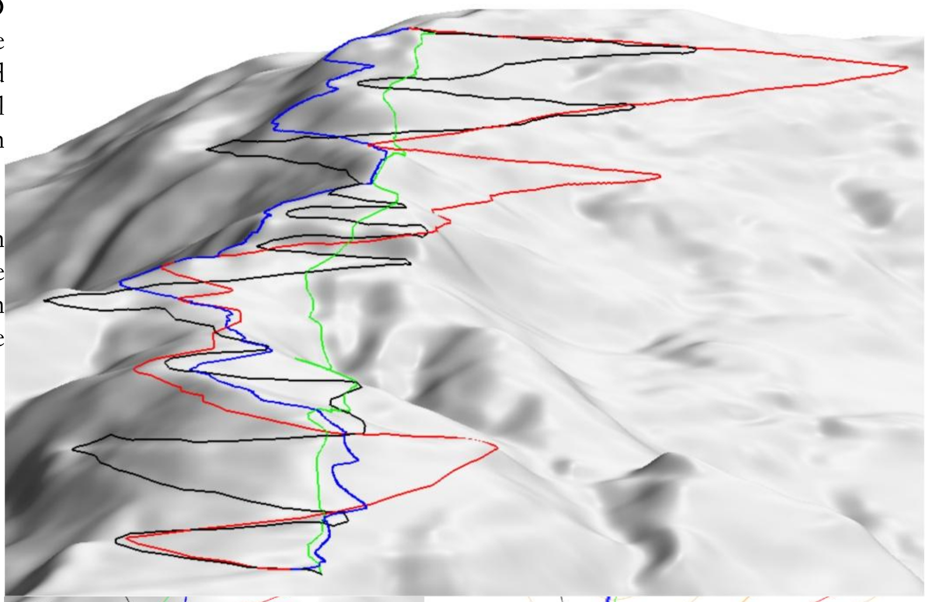


Figure 7: 3D view of the mountain road case. The real road is drawn in black,the anisotropic path is shown in blue, the isotropic cost in green and the inertialanisotropic path in red.

Figure 8: Details of the mountain road case. The real road is drawn in black, the isotropic cost in green, the anisotropic path is shown in blue and the inertial anisotropic path in red.

We can see that the inertial path (red line) climbs following a trajectory with a uniform slope and few bends. It has 10 bends, while the real road has 19. In contrast, the non-inertial anisotropic path produces a very large number of bends. In fact, the wide segments of the blue line in Figure 6 are a narrow zigzag, which are more easily observed in the detailed image in Figure 8. In this path, it is possible to distinguish a micro-scale trajectory (which takes into account the anisotropy) and an isotropic macro-scale trajectory. The macro-scale trajectory, which can be understood as the smoothed or generalized trajectory, does not consider the anisotropic nature of the problem. In fact, the smoothed trajectory of the anisotropic path is similar to the isotropic path; both climb crossing perpendicular to the elevation contours.

5.2 Conical DEM

This test was designed to compare our approach with the least cost road computation method proposed by Collischonn et al. [5]. We created a synthetic conical digital elevation model like the one used by Collischonn et al. [5]. The DEM was created using map algebra (see Appendix A.2 for details). Anisotropic directional friction was computed by applying Equation 7 using values of α=1000 and β=1.

The high value of α in this case produced a high value of friction due to slope. Using these values it would be preferable to move twice the distance across a flat area than to move up a path with a 1.1% slope.

The inertial component was computed by setting the coefficients $k_1$to 30 and $k_2$to 2. These values introduce an additional cost of 30 (three times the cost of crossing a flat cell) for turning 22.5°. This cost increases to 120 for a turn of 45° and to 270 for a turn of 67.5°. Therefore turning 22.5° four consecutive times has the same cost as turning 45° just once.

The actual values of parameters k_1 and k_2 are not relevant when testing the algorithm. They are included here to facilitate reproduction and to demonstrate their physical meaning with regards to the actual problem.

The results are shown in red on Figure 9, in comparison with the anisotropic (in blue) and the isotropic paths (in green).

We can see that the inertial path climbs following a trajectory with a constant slope and curvature.

The isotropic cost path climbs following a straight line. The anisotropic path again exhibits a dual behavior. At a micro-scale level, it considers the anisotropy of the friction, but at a macro-scale it climbs following a straight path as the isotropic path. The path computed by Collischonn et al. [5] climbs smoothly but features sharp bends.

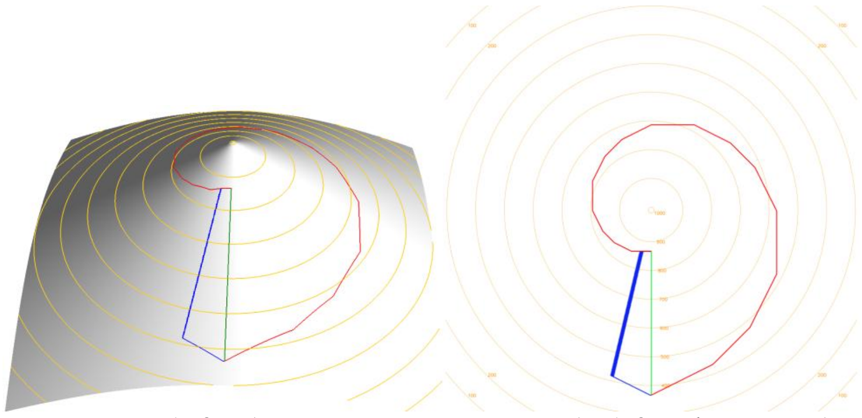


Figure 9: Result for the cone case. 3D view on the left and map on the right. The isotropic path is shown in green, the anisotropic path in blue and the inertial anisotropic path in red.

5.3 Stelvio pass

These tests aimed to show the influence of the different parameters on the resulting path and to perform a quantitative evaluation of the paths generated by the proposed algorithm. We used real elevation data of a mountain pass in northern Italy. The working region was 11 x 17 km with a 1 m resolution, and the elevation ranged from 1200 to 3800 m (see Appendix A.3 for details).

As the goal of this test is to show the value of the parameters used to compute the turning friction we run our algorithm with different values of $k_1$and k_2. Figure 10 shows some of the paths computed based on this model and Figure 11 shows one section in detail. The two figures reflect the behavior previously described for the isotropic and anisotropic paths and that the number of bends in the different inertial paths depends on the cost used for the turning friction coefficients.

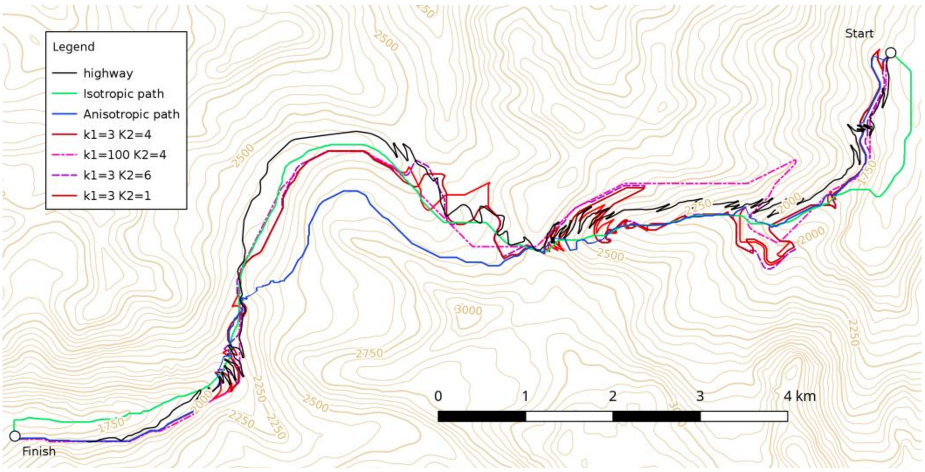


Figure 10: Map of the Stelvio pass comparing the highway (in black), the isotropic path (in green), the anisotropic one (in blue) with several paths generated by the proposed method using different values for the parameters for the turning friction.

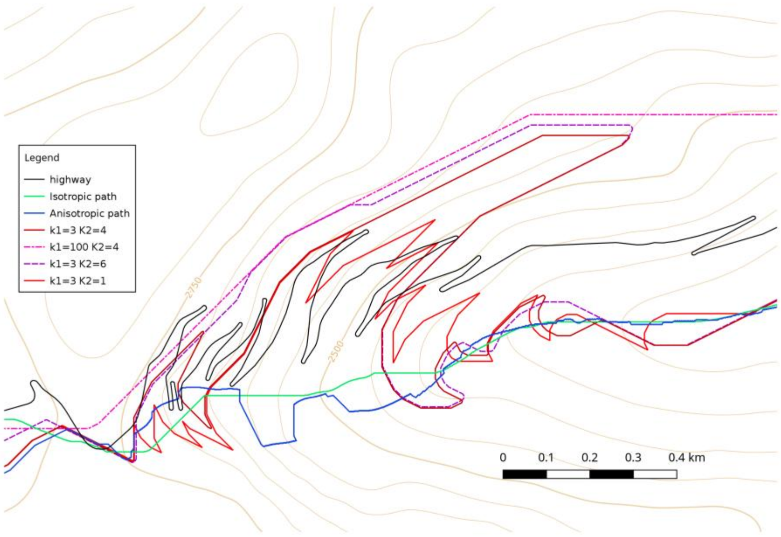


Figure 11: Detailed view of the central section of the Stelvio pass case study.

To perform a numerical evaluation of the paths we computed the length, the average slope and the sum of absolute angles for all the bends. The average slope was computed as the average absolute value of slopes, using the following expression:

$$slope = \frac{\sum_i |S_i| L_i}{L} \quad (8$$

Where $S_i$is the slope of segment i and L_i is the length of the segment. Appendix A.3 gives a detailed explanation of how these parameters were obtained from the vector maps representing the paths.

The paths generated from the proposed algorithms using different values for the turning cost were compared with the isotropic and anisotropic paths and with the highway. The numerical results are shown in Table 1. Two different values for the slope of the highway are reported: the real road slope (7.3) and the one computed using the interpolated high-resolution elevation model (16.7).

Path	Length (m)	# Turns	Slope
Highway	26740	54.8	7.3 (16.7)
Isotropic	15352	563.6	13.36
Anisotropic	22843	2125.4	9.60
$k_1 = 3$, $k_2 = 1$	21308	28.8	10.44
$k_1 = 3$, $k_2 = 4$	20574	20.2	10.82
$k_1 = 3$, $k_2 = 6$	18936	13.1	11.81
$k_1 = 100$, $k_2 = 4$	15911	3.0	14.41

Table 1: Numerical comparative of different paths.

We plotted the values in Table 1 using a three axis representation to facilitate the visual comparison (see Figure 12). Note that the turns axis is using a logarithmic scale. The data show the high value for the sum of absolute angles for the bends in the anisotropic path and how this value can be controlled using the proposed algorithm. It can be seen that changing the turning friction is is possible to obtains paths ranging from 2.5 turns (for *k_1=100* and *k_2=4*) to 2500 (for *k_1=k_2=0* that correspond to the anisotropic path). Changing the turning cost influence the slope of the path because the relative contribution to the friction of the slope is increased when the turning friction is decreased.

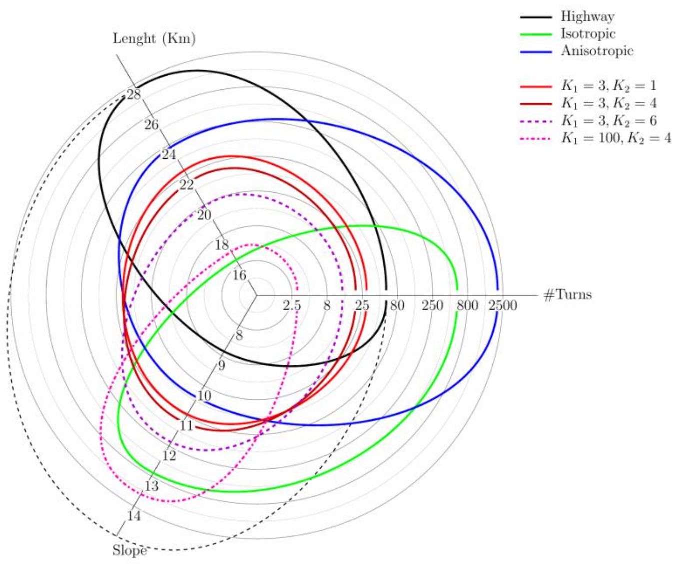


Figure 12: Graphical comparison of the different paths for the Setelvio pass case study.

6 Conclusions

Cost surfaces are a powerful GIS tool and can be used to solve many different problems. They hold, for each cell, the integration of a friction function along the least-cost path from a given starting area. Isotropic friction is usually represented as a friction map. However, this representation cannot be used to solve anisotropic problems, as the friction depends on the direction of movement in such cases. A small number of algorithms have been developed for computing anisotropic cost surfaces. These algorithms are very limited as they impose strong restrictions on the friction function. The most general method to compute anisotropic cost surfaces is the one proposed by Valdez et al. [26] and Romo et al. [27], but even these do not take into account the contribution of the inertia to the friction, that is the cost of changing the direction of movement. Neglecting inertia may produce paths featuring a dense sequence of short zigzag segments, which are quite unnatural. We have shown that anisotropic paths generated using these approaches have a macro-scale trajectory that is similar to that obtained using isotropic cost surface, which is probably why they have not become widespread.

In the present paper, we have proposed a new algorithm for computing anisotropic cost surfaces that considers real anisotropic friction functions, including inertia. The proposed method adds a component to the friction due to inertia that depends on the change of direction based on a quadratic equation. Our method generates smooth paths that do not exhibit a micro scale isotropic behavior.

The proposed algorithm generalizes the one put forward by Valdez et al. [26] and Romo et al. [27], because the result of their algorithms can be obtained with the one proposed here when there is no inertial friction ($k_1=k_2=0$).

Our algorithm was tested by comparing it against previous approaches and real mountain roads. Results show that our method is flexible and generates paths that takes incorporate the anisotropy of the friction while minimizing changes in direction. We have also shown that real problems can be modeled flexibly while using meaningful parameters.

References

[1] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[2] Harri Antikainen. Comparison of Different Strategies for Determining Raster-Based Least-Cost Paths with a Minimum Amount of Distortion. *Transactions in GIS*, 17(1):96–108, 2013.

[3] Denis J Dean. Optimal routefinding with unlimited possible directions of movement. *Transactions in GIS*, 15(1):87–107, 2011.

[4] Denis J Dean, Vaishnavi Thakar, and Neeraj Sirdeshmukh. Optimal Routefinding Across Landscapes Featuring High-cost Linear Obstacles. *Transactions in GIS*, 20(4):613–625, 2016.

[5] Walter Collischonn and Jorge Victor Pilar. A direction dependent least-cost-path algorithm for roads and canals. *International Journal of Geographical Information Science*, 14(4):397–406, 2000.

[6] Chaoqing Yu, JAY Lee, and Mandy J Munro-Stasiuk. Extensions to least-cost path algorithms for roadway planning. *International Journal of Geographical Information Science*, 17(4):361–376, 2003.

[7] Ashis Kumar Saha, Manoj K Arora, Ravi Prakash Gupta, ML Virdi, and Elmar Csaplovics. GIS-based route planning in landslide-prone areas. *International Journal of Geographical Information Science*, 19(10):1149–1175, 2005.

[8] Jago Cooper. Modelling mobility and exchange in pre-columbian cuba: gis led approaches to identifying pathways and reconstructing journeys from the archaeological record. *Journal of Caribbean Archaeology*, pages 122–137, 2010.

[9] Markus Neteler and Helena Mitasova. *Open source GIS: a GRASS GIS approach*. Springer Science & Business Media, 2002.

[10] AH Bevan. Computational models for understanding movement and territory. In *Tecnologías de información geográfica y análisis arqueológico del territorio:Actas del V Simposio Internacional de Arqueología de Mérida*, pages 383–394. Consejo Superior de Investigaciones Científicas, 2011.

[11] Justin Leidwanger. Modeling distance with time in ancient Mediterranean seafaring: a GIS application for the interpretation of maritime connectivity. *Journal of Archaeological Science*, 40(8):3302–3308, 2013.

[12] Tristan A Nuñez, Joshua J Lawler, Brad H Mcrae, DJOHN PIERCE, Meade B Krosby, Darren M Kavanagh, Peter H Singleton, and Joshua J Tewksbury. Connectivity planning to address climate change. *Conservation Biology*, 27(2):407–416, 2013.

[13]Yosoon Choi and Antonio Nieto. Optimal haulage routing of off-road dump trucks in construction and mining sites using Google Earth and a modified least-cost path algorithm. *Automation in Construction*, 20(7):982–997, 2011.

[14] Jieun Baek and Yosoon Choi. A new algorithm to find raster-based least-cost paths using cut and fill operations. *International Journal of Geographical Information Science*, 31(11):2234–2254, 2017.

[15] Steeve Ebener, Zine El Morjani, Nicolas Ray, and Michael Black. Physical accessibility to health care: from isotropy to anisotropy. *GIS@ development*, 9(6), 2005.

[16] Rui Pedro Juliao et al. Accessibility and GIS. In *ERSA Conference papers, European Regional Science Association*, pages 1–11, 1999.

[17] Timothy S Hare. Using measures of cost distance in the estimation of polity boundaries in the Postclassic Yautepec Valley, Mexico. *Journal of Archaeological Science*, 31(6):799–814, 2004.

[18] Jonathan A Greenberg, Carlos Rueda, Erin L Hestir, Maria J Santos, and Susan L Ustin. Least cost distance analysis for spatial interpolation. *Computers & Geosciences*, 37(2):272–276, 2011.

[19] Gemma Davies and Duncan Whyatt. A least-cost approach to personal exposure reduction. *Transactions in GIS*, 13(2):229–246, 2009.

[20] J. Ronald Eastman. IDRISI GIS, 1987.

[21] J Ronald Eastman. Guide to GIS and Image Processing Volume. *Clark University. USA*, 2001.

[22] GRASS Development Team. r.walk, GRASS 6.4 Users Manual, 2012.

[23] Eric Langmuir, Scottish Sports Council, and Mountain Leader Training Board. *Mountaincraft and leadership: a handbook for mountaineers and hillwalking leaders in the British Isles*. Scottish Sports Council, 1984.

[24] Inc. Environmental Systems Research Institute. Arc GIS 9.2 Desktop Help, 2015.

[25] Rupert Gietl, Michael Doneus , and Martin Fera. Cost Distance Analysis in an Alpine Environment: Comparison of Different Cost Surface Modules. In *Layers of Perception. Proceedings of the 35th International Conference on Computer Applications and Quantitative Methods in Archaeology (CAA)*, page 342–250, 2007.

[26] Jose Valdez and Denis J Dean. An Efficient Algorithm For Reconstructing Anisotropic Spread Cost Surfaces After Minimal Change To Unit Cost Structures. In *3rd Southern Forestry GIS Conference*. Georgia Center for Continuing Education, October 2000.
[27] Celia Romo and Juan Carlos Torres. Superficies de coste anisotrópicas. In *V Jornadas SIG libre*, Girona, 2011. Universitat de Girona. Servei de Sistemes d'Informació Geogràfica i Teledetecció.
[28] Robert Geisberger and Christian Vetter. Efficient routing in road networks with turn costs. In *International Symposium on Experimental Algorithms*, pages 100–111. Springer, 2011.
[29] Stephan Winter. Modeling Costs of Turns in Route Planning. *GeoInformatica*, 6(4):345–361, 2002.

A Appendix

A.1 Alpujarra mountain road

The data used correspond to region N:4086790, S: 4085080, E: 462920, W: 460550 of UTM zone 30. The elevation model and the orthophoto were obtained from the *LINEAv2* public service of the Andalusian regional government (*http://www.juntadeandalucia.es/institutodeestadisticaycartografia/lineav2/.*)

The orthophotos have a base resolution of 0.5 m, while the digital elevation model has a resolution of 10 m. The digital elevation model was interpolated to generate a 0.5 m DEM to match orthophoto resolution, using the *r.resamp.bspline* command:

```
r.resamp.bspline input="dem10" output="H" method="cubic" lambda=0.01
```

The real road map in the region is also used for comparison. It has been downloaded using the *IDEAndalucia* service (*http://www.ideandalucia.es/clientedescarga/*).

The anisotropic friction maps were computed using map algebra. For example, the computation for the south-easterly direction used the following r.mapcalc expression:

```
r.mapcalc "Fa_SE= if(abs(H−H[1,1])/(0.707*(nsres()+ewres()))>0.2,null(),\
100*(H−H[1,1])*(H−H[1,1])/(0.707*(nsres()+ewres()))+(0.707*(nsres()+ewres())))"
```

The GRASS command used to compute the anisotropic cost was:

```
r.acost −k input=fd k1=3 k2=4 output=sci outdir=lci start_coordinates=461621,4085201
```

The shortest path was computed using the anisotropic drain function:

```
r.adrain −k input=sci indir=lci output=ci vector_output=ci start_direction=E \
start_coordinates=461781,4086552
```

To compare with a non-inertial cost, a cost surface was computed setting the turning friction to zero:

```
r.acost −k input=fd k1=0 k2=1 output=scni outdir=lcni start_coordinates=461621,4085201
```

A.2 Conical DEM

The DEM was created using map algebra:

```
dem=1010−(sqrt((x()−5000)*2+(y()−5000)*2))/7
```

in the region [0,10000]×[0,10000] with a resolution of 10 m.

The GRASS command used to compute the anisotropic inertial cost was:

```
r.acost −k input=fda k1=30 k2=2 output=si outdir=li start_coordinates=5000,500
```

The shortest path generated for a starting point at (5000,4000), with west as the incoming direction, was computed using

```
r.adrain −k input=si indir=li output=ci vector_output=ci start_direction=W \ start_coordinates=5000,4000
```

We generated the perturbed DEM using map algebra:

```
r.adrain −k input=si indir=li output=ci vector_output=ci start_direction=W \start_coordinates=5000,4000
```

A.3 Passo dello Stelvio

The elevation data was imported from *https://www.eea.europa.eu/data-and-maps/data/eu-dem* from which an elevation map with a resolution of 25x25 m can be downloaded. The map was restricted to N: 2607625, S: 2596325, E: 4364575, W: 4347100, and interpolated to generate a 1 m resolution map using BSpline resampling:

```
r.resamp.bspline input="EUdem@PERMANENT" output="H" method="bicubic"
```

The road map was obtained from Open Street View.

The anisotropic friction maps were computed as in the case of the Alpujarra mountain road test.

The resulting paths were exported to text files in order for analysis. This conversion was carried out using the following GRASS script that takes the vector map of a path as argument:

```
v.to.points $1 output=p$1 use=vertex −t −−o
v.db.addtable p$1 layer=2 columns="h double precision"
v.what.rast map=p$1 rast=H col=h layer=2
v.out.ascii −−overwrite p$1 output=p$1.txt format=point layer=2 columns=h
```

The resulting text files were processed using a spreadsheet, computing the total length of the path, the sum of the absolute values of the turning angles and the sum of the absolute values of the slopes.