

Centre for Geo-Information

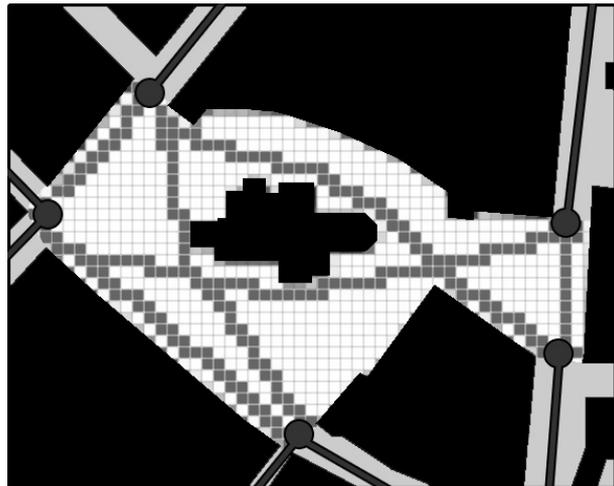
Thesis Report GIRS-2010-13

PEDESTRIAN ROUTE PLANNING IN A HYBRID DATA ENVIRONMENT

Calculating optimal routes based on vector and raster data

Ing. J.H. Bakermans

May 2010



WAGENINGEN UNIVERSITY

WAGENINGEN UR



PEDESTRIAN ROUTE PLANNING IN A HYBRID DATA ENVIRONMENT

Calculating optimal routes based on vector and raster data

Ing. J.H. Bakermans

Registration number 84 10 28 026 020

Supervisors:

Dr. ir. S. De Bruin

Dr. ir. G.B.M. Heuvelink

A thesis submitted in partial fulfillment of the degree of Master of Science
at Wageningen University and Research Centre,
The Netherlands.

May 2010

Wageningen, The Netherlands

Thesis code number: GRS-80439

Thesis Report: GIRS-2010-13

Wageningen University and Research Centre

Laboratory of Geo-Information Science and Remote Sensing

Preamble

This thesis is the result of a long but inspiring journey through the fields of navigation, ArcGIS and Python. Even though moments of (t)error took turns with 'Eureka!', using the Python programming language was a pleasant experience. Also the various obstacles I encountered were interesting challenges that taught me a lot.

Of course while doing your thesis you are surrounded by inspiring people. First of all I would like to thank my supervisors Sytze and Gerard for their outstanding support during this thesis. They really complete each other both with respect to the content as for their criticism.

Also many thanks to my fellow thesis students that provided a great working ambiance. They pulled me through hard times and reminded me that I was not the only stressed student. I could not have done this without you!

Jan Bakermans

Table of content

Preamble	V
Summary	8
1 Introduction	9
1.1 Background	9
1.2 Problem definition	10
1.3 Research objective and research questions.....	11
1.4 Report structure.....	11
1.5 Research delineation.....	12
2 Methodology	13
2.1 Process overview.....	13
2.2 Deriving free walkable space	14
2.3 Assigning network costs and benefits.....	15
2.4 Creating internal paths.....	16
2.4.1 <i>Calculating paths</i>	16
2.4.2 <i>Solving raster issues</i>	17
2.5 Deriving the network dataset.....	17
2.6 Dijkstra's shortest path algorithm.....	18
3 Implementation	19
3.1 Deriving free walkable space	19
3.2 Assigning network costs and benefits.....	21
3.3 Creating internal paths.....	21
3.3.1 <i>Source and destination points</i>	21
3.3.2 <i>Raster paths</i>	23
3.3.3 <i>Internal paths</i>	26
3.4 Deriving the network dataset.....	27
3.4.1 <i>Pedestrian infrastructure</i>	27
3.4.2 <i>Network dataset</i>	27
3.5 Dijkstra's shortest path algorithm.....	28
4 Case study 's-Hertogenbosch	29
4.1 Introduction	29
4.2 Data preparation.....	29
4.2.1 <i>Datasets</i>	29
4.2.2 <i>Preprocessing</i>	30
4.2.3 <i>Workflow</i>	30
4.3 User interaction.....	30
4.3.1 <i>Expert knowledge</i>	30
4.3.2 <i>Scenarios</i>	31
4.4 Results	32
4.4.1 <i>Process</i>	32
4.4.2 <i>Routes</i>	33
5 Discussion	34
5.1 Method.....	34
5.2 Implementation.....	35
5.3 Case study	36
6 Conclusion	37
7 References	38
Annex A: Detailed process overview	39
Annex B: Python script	40

Summary

Navigation is a common daily activity for human beings and people use a wide range of tools to make it easier. Following car drivers pedestrians have found their way to route planning, however a suitable dataset is not yet provided. Due to their origin in car navigation current pedestrian navigation datasets do not include free walkable space. This is open space that can be freely accessed by fit pedestrians e.g. squares, parks or parking lots. This drawback leads to the idea for this research.

It presents an approach that uses a hybrid data environment with both vector and raster data to calculate an optimal path as part of pedestrian route planning. Hence, the research objective is to calculate the optimal route for pedestrians in an urban environment based on a hybrid dataset that contains vector network data to represent corridor-like paths and raster data to represent free walkable space.

To reach the objective a method is provided on a conceptual level. First the free walkable space is to be derived from the topographical map. Next, the free walkable space has to be divided by a user into subspaces with equal accessibility values. The resulting weighted free walkable space together with a road dataset can then be used to calculate paths passing through the free walkable space. In this process a set of fixed paths is also created. Subsequently, the fixed paths and the internal paths are joined into one pedestrian infrastructure dataset. The latter is used to create a network dataset on which Dijkstra's algorithm can be applied to find the shortest path (Dijkstra, 1959).

Implementation of the method takes place by using ESRI's ArcGIS Desktop software and the Python programming language. Furthermore some user actions are required for editing and adding expert knowledge. Finally this implementation was applied on a case study in the city center of 's-Hertogenbosch (NL) and to test resulting network dataset, three source/destination scenarios were assessed.

Overall the method implemented on a case study provides satisfying results as it returns plausible pedestrian routes that are slightly better than conventional methods. Main shortcomings involve parameters to fine tune the implementation: even a minor change can have major effect on the final output. Furthermore, the method lacks an inconsistency check for fixed paths and walkable areas which causes the network to be imperfect. Additionally the raster path calculation used still fails to yield the desired paths.

Future research should focus on the derivation of fixed walkable paths to match with the walkable area and on optimizing path calculation in a raster environment.

1 Introduction

1.1 Background

Navigation is a common daily activity for human beings. It is the process of planning and following a certain trajectory in order to get from one place to another. Within the task of navigation two components can be distinguished: locomotion and wayfinding (Montello, 2005; Wiener, et al., 2009). For this research, wayfinding is the most important of the two. It concerns a set of tasks that involve cognitive processes like decision making and planning. Key issue is that these tasks aim at reaching destinations beyond what is perceived in the direct environment. Locomotion, on the other hand, implies movement of an individual as a reaction on its direct environment but is not further discussed here.

To lighten their wayfinding activities, people use assistance from devices that range from paper maps to location based services. Until recently, the latter were primarily designed to serve car drivers. However, in the past few years they have become more and more developed and have widely gained user acceptance. With the present state of mobile devices real time navigation systems have the potential to target a broader public. Not surprisingly, there is a trend is to get the navigation system out of the vehicle and into the hands of cyclists, hikers and pedestrians.

A different target group brings about considerably different information needs. Pedestrians for example do not content themselves with traditional navigation systems. This target group is able to exploit its cognitive resources in a much more intense way when navigating, thus any information received needs to be cut out for that (Stark, et al., 2007). In line with this requirement, current research around pedestrian wayfinding drives at integrating knowledge of cognitive science, psychology and artificial intelligence into the domain of pedestrian navigation, already striving to provide insight into tailored route directions (Rehrl, et al., 2007).

Where the information needs has been subject of various studies, one of the most common wayfinding tasks still leaves some questions: route planning; an area that has been profoundly investigated for the use in car navigation systems. But application of the same methodology on pedestrian navigation systems is not possible without some major adaptations (Corona & Winter, 2001b). Main reason for this is that pedestrians are not strictly bounded to an infrastructure; they experience a higher degree of freedom in their movements. This phenomenon should somehow be captured in the datasets that form the basis for pedestrian navigation systems (Corona & Winter, 2001a). Attempts to do so exist but do not provide a concrete solution.

(Gaisbauer & Frank, 2008) for example, introduce a wayfinding model which aims to cover all walkable space. Walkable space can be defined as all space that is accessible by pedestrians. Their model is based on the concept of Lynch (1960) who depicts decision points as points where people have to decide where to go based on their perception. Closely related to locomotion, these points involve only the visible route options. The model divides walkable space into decision scenes (i.e. direct surroundings of a decision point) by using the decision points and it subsequently converts these areas to a graph. These represent the internal structure of these decision scenes as a network of portals (nodes) through which pedestrians can enter other decision scenes and connecting routes (edges) that can be used to traverse a scene. Still, this concerns a conceptual model which leaves open ends on the creation of decision scenes and the calculation of paths for the internal structure.

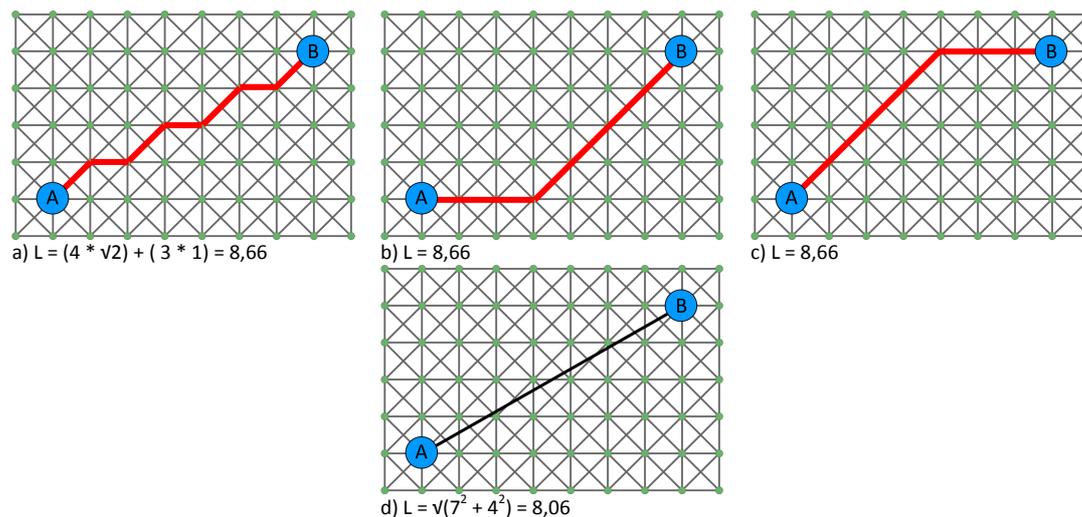


Figure 2) Three paths (a, b, c) calculated from A to B which overestimate the length of the desired path (d)

Based on these issues, seven research questions are formulated. These are stated in the next section.

1.3 Research objective and research questions

The research objective reads as follows:

'To calculate the optimal route for pedestrians in an urban environment based on a hybrid dataset that contains vector network data to represent corridor-like paths and raster data to represent free walkable space.'

This objective forms the basis for seven research questions (RQs) which are formulated below.

- RQ1: How to decide which parts of a study area can be labeled as free walkable space and which areas are to be represented as nodes and edges in a graph?
- RQ2: How can a raster representation of free walkable space be connected with a network of nodes and edges?
- RQ3: How to determine the costs or benefits attached to nodes, edges and raster cells?
- RQ4: How to overcome miscalculations in route costs caused by the square structure of raster data?
- RQ5: How to calculate the optimal path in a hybrid environment?
- RQ6: How can a network dataset suitable for pedestrian route planning be derived from the topographical map available in The Netherlands (TOP10_vector)?
- RQ7: Which implementation issues, suggestions for future research and insights arise from practical application of the methodology?

1.4 Report structure

Chapter 2 presents the methods applied in this thesis at a conceptual level. Subsequently, the more technical implementation of the method is discussed in chapter 3 whereas chapter 4 contains the results of this implementation applied on a case study. The results of chapters

2 through 4 are be discussed in chapter 5. Finally, chapter 6 concludes this research by answering the research questions and suggesting directions for further research.

1.5 Research delineation

This research deals with planning an optimal route for pedestrians in an urban environment. In order to account for the high degree of movement freedom, it integrates the concept of free walkable space which together with a vector network representing narrow corridors forms the basis for calculating such route.

Free walkable space was defined as all space that is accessible by pedestrians. However, because this research is about pedestrian route planning some refinement of this concept has to be made. First of all this research focuses only on the outdoors, in particular on the urban environment. Second, only public areas have been considered for navigation purposes. A third aspect of the concept is the pedestrian in general. This is a mixed group of human beings with all kinds of (dis)abilities, thus the accessibility of areas differs due to this variability. Consequently, a more appropriate and refined definition of 'walkable space' is adopted: "All outdoor space that is open and accessible by fit pedestrians".

Another worth mentioning aspect of this research is the definition of 'optimal'. As mentioned before, end users are not alike thus optimal is a personal concept. To grasp this concept this research was set out to integrate criteria that affect the optimal route at most. Though distance is obviously (one of) the most important criteria in efficient pedestrian route planning, others were likely to be relevant as well.

2 Methodology

2.1 Process overview

To allow pedestrian route planning, there is need for a method that is capable of constructing a pedestrian network. Hence, the goal of the process described here is to yield a network dataset on which a route optimizer (e.g. Dijkstra's shortest paths algorithm (Dijkstra, 1959)) can be applied to calculate the least cost path between start and end points. First, this section presents an overview of the overall process whereas following sections will zoom in to this model to illuminate each subprocess. A flowchart of the entire process discussed in this chapter can be found in Annex A.

Globally, the process comprises two main input datasets and one output dataset; inputs are a polygon based topographical map and a polyline based road dataset and the output is a pedestrian network (Figure 3a).

The first subprocess takes care of deriving the free walkable space from the topographical map and thereby tackles RQ1. This will be discussed in section 2.2.

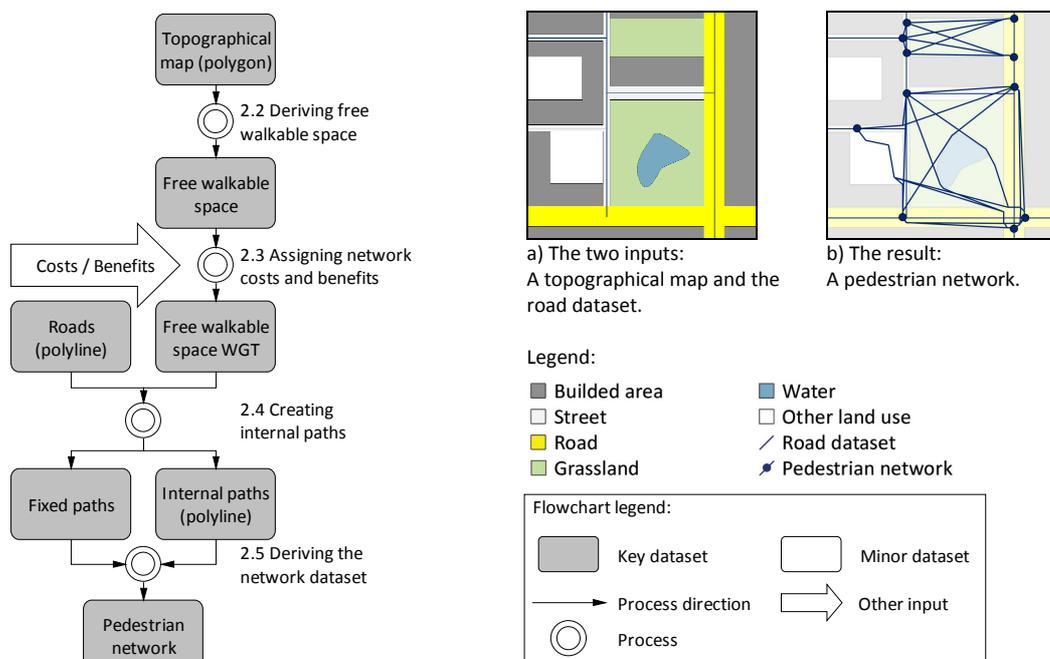


Figure 3) This figure presents a scheme (left) and illustrative examples (right) from the process and its datasets. The addition 'WGT' in the scheme indicates that the dataset is enriched with accessibility weight values. Furthermore, the first example (a) depicts the input datasets whereas example two shows the final result (b).

Next, its product will undergo editing by a user whose job is to divide the free walkable space into subspaces with equal accessibility values, based on his expert knowledge of the site or any available data. This process relates to RQ3 and will be discussed in section 2.3.

Section 2.4 zooms in on the next subprocess which relates to both RQ2 and RQ4. It uses the so-called weighted free walkable space as an input together with the road dataset to calculate paths passing through the free walkable space, leading from one road to another. A side product of this subprocess is the set of fixed paths that conjoin the free walkable spaces.

The subprocess of deriving the network dataset combines both the fixed paths and the internal paths into one joined pedestrian network dataset (Figure 3b). It applies to RQ6 and will be explained in section 2.5. Finally, a network dataset is derived that is suitable for pedestrian route planning (RQ5) as is elaborated in section 2.6.

2.2 Deriving free walkable space

This is the subprocess where the free walkable space is extracted from the main input topographical dataset. A suitable dataset can be any topographical dataset that is polygon based but a large scale is preferred to provide enough detail to discern roads and open areas (i.e. scale 1:10.000).

All topography classes in the dataset were reclassified into areas that are 'walkable' or areas that are 'NOGO' in which areas marked 'walkable' are potentially eligible for a free walkable space representation (Figure 4a, b). The reclassification is done best based a logical interpretation of the classes. However, reclassification of some classes is doubtful and additional information is required. For example a class like 'other land use' can correspond to a building but might as well be an open parking space, or the class 'grassland' could be a (fenced) meadow but it could also be represent an urban park. In such case the classification 'walkable' was assigned. After all, a detailed assessment of these areas was done later when site knowledge was used to assign costs to zones within the free walkable space.

The freedom of movement of pedestrians is assumed not to be limited by a transition in topographic class given that both classes are marked 'walkable'. For example when a pedestrian walks from a square into a field he does not experience a barrier. Therefore all boundaries between walkable areas are dissolved to yield continuous areas of walkable space (Figure 4c).

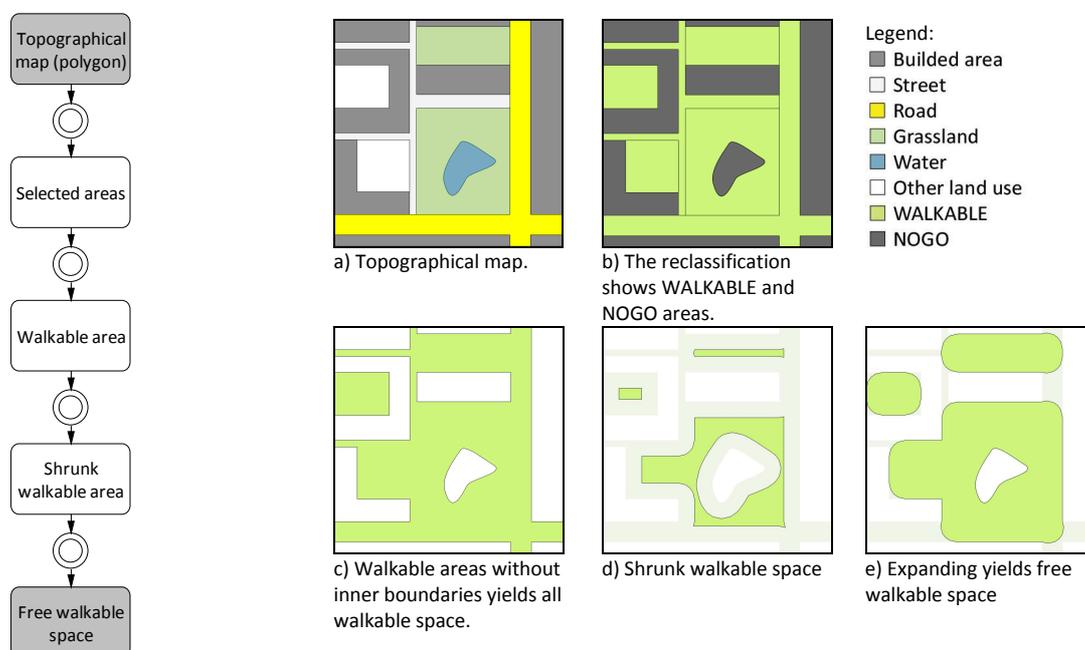


Figure 4 In this figure the process of deriving free walkable space is shown. First the scheme (left) subsequently a visualization. The process starts of with a topographical map (a), next a reclassification (b), then the boundaries are removed (c), subsequently the areas are shrinked (d) and finally its proportions are restored (e). Note that roads are included at a later stage.

Furthermore, a pedestrian experiences narrow parts (e.g. roads) as corridors in which only one route option is available: follow the corridor until the next decision point (Lynch, 1960). Therefore only the wide open spaces are suitable to be represented as free walkable space in the network. Wide open spaces were found by shrinking the complete walkable space from the edge inwards until the narrow parts vanish. What remains are cores of objects that indicate the presence of areas which are large enough to be represented as free walkable space (Figure 4d).

The final representation of free walkable space was realized by expanding the residue of the shrink action in such a way that it is reinstated to the original proportions of the walkable space. The product then represents all free walkable areas but not the narrow streets. (Figure 4e)

2.3 Assigning network costs and benefits

The next subprocess is a laborious one. The user has to assign costs to the given free walkable spaces (Figure 5). This can be done by means of software which is capable of editing geodata. It involves modifying the shapes and altering their attribute values. By manually dividing the free walkable space into smaller zones the user enables him to assign weights to specific locations inside the free walkable space (Figure 5a, b). If necessary, he can extend his expert knowledge of the site by consulting aerial photographs, detailed city plans or other resources.

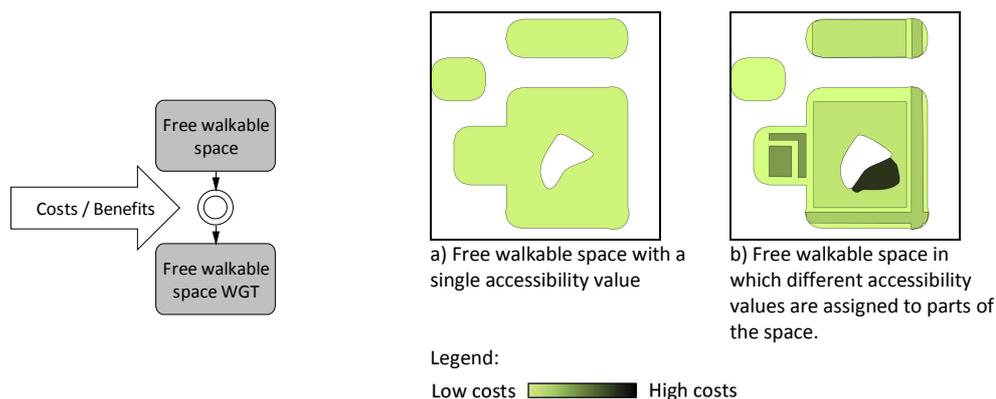


Figure 5) Here the process of adding costs to the free walkable space is depicted by means of a schem (left) and visual examples (right) . The input is the free walkable space dataset (a), output is this space augmented with accessibility values (b).

The detailed information that the user provides is about accessibility and obstacles. In this respect, accessibility implies zones that are less accessible to pedestrians like roads with heavy traffic or areas with dense vegetation. Obstacles are for example fences, hedges and ponds but also buildings. Because a least cost path calculation will be used to calculate paths this detailed information can be represented as costs. Note that the assignment of costs is merely subjective and that relative figures can be used, some examples:

- A park can be divided into easy accessible pathways with weight 1, a less accessible field with weight 3 and an inaccessible pond with weight 99;
- A traffic junction contains specific footpaths for pedestrians so here a division can be inaccessible roads with weight 99 and perfectly accessible sidewalks with weight 1.

With this result, a weighted free walkable space dataset has been generated that can be used for calculation of the internal paths.

2.4 Creating internal paths

2.4.1 Calculating paths

The next step is to gather all paths passing through the free walkable space, and with that, connecting all roads leading to the free walkable space in question. To achieve this, first all points are retrieved in which a road enters a free walkable space. With this process, a side product is produced. It contains the difference of the roads in the original road dataset and the free walkable space: the fixed paths (Figure 6a). These will be used later on in the process to complete the pedestrian network because they represent the connections between the various free walkable spaces.

The above mentioned points are actually the intersections of the polyline road dataset with the boundary of free walkable space objects and constitute the begin and end points for the path calculation (Figure 6b). Next, a raster environment is established containing for each cell the cost of traveling through that cell which is based on the weight value of the corresponding user input (Figure 6c, d). The so-called cost grid is essential for path calculation in a raster environment because it determines along which cells the actual raster path will travel. Unfortunately the costs defined by the user will not suffice in case the cost grid is traversed diagonally; in that case the least cost path calculation brings about some issues regarding the square structure of the raster environment. Section 2.4.2 describes how these issues are dealt with.

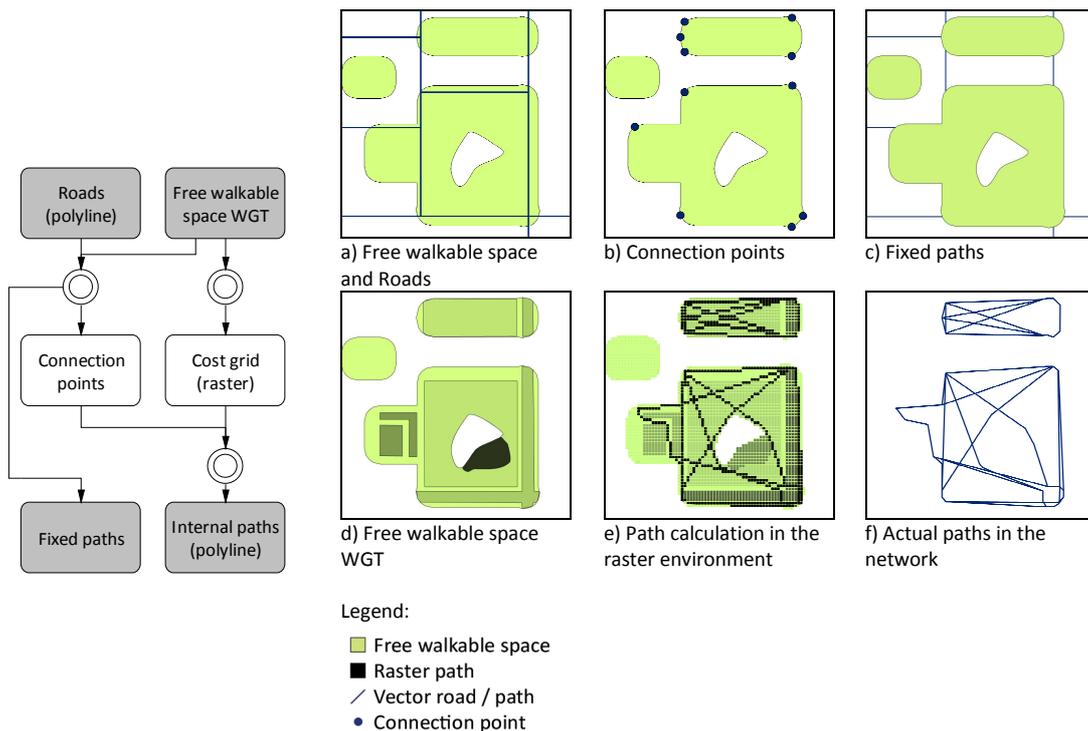


Figure 6) This figure illustrates the process of creating the internal paths. Besides the scheme (left) some illustrative examples are included (right). Point of departure is the free walkable space and the road dataset (a). From these, connection points (b) and fixed paths are retrieved (c). The weighted free walkable space (WGT) is used (d) to calculate paths in the raster environment (e). The process yields the actual paths in the network (f). Note that nodes in the road network can disappear when overlaid by free walkable space.

The path calculation is done for all possible connections inside a free walkable space object (Figure 6e). For an object with a collection of n connection points the number of possible internal connections is: $n(n-1)/2$. Each path is based on the least cost path calculated in the

raster environment. It is converted to a polyline and added to the dataset of internal paths (Figure 6f). In this way each possible path is represented as an edge in the actual network.

2.4.2 Solving raster issues

The use of a Moore neighborhood causes the raw weights provided by the user to be impractical for direct raster path calculation. This is because the weights per cell are not yet related to the direction of the path crossing it. The actual weight raster has a cost/distance relationship covered by means of the integration of Euclidian distance. There are several steps required to get there.

First, the raster dataset which is created from the user's weight input is used to calculate for each cell the accumulated path cost to that cell from a source. This is done for all connection points. The next step is to combine the grids of a source and destination point into a least cost corridor. Again, this is done for each point combination inside a free walkable area.

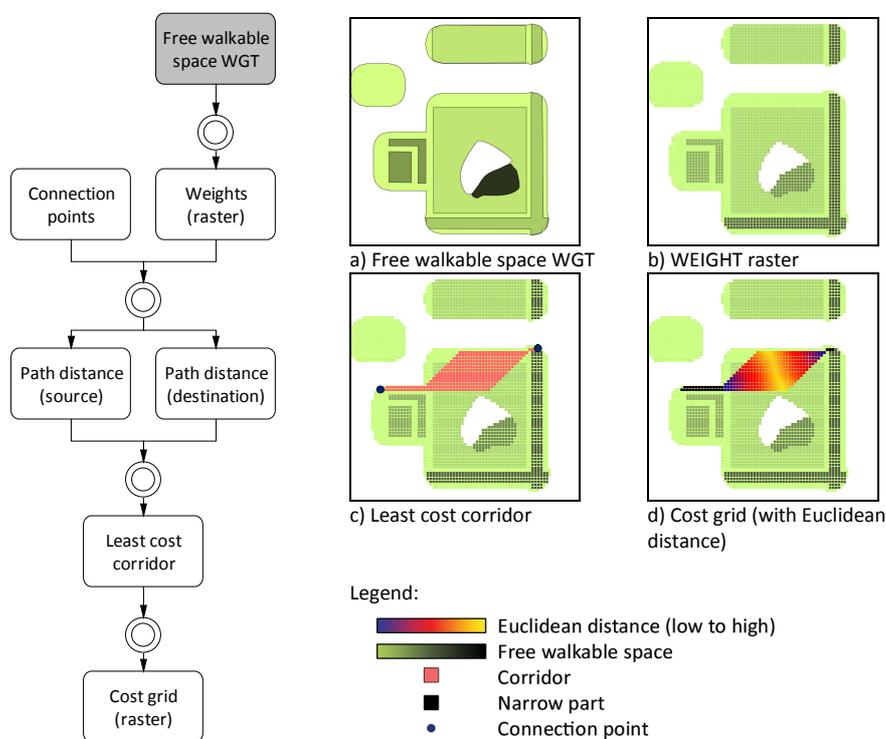


Figure 7 Process scheme (left) of creating the cost grid illustrated with examples (right). The polygon dataset of weighted free walkable space (a) is converted to a weight raster (b). With this raster for each combination of connection points two path distance grids are calculated that result in a least cost corridor (c). This corridor is enriched with Euclidean distance values to form the cost grid (d)

The resulting corridor represents a narrow lane through which the actual path has to go. It could however, contain wider parts where there is more than one route option available due to the use of the Moore neighborhood. For these wider parts of the corridor, the Euclidean distance is calculated from each cell to the nearest narrow part. These distance values make the actual cost grid. So the further away a cell is from a narrow part, the higher the cost value is.

2.5 Deriving the network dataset

The final step in constructing the network dataset consists of combining the calculated internal paths (Figure 8a) with the fixed paths (Figure 8b). The features from both of these

datasets are transferred into a single feature class (Figure 8c). As the previous process restored the connections between all begin and end points, the final dataset offers the same routing possibilities as those that the original road dataset offered augmented with new possibilities that have emerged from the path calculation in the free walkable space. It is a complete pedestrian network dataset in which the movement freedom in free walkable space is captured.

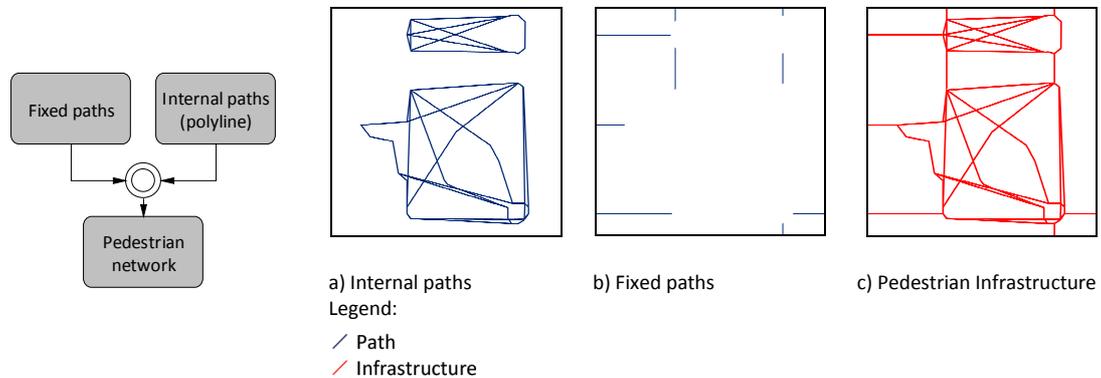


Figure 8) The creation of the pedestrian infrastructure visualized as a scheme (left) and examples (right). The internal paths (a) and the fixed paths (b) are combined into one pedestrian infrastructure (c)

2.6 Dijkstra's shortest path algorithm

The final pedestrian route planning will take place within an environment that solely consists of vector data. After all, all optimal routes within the raster environment are now represented as edges in a pedestrian network. The algorithm used is the Dijkstra's shortest path algorithm which is able to solve the shortest path problem for a graph with nonnegative edge path costs by constructing trees of minimum total length (Dijkstra, 1959). The concept of 'shortest path' should in fact be understood as 'the least cost path' because the costs used in this study do not represent Euclidean distance, but rather a cost distance. For example, walking through grass or crossing a busy road is more expensive than walking over a footpath.

3 Implementation

The methods discussed in the preceding chapter involve various data actions. In order to implement all these processes, the Python programming language was used to formulate a script for each of them. Python is open source and is able to exploit the geoprocessor of ESRI's ArcGIS Desktop software. So a range of tools in ArcGIS and the strength of programming were combined to yield the result. This chapter discusses for each process which tools were used, with which in- and outputs and how they were interconnected with Python. The Python script is enclosed in Annex B.

3.1 Deriving free walkable space

This process needs the topographical dataset as input. From that, the walkable area was retrieved by selecting only those topographical types that were marked 'walkable' and storing them in a new feature class. The tools used are given in Table 1.

Table 1) Tools needed to create a feature class of the walkable areas

Tool 1	MakeFeatureLayer_management
In	Topographical dataset (main input)
Out	Selectable layer
Tool 2	SelectLayerByAttribute_management
In	Selectable layer from tool 1 SQL query to select the proper features based on the reclassification of topographical type.
Out	Selection of walkable areas
Tool 3	CopyFeatures_management
In	Selection of walkable areas from tool 2
Out	New feature class with only the selected walkable areas.

The input used for tool 2 was based on an expert-driven reclassification. Table 2 shows an example in which a self explaining reclassification of the Topographical map of The Netherlands is done (TOP10vector). It was based on its topography field and the reclassification can be seen as a Boolean map: NOGO (0) and WALK (1).

Table 2) Example of reclassification. Used dataset: TOP10 vector, scale 1:10.000, dated 2006

Description (EN)	Description (NL)	Topography field (TOPO_CODE)	Reclassification
Build-up area	Beb. Gebied/Huizenblok	1013	NOGO
Large building	Groot Gebouw	1023	NOGO
Main connection road	Hoofdverbindingroute 7	2303	NOGO
Connection road 8	Verbindingsroute 8	3003	NOGO
Connection road > 7	Verbindingsroute >7	3103	NOGO
Local road > 7	Lokale weg >7	3143	WALK
Local road > 4	Lokale weg >4	3243	WALK
Local road > 2	Lokale weg >2	3343	WALK
Other road type > 2m	Overige weg >2m	3403	WALK
Partly metalled road 3	Ged. verharde weg 3	3413	WALK
Unmetalled road 3	Onverharde weg 3	3433	WALK
Pedestrian zone	Voetgangersgebied	3473	WALK
Street	Straat	3533	WALK
Cycle path	RWP Rijwielpad	3603	WALK
Parking area	Parkeerterrein	3903	WALK
Deciduous forest	Loofbos	5023	WALK

Grassland	Weiland	5213	WALK
Other land use	Overig bodemgebruik	5263	WALK
Water	Landblauw	6113	NOGO
Dock	Aanlegsteiger	6513	WALK

The next step was to shrink the walkable areas, which is relatively easy as only two tools were required. First the Dissolve tool was used to create a seamless object containing all walkable areas. Next, the Buffer tool was used with a negative value as buffer distance. Consequently, the free walkable space was created by applying the same Buffer tool with the inverse buffer distance. Table 3 depicts the in- and output used for these tools.

Table 3) From walkable areas to free walkable space

Tool 4	Dissolve_management
In	The feature class with selected walkable areas from tool 3
Out	Dissolved area
Tool 5	Buffer_analysis
In	In_feature: Dissolved area from tool 4 Buffer_distance: -15 meters
Out	Area minus a buffer
Tool 6	Buffer_analysis
In	In_feature: Area minus a buffer from tool 5 Buffer_distance: + 15 meters
Out	Free walkable space

Due to the dissolve performed in tool 4, the feature class contained a single feature consisting of many parts (a MultiPolygon object). Applying the Multipart To Singlepart tool (tool 7) yielded a dataset consisting of single part polygons (i.e. one polygon per record).

After this process the output has to undergo considerable manual adaptations. Therefore the dataset was prepared for editing (Table 4). First of all an Area_ID and a WEIGHT field were added. The first one was assigned the Object_ID of the corresponding feature and the latter was for initially assigned a one, later on in the process this value is to be edited manually. A backup was created of this feature class to have a clean slate in case something would go wrong during editing.

Table 4) Tools needed to create a feature class of the walkable areas

Tool 7	MultipartToSinglepart_management
In	Free walkable space with just one all-embracing feature from tool 6
Out	Free walkable space with separate features for all single parts
Tool 8	AddField_management (Tool is run twice to add two fields)
In	In_feature: Free walkable space from tool 7 Field_name: "Area_ID" and "WEIGHT" Field_type: Double Precision: Number of digits used is 5 Scale: Number of digits after separator is 0
Out	Fields added to the input feature
Tool 9	CalculateField_management (Tool is run twice to calculate two fields)
In	In_feature: Free walkable space from tool 8 Field names: Those created in tool 8 Values: "Object_ID" and "1"
Out	Free walkable space with fields calculated
Tool 10	CopyFeatures_management
In	Free walkable space from tool 9
Out	Free walkable space back up

3.2 Assigning network costs and benefits

Assigning weights to the free walkable space was done manually inside ArcMap. Each free walkable space feature was cut up into smaller pieces (Table 5 and Figure 9 (left)). While doing this, the Area_ID and WEIGHT field adopted the value from their parent object. Each cutting is then retraceable to an embracing free walkable space object. The weights were added by editing the WEIGHT field (Figure 9(right)). The edited dataset was used as an input for the process of creating internal paths (section 3.3)

Table 5) Manual actions performed to edit the free walkable space dataset

Action 1	Edit free walkable space from tool 9
Modify	Divide polygons into zones
Edit values	Alter WEIGHT values

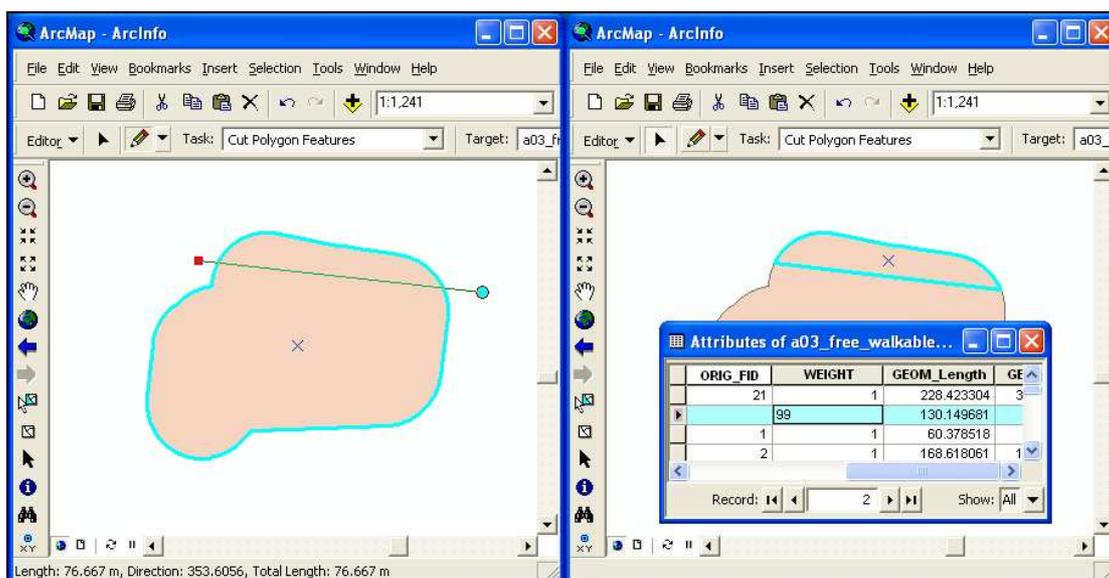


Figure 9) Editing Free walkable space with ArcMap

3.3 Creating internal paths

3.3.1 Source and destination points

The road dataset was overlaid with the unedited free walkable space by means of the Erase tool. This resulted in the fixed walkable paths. Table 6 shows the tool and its inputs.

Table 6) Originating of the fixed paths dataset

Tool 11	Erase_analysis
In	In_feature: Road dataset (main input) Erase_feature: Free walkable space from tool 9
Out	Fixed paths

The end vertices of the fixed paths that touch the boundaries of a free walkable space are seen as 'connection points'. To get these, first all end vertices were converted to point features using the Feature Vertices To Points tool. Next, a feature layer was made from these points to be able to select features by their location. Due to digital inaccuracies a search distance was applied to make sure that all points within a distance of 0.1 meter from the parent free walkable space were selected. The whole selection was stored in a feature class by using the Copy Feature tool (Table 7).

Table 7) Tool sequence that yields all connection points

Tool 12	FeatureVerticesToPoints_management
In	In_feature: Fixed paths dataset from tool 11 Point_location: Determines which vertices will be converted. In this case only both ends.
Out	Point dataset
Tool 13	MakeFeatureLayer_management
In	Point dataset from tool 12
Out	Selectable layer
Tool 14	SelectLayerByLocation_management
In	In_layer: Selectable layer from tool 13 Select_feature: Features to base selection on, this is the free walkable space dataset. Overlap_type: Defines on what condition the points will be selected. The points that are WITHIN the select features. Search_distance: 0.1 meters
Out	Selection
Tool 15	CopyFeatures_management
In	Selection from tool 14
Out	Connection points

The resulting dataset contained all connection points. For future reference an Area_ID and a Point_ID field were added to the connection point dataset of which the values were calculated later (Table 9). The connection points were placed in a selectable layer (Table 8).

Table 8) Tools to add area and point ID fields plus creating a selectable layer.

Tool 16	AddField_management (Tool is run twice to add two fields)
In	In_feature: Connection point dataset from tool 15 Field name: "Area_ID" and "Point_ID" Field type: both "SHORT"
Out	Connection points with fields added
Tool 17	MakeFeatureLayer_management
In	Connection points from tool 16
Out	Selectable layer

While the script from block 1 looped through all free walkable space features, each feature was placed into a layer with the Make Feature Layer tool. Next, points that were within the current feature were selected. This was done by using the current feature layer as a selection mask and the connection point layer to select points from. The result is a set of connection points belonging to the current feature. This set was altered by the Calculate Field tool which set the Area_ID to the Object_ID of the current feature. (Table 9)

Table 9) Script block to iterate ID assignment

Block 1	Looping through the free walkable space						
In	Selectable layer from tool 17						
Process	For each feature in Free walkable space back up, calculate AreaID:						
	<table border="1"> <tr> <td>Tool 20</td> <td>CalculateField_management</td> </tr> <tr> <td>In</td> <td>In_feature: Free walkable space back up from tool 10 Where_clause: ObjectID = current ObjectID</td> </tr> <tr> <td>Out</td> <td>Layer with area selected by where_clause</td> </tr> </table>	Tool 20	CalculateField_management	In	In_feature: Free walkable space back up from tool 10 Where_clause: ObjectID = current ObjectID	Out	Layer with area selected by where_clause
Tool 20	CalculateField_management						
In	In_feature: Free walkable space back up from tool 10 Where_clause: ObjectID = current ObjectID						
Out	Layer with area selected by where_clause						
	<table border="1"> <tr> <td>Tool 19</td> <td>SelectLayerByLocation_management</td> </tr> <tr> <td>In</td> <td>In_feature: Selectable layer from tool 17 Select feature: Area layer from tool 18 Overlap type: WITHIN Search_distance: 0.1 meters</td> </tr> <tr> <td>Out</td> <td>Selection of points that belong to area</td> </tr> </table>	Tool 19	SelectLayerByLocation_management	In	In_feature: Selectable layer from tool 17 Select feature: Area layer from tool 18 Overlap type: WITHIN Search_distance: 0.1 meters	Out	Selection of points that belong to area
Tool 19	SelectLayerByLocation_management						
In	In_feature: Selectable layer from tool 17 Select feature: Area layer from tool 18 Overlap type: WITHIN Search_distance: 0.1 meters						
Out	Selection of points that belong to area						

In	in_layer: Selectable layer from tool 19 Field_name: "Area_ID" Value: Current ObjectID
Out	Connection point dataset with ArealD field calculated
<p>For each point in pointset selected by tool 19: Set value of Point_ID field to i, Increase i with 1 (Starting with i = 1)</p>	
Out	Connection point dataset with ID fields calculated

As a result, all points have a corresponding ArealD and for each area the points were numbered 1 to n in which n is the total number of points within the area.

3.3.2 Raster paths

Data preparation and settings

Some actions were carried out to prepare the iterative process of calculating paths. That is, the weight dataset was converted to raster, a feature class was created to contain the output polyline paths and connection points were expanded. The latter one means converting the connection points to a polygon object by buffering them. This was necessary since the used tools convert objects to raster cells when they are used as source or destination inputs. Since all connection points are located on the edge of the rasterized weight dataset there was a chance their raster equivalent laid outside the raster mask. The expanded connection points yield more cells than its parent point objects what ensured that at least one cell fell within the rasterized area. (Table 10)

Table 10) Tool sequence that buffer connection points

Tool 21	PolygonToRaster_conversion
In	In_feature: Free walkable space WGT from action 1 Value_field: "WEIGHT" Cell_size: 1 meter
Out	Rasterized areas
Tool 22	CreateFeatureClass_management
In	Path: Path of the new feature class Name: Name of the new feature class Geometry_type: Polyline
Out	New polyline feature class for internal paths
Tool 23	Buffer_analysis
In	Connection points from block 1 Buffer distance: 1 meter
Out	Buffered connection points

The iteration contained a nested loop to calculate all possible connections based on the Point_ID that was assigned in block 1. This loop was regulated by a parameter n that held the total number of connection points for a feature. This parameter was stored as the attribute 'Range'. Adding the attribute field happens with the usual tool while it takes a script block to calculate the value for n . (Table 11)

Table 11) Tool sequence that calculates fields for connection points.

Tool 24	AddField_management
In	In_feature: Buffered connection points from tool 23 Field_name: "Range" Field_type: Short
Out	Buffered connection points with Range field added

Block 2	Calculating n values																		
In	Buffered connection points with Range field from tool 24																		
Process	<p>For all buffered connection points: Add AreaID to list (no duplicates)</p> <p>For each AreaID in list:</p> <table border="1" data-bbox="464 405 1240 620"> <tr> <td>Tool 25</td> <td>MakeTableView_management</td> </tr> <tr> <td>In</td> <td>In_feature: Buffered connection points from tool 24 Where_clause: AreaID = current AreaID</td> </tr> <tr> <td>Out</td> <td>Table with all points of the current area</td> </tr> <tr> <td>Tool 26</td> <td>Statistics_analysis</td> </tr> <tr> <td>In</td> <td>In_table: Table from tool 25 Statistics_field: Point_ID MAX</td> </tr> <tr> <td>Out</td> <td>Statistics table</td> </tr> </table> <p>The value of n is retrieved from the statistics table and is stored in a variable: Range_max</p> <table border="1" data-bbox="464 732 1240 869"> <tr> <td>Tool 27</td> <td>CalculateField_management</td> </tr> <tr> <td>In</td> <td>In_table: Table from tool 25 Field_name: "Range" Value: Range_max</td> </tr> <tr> <td>Out</td> <td>Range calculated</td> </tr> </table>	Tool 25	MakeTableView_management	In	In_feature: Buffered connection points from tool 24 Where_clause: AreaID = current AreaID	Out	Table with all points of the current area	Tool 26	Statistics_analysis	In	In_table: Table from tool 25 Statistics_field: Point_ID MAX	Out	Statistics table	Tool 27	CalculateField_management	In	In_table: Table from tool 25 Field_name: "Range" Value: Range_max	Out	Range calculated
Tool 25	MakeTableView_management																		
In	In_feature: Buffered connection points from tool 24 Where_clause: AreaID = current AreaID																		
Out	Table with all points of the current area																		
Tool 26	Statistics_analysis																		
In	In_table: Table from tool 25 Statistics_field: Point_ID MAX																		
Out	Statistics table																		
Tool 27	CalculateField_management																		
In	In_table: Table from tool 25 Field_name: "Range" Value: Range_max																		
Out	Range calculated																		
Out	Buffered connection points with Range field calculated																		

Additionally, the overall maximum is required to set the limit for the iteration. The following script block takes care of that (Table 12):

Table 12) Block to calculate iteration limit

Block 3	Calculating overall maximum n						
In	Buffered connection points with Range field calculated from block 2						
Process	<table border="1" data-bbox="464 1122 1240 1234"> <tr> <td>Tool 28</td> <td>Statistics_analysis</td> </tr> <tr> <td>In</td> <td>In_table: Table from tool 27 Statistics_field: Point_ID MAX</td> </tr> <tr> <td>Out</td> <td>Statistics table</td> </tr> </table> <p>The value of n is retrieved from the statistics table and is stored in a variable: Range_max</p>	Tool 28	Statistics_analysis	In	In_table: Table from tool 27 Statistics_field: Point_ID MAX	Out	Statistics table
Tool 28	Statistics_analysis						
In	In_table: Table from tool 27 Statistics_field: Point_ID MAX						
Out	Statistics table						
Out	Variable with overall maximum n						

Path calculation

Internal paths were calculated in a raster environment. The script block performed this and instantly added the path as a polyline to the feature class of internal paths. The process is extensively discussed at a conceptual level in section 2.4. Table 13 shows how this operation was structured and what tools were used. The insertion of paths into the polyline feature is discussed in section 3.3.3

Table 13) Structure of path calculating block

Block 4	Calculate paths				
In	Buffered connection points from block 2 Rasterized areas from tool 21 Variable 'Range_max' from block 3 Free walkable space backup from tool 10				
Process	<p>For each fromID ranging from 1 to Range_max: Set variable Range_to = fromID + 1 For each toID ranging from Range_to to Range_max + 1: Perform loop in which fromID and toID are variables.</p> <table border="1" data-bbox="539 1984 1240 2040"> <tr> <td>Tool 29</td> <td>Select_analysis</td> </tr> <tr> <td>In</td> <td>In feature: Buffered connection points from block 2</td> </tr> </table>	Tool 29	Select_analysis	In	In feature: Buffered connection points from block 2
Tool 29	Select_analysis				
In	In feature: Buffered connection points from block 2				

	Where_clause:	PointID = fromID AND Range <> Point ID
Out	Path sources	
Tool 30	Select_analysis	
In	In feature: Buffered connection points from block 2 Where_clause: PointID = toID	
Out	Path destinations	
Tool 31	PathDistance_sa	
In	In_feature: Path sources from tool 29 Cost_raster: Rasterized areas	
Out	Source distance raster	
Tool 32	PathDistance_sa	
In	In_feature: Path destinations from tool 30 Cost_raster: Rasterized areas	
Out	Destination distance raster	
Tool 33	Corridor_sa	
In	Destination distance raster from tool 32 Source distance raster from tool 31	
Out	Least cost corridor	
Tool 34	ZonalStatistics_sa	
In	Free walkable space Backup from tool 10 Zone_field: ObjectID In_value_raster: Corridor from tool 33 Statistics_type: Minimum Ignore_nodata: Data	
Out	Areas of which all cells are assigned the minimum corridor value of that area.	
Tool 35	SingleOutputMapAlgebra_sa	
In	Conditional statement	
Out	Boolean map whether corridor cells are equal to the minimum value or not	
Tool 36	SetNull_sa	
In	In_raster: Boolean map from tool 35 In_false_constant: 1 Where_clause: Value = 0	
Out	Only corridor cells equal to the minimum value	
Tool 37	SingleOutputMapAlgebra_sa	
In	Focal expression to retrieve the sum of cells within a radius of 3 meters	
Out	Corridor with sum values	
Tool 38	SingleOutputMapAlgebra_sa	
In	Focal expression to retrieve the maximum value of all cells within a rectangle of 5x5	
Out	Corridor with maximum values	
Tool 39	Reclassify_sa	
In	In_raster: Corridor with max values from tool 38 Reclass_field: Value Remap: Values below 25 become 1, 25 and further become NoData	
Out	Narrow parts of the corridor	
Tool 40	EucDistance_sa	
In	Narrow parts of the corridor from tool 39	
Out	Euclidean distance inside the corridor	
Tool 41	Plus_sa	
In	In_raster: Euclidean distance raster from tool 40 In_constant: 1	
Out	Euclidean distance raster with a minimum value of 1 instead	

	of 0
Tool 42	CostBackLink_sa
In	In_source: Path source dataset from tool 29 In_cost: Euclidean distance raster from tool 41
Out	Back link raster Distance raster
Tool 43	CostPath_sa
In	Back link raster from tool 42 Distance raster from tool 42 Path_type: Each Zone Destination_field: AreaID
Out	Least cost paths for each handled area
Tool 45	Con_sa
In	In_raster: Least cost paths from tool 43 Constant: 1
Out	Paths with a single value
Tool 46	RasterToPolyline_conversion
In	In_raster: Paths with single value from tool 45 Simplify: Simplify
Out	Paths with a single value
	Final action in this loop added the polylines to the feature class with internal paths created by tool 22. Section 3.3.3 zooms in to this operation which contains: Tool 47 Tool 48 Tool 49 Block 5
Out	Internal paths

3.3.3 Internal paths

As part of script block 4, the converted paths were inserted to the feature class with internal paths. Therefore begin and endpoints had to match with the corresponding connection points. This was done by the following tools that disentangled the paths and fitted them into the feature class (Table 14).

Table 14) Adding paths to the internal paths dataset

Tool 47	FeatureVerticesToPoints_management
In	Polyline paths from tool 46
Out	Path point dataset
Tool 48	GenerateNearTable_analysis
In	In_feature: Path point dataset from tool 47 Near_feature: Connection point dataset from block 1 Search_radius: 1 Location: Location
Out	Near table
Tool 49	TableSelect_analysis
In	In_table: Near table from tool 48 Where_clause: Near distance < search_radius from tool 48
Out	Table selection of connection points closest to end points
Block 5	Add path to polyline dataset
In	In_feature_1: Polyline paths with a single value from tool 46 In_feature_2: Path point dataset from tool 47 In_table: Table selection from tool 49 In_feature_3: Internal paths from tool 22
Process	For each path in In_feature_1: Loop through points in In_feature_2 that belongs to this path (where: ORIG_FID = current ObjectID) Get record from In_table where IN_FID = ObjectID of the first point and set [begin point] to its NEAR coordinates

Add [mid section] points from In_feature_2	
Get record from In_table where IN_FID = ObjectID of the last point and set [end point] to its NEAR coordinates	
Finally, insert [begin_point + mid_section + end_point] as a polyline to In_feature_3	
Out	Internal paths

3.4 Deriving the network dataset

3.4.1 Pedestrian infrastructure

The pedestrian infrastructure was constructed from the internal paths and the fixed paths. First a copy was made to preserve the original fixed path dataset and then the internal paths were appended to the fixed paths (Table 15).

Table 15) Combining the datasets

Tool 50	Copy_management
In	Fixed paths dataset from tool 11
Out	Copy of fixed paths

Tool 51	Append_management
In	In_feature: Internalpaths from block 4 Target: Copy of fixed paths from tool 50
Out	Pedestrian infrastructure

3.4.2 Network dataset

The final network dataset was constructed out of the pedestrian infrastructure feature class from tool 51. This operation could not be implemented by means of a script as the ArcGIS geoprocessor lacks the commands to automate this. However, ESRI does offer a way to activate this action manually using its ArcCatalog component. Globally four actions were required. First a new feature dataset was created; second the pedestrian infrastructure dataset was copied into this feature dataset; next the network dataset was constructed out of the feature dataset and finally the network was build. The settings used for this action are depicted in the table below (Table 16).

Table 16) Manual actions performed to construct the Pedestrian Network

Action 2	Creating a new feature dataset
Coordinate system for XY coordinates:	Import spatial reference from the pedestrian infrastructure dataset.
Coordinate system for Z coordinates:	None
Tolerance for XY, Z and M:	Default values were retrieved from importing the spatial reference
Resolution and domain extent:	Default values

Action 3	Tool: CopyFeatures_management
In:	Pedestrian infrastructure from tool 51
Out:	Pedestrian infrastructure feature class in the feature dataset from manual action 1

Action 4	Constructing the network dataset
Name of the new Network Dataset:	Pedestrian_Network
Participating Feature classes:	Pedestrian infrastructure from manual action 2
Connectivity settings:	Default (nodes at coincident endpoints)
Modify with elevation data:	NO
Model turns:	NO
Cost attributes:	Default attribute based on shape length
Driving direction settings:	None

Action 5	Tool: BuildNetwork_na
In:	Network dataset from manual action 3
To:	Builed Network Dataset

3.5 Dijkstra's shortest path algorithm

Calculating a least cost path by means of the Network Analyst toolbox within ArcMap requires a feature class which holds a source and destination location. The action to create this feature class was initiated manually in ArcCatalog (Table 17).

Table 17) Manual action to create source / destination feature class

Action 6	Tool: CreateFeatureClass_management
In	Path: Path of the new feature class Name: Name of the new feature class Geometry_type: Point features
Out	New point feature class for locations

Source and destination locations were added to the feature class by editing the point dataset in ArcMap and creating new point features (Table 18).

Table 18) Manual actions performed to add points to the point feature class

Action 7	Add points
Create new:	Create new location points and add location information.

The actual route calculation needed three more actions to be done. First a route layer was created of the network dataset which embraced all input for the desired route. Next, the source and destination points were added as locations to this route layer. Finally, the route layer was solved which yields the optimal route in de underlying network dataset. The tools that were activated are shown in Table 19.

Table 19) Manual actions to solve the route

Action 8	Tool: MakeRouteLayer_na
In:	In_analysis_network: Network Dataset from action 4 Out_network_analysis_layer: Impedance_attribute: Shape length
Out:	Route layer

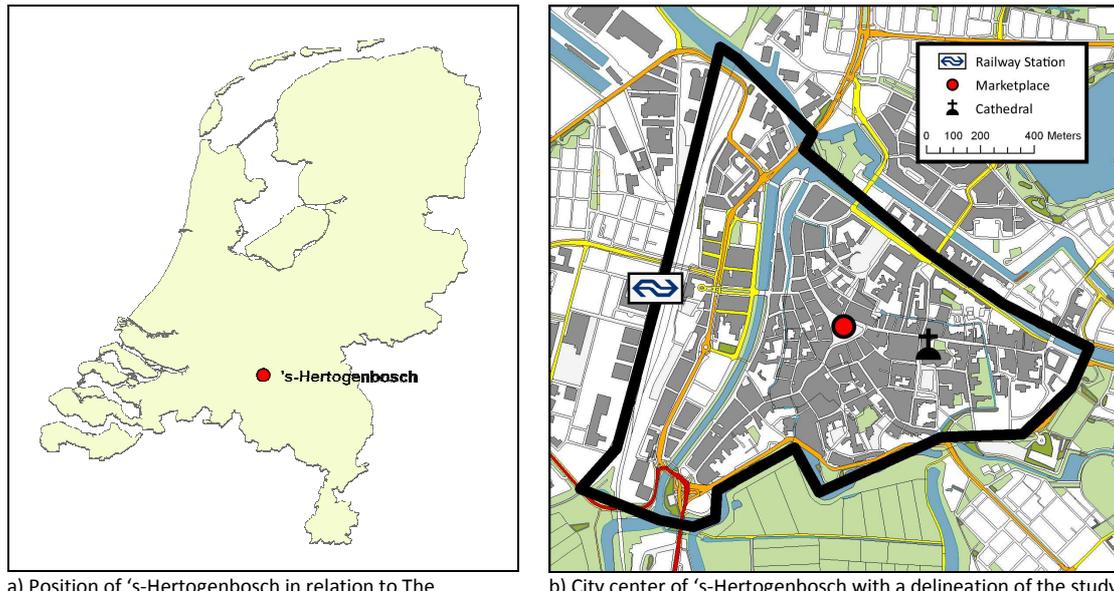
Action 9	Tool: AddLocations_na
In:	in_analysis_layer: sub_layer: Network Dataset from action 4 in_feature: Point feature class from action 7 field_mappings: Constant property value search_tolerance: 10 meters snap_to_position_along_network: NO_SNAP
Out:	Route layer with source and destination locations

Action 10	Tool: Solve_na
In:	In_analysis_layer: Route layer from action 9 Ignore_invalids: HALT
Out:	Route layer with optimal route from source to destination location

4 Case study 's-Hertogenbosch

4.1 Introduction

The method and associated tools discussed in chapters 2 and 3 was applied on a case study for the city center of 's-Hertogenbosch in The Netherlands (Figure 10a). The study area is confined to the neighborhoods 'Het Zand', 'Binnenstad-Centrum' and 'Binnenstad-Oost'. Important locations within this area are the 's-Hertogenbosch Central Station of the Dutch Railways (NS), the city market place and its renowned cathedral Sint-Jan (Figure 10b).



a) Position of 's-Hertogenbosch in relation to The Netherlands

b) City center of 's-Hertogenbosch with a delineation of the study area (in black) and the position some interesting locations.

Figure 10 Geographical context of the case study with its position in relation to The Netherlands (a) and a detailed map of the study area (b).

This chapter presents the results of the implementation. First a short description is given of the two input datasets that were used and how they were prepared to fit the implementation. Furthermore this section provides insight in the method to structure its workflow (see section 4.2). Section 4.3 deals with the aspects of user interaction in which the user has to complete the free walkable space. It illustrates what user input was used for the process but also what scenarios were used to test the use of the pedestrian network.

4.2 Data preparation

4.2.1 Datasets

The topographical map used for this case study was retrieved from the most detailed topographical map available in The Netherlands: TOP10vector. (kadaster.nl) This file has been developed by the Dutch Topographical Service Cadastre and contains a large scale topographical map of the whole country (i.e. scale 1 : 10.000). The version is dated 2006

As a road dataset, the block dataset from the Dutch National Road Data Bank was used (Nationaal Wegen Bestand), also dated 2006. This dataset was the most suitable one because it contains continuous blocks of roads and streets which is essential for constructing a network dataset. Both datasets were retrieved from the GeoDataBase available at the GeoDesk, a service unit within the Geo-Information Centre of Wageningen UR (University & Research Centre).

4.2.2 Preprocessing

The input datasets were clipped according to the defined study area. In order to do so, the neighborhood dataset of The Netherlands was used to create a clipping mask. The neighborhoods 'Het Zand', 'Binnenstad-Centrum' and 'Binnenstad-Oost' were selected from this dataset since these administrative boundaries include the entire city center and the railway zone.

Additionally, as both datasets contained huge amounts of redundant attributes (the TOP10vector and the road dataset have respectively 16 and 36 attributes), these were deleted before initiating the process. Only the required attributes were kept, i.e. the topographical code.

4.2.3 Workflow

Data handling implies queries, transformations and operations on geodata. To structure these actions in this research and store the geodata in an orderly fashion each feature class produced inherited a hierarchical index indication accordingly to its place in the processing chain (project, component or step).

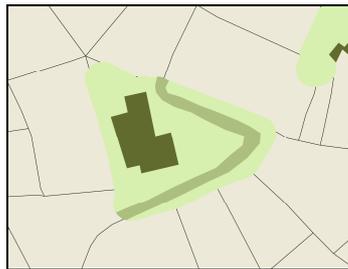
4.3 User interaction

4.3.1 Expert knowledge

In this case study an expert-based selection was made of areas that were completed. Furthermore, a TOP10 building dataset was used to raise costs for buildings that were not present in the topographical map. Figure 11 shows some of the completed areas. Note that all costs were assigned in a subjective fashion and correspond to the costs mentioned in the table below (Table 20)

Table 20) Overview of the assigned accessibility values

Accessibility	WEIGHT	Legend color
Normal / preferred	1	
Not preferred	2	
Dense traffic	5	
No accessibility	99	



a) Marketplace



b) A recently build up area



c) Railway station zone

Figure 11) Areas completed with accessibility values (left) and their accompanying aerial photographs (right) (LUFO 2006)

4.3.2 Scenarios

To be able to calculate routes, some source and destination scenarios are chosen. For all scenarios, the 's-Hertogenbosch Railway Station is set as a source point. Destination points are some points of interest in the city center.

- Scenario 1: From railway station to cathedral, this is one of the main attractions of 's-Hertogenbosch.
- Scenario 2: From railway station to Southern park, the biggest park of 's-Hertogenbosch just outside the city center.
- Scenario 3: From railway station to bastion, a remainder of old fortifications with an outlook over the vast swamp fields adjacent to the city center (Het Bossche Broek).

4.4 Results

4.4.1 Process

In this section the results are discussed from the subprocesses that were initiated to yield a pedestrian network dataset. They are presented by displaying the products that were created during the process. For practical reasons an excision was made out of the whole study area which contains the market place (for location see Figure 10b). The results of the actual route calculation with the pedestrian network dataset will be discussed in the next section (4.4.2).

The free walkable space was derived from all walkable areas. The result shows a free walkable space that has the same proportions as the original market place together with the adjacent road (Figure 12b). Each free walkable space consists of a single part. Note that these spaces have rounded edges.

Assigning the network costs resulted in free walkable space objects that consist of multiple parts, augmented with WEIGHT values that indicate accessibility (Figure 12c). The result shows the internal structure of the corresponding walkable area in a more detailed fashion than the topographical dataset.

Internal path calculation resulted in paths traversing the free walkable space. The result shows that the connection points are end points of the fixed paths that lead to the free walkable space and that all points were mutually connected. It also shows how the paths avoid zones with higher costs (Figure 12d).

The result of deriving the network dataset shows that all calculated paths are represented as an edge in the network dataset (Figure 12e). Examining the nodes learns that the connection points were seen as nodes whereas internal path crossings were not.

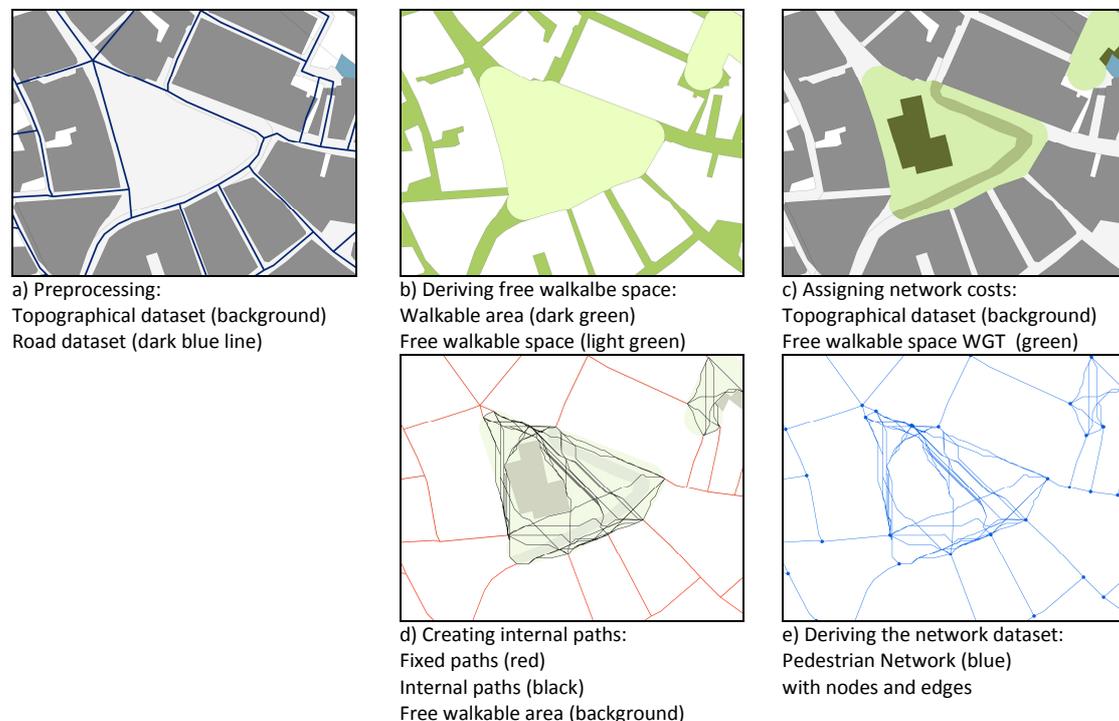


Figure 12) Datasets involved with implementation. Mentioned are the accompanying processes and its result datasets.

4.4.2 Routes

With the pedestrian network dataset routes were calculated for the mentioned scenarios, their results are shown in (Figure 13). Note that all routes start at the railway station. In the first part of the route all scenarios follow the same trajectory. They make use of the available sidewalks and avoid the roads. Furthermore all routes cross the water body (Dieze) at the same spot after which they each go in separate directions.

The first scenario directs towards the cathedral. It crosses the market place where it traverses through free walkable space (Figure 13a). The second scenario aims for the Southern park and also intersects a small corner of the market place (Figure 13b). The third route mainly uses the fixed paths to reach its goal (Figure 13c). The same scenarios are also calculated by means of the pedestrian route planning functionality of Google Maps.

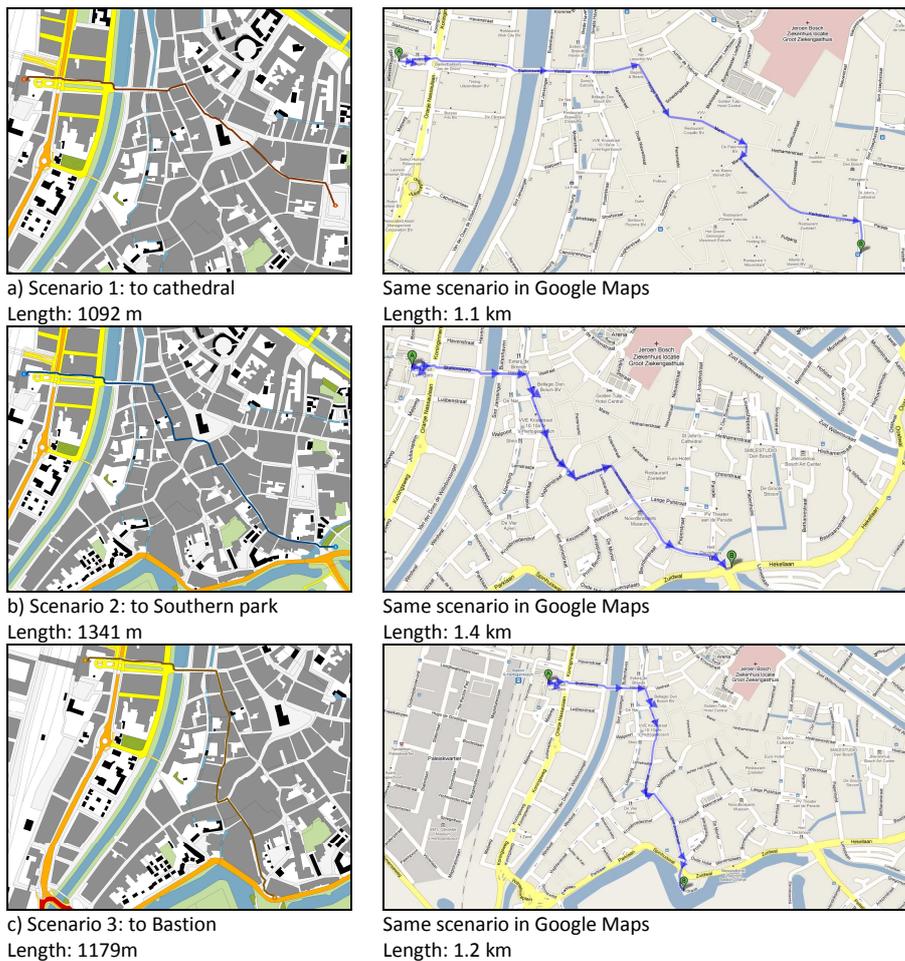


Figure 13) Route results for three scenarios retrieved by implementing the method to a case study in 's-Hertogebosch (left) and by querying the pedestrian route planning functionality in Google Maps (right, source: Google 2010; map data by Tele Atlas 2010).

5 Discussion

In this research a pedestrian network dataset was created. Raster data was used to calculate detailed routes through a free walkable space and an existing road network dataset was used to represent the fixed paths between these spaces. Finally, the pedestrian network dataset was used to calculate optimal routes for three route scenarios. This chapter discusses the results of method, implementation and case study that were involved.

5.1 Method

The method presented a way to combine vector and raster data into a network dataset that is suitable for calculating optimal routes for pedestrians; it integrates the high degree of movement freedom into a network dataset by introducing the concept of free walkable space; offers the possibility to limit that freedom by enriching the free walkable space with expert-knowledge about accessibility and it covers the connection of possible paths in a raster environment with a network of nodes and edges. It brings back the hybrid environment to a manageable vector environment. Besides these points the method also knows some drawbacks, these will be discussed in the following paragraphs.

The derivation of the free walkable space depends on critical factors like the reclassification and the buffer distance. These parameters influence the amount and size of the resulting free walkable space. When for example a smaller value is chosen for the parameter buffer distance, some roads may be included whereas with a higher value some areas might be omitted (Figure 14).

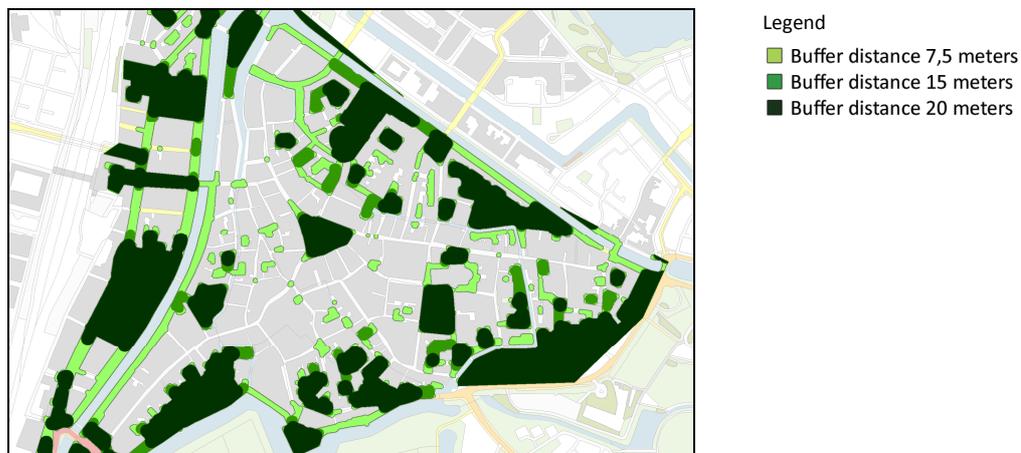


Figure 14) This map indicates that different buffer distances yield different free walkable space outputs.

Assigning costs to the free walkable space is done by a user which is bound to certain rules concerning the cost grid that is created from his input. For example if a zone is physically impenetrable, the user will assign an extreme high cost to it. However, this cannot guarantee that the route will not cross this zone. Particularly when there is no other route option available the least cost path will go straight through this impenetrable zone.

In the process of calculating internal paths not so much the path calculation but the creation of fixed paths brings along some worth noticing issues. Overlaying the fixed paths with the walkable area dataset created while deriving the free walkable space shows some inconsistencies (Figure 15).

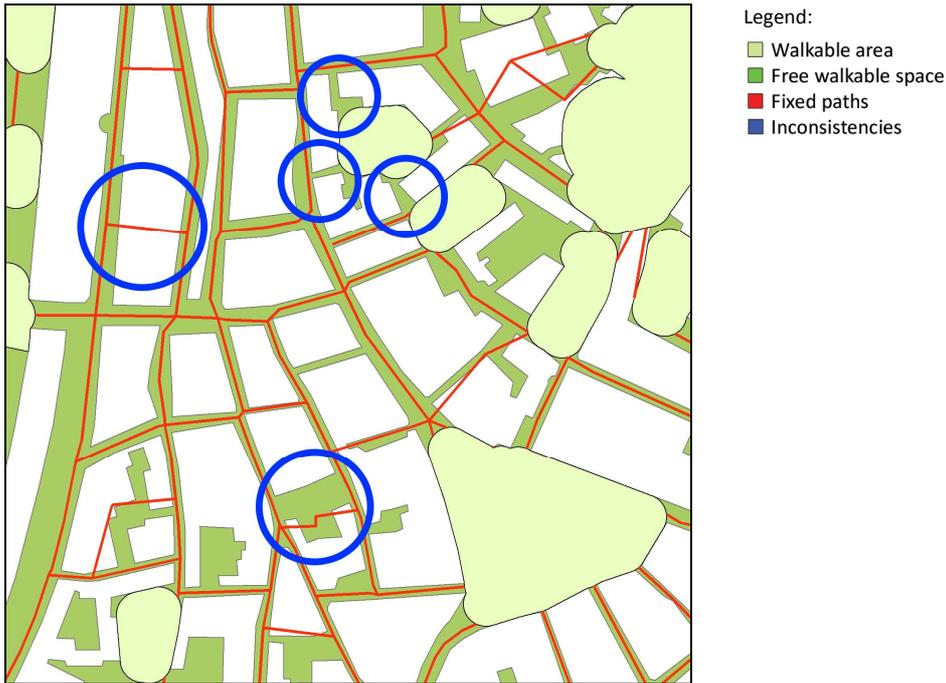


Figure 15) The blue circles indicate the locations where inconsistencies were found between the fixed paths (red) and the walkable areas (dark green).

Furthermore, the presented method to solve raster issues still yields some less than optimal paths (Figure 16).

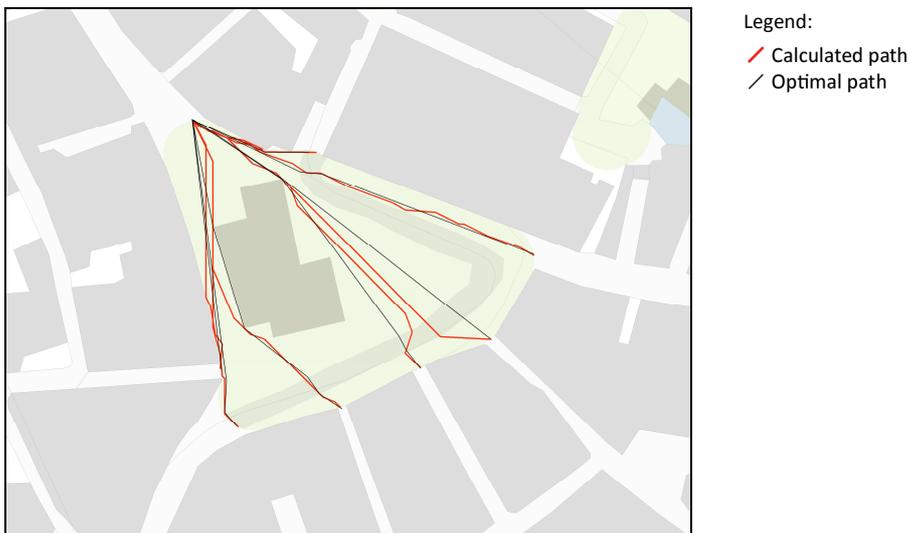


Figure 16) Map that shows that calculated paths still deviate from the direct paths one would intuitively follow.

5.2 Implementation

The implementation translated the method to an applicable script that yields a pedestrian infrastructure dataset. As an input the script requires the topographical dataset and the road dataset. With additional implementation steps a pedestrian network dataset can be created from the output with which optimal routes can be calculated. Implementation focuses on the programming language Python which is used to access tools within ESRI's ArcGIS Desktop and to interconnect them in order to automate the whole process.

One implementation issue lies within the fact that the final stage of the process cannot be automated by Python. Therefore user actions are required which seem unnecessarily laborious.

Furthermore, the presented process is tailored for the input datasets used in the case study. This means that it is not yet suited for use with any other topographical or road input dataset.

5.3 Case study

Applying the implementation to the city centre of 's-Hertogenbosch resulted into a suitable pedestrian network dataset and each route scenario returned a plausible optimal path.

The length of the routes calculated in ArcGIS correspond to those retrieved from the pedestrian routing functionality of Google Maps (Google, 2010). And even show a slight positive difference. The difference is expected to increase when a larger study area is used where there are more free walkable spaces.

6 Conclusion

Main goal of this thesis was to calculate the optimal route for pedestrians in an urban environment based on a hybrid dataset that contains vector network data to represent corridor-like paths and raster to represent free walkable space. The provided method and its implementation succeeded in calculating such routes for several scenarios in a case study.

The first research question required a method to decide which parts of a study area could be labeled as free walkable space and which areas were to be represented as nodes and edges in a graph. To label the parts of a study area as 'free walkable space' or 'fixed paths' the parts that are inaccessible to pedestrians need to be excluded from labeling. From the walkable areas, an area can be labeled as free walkable space when its core still exists after applying a negative buffer on its original proportions. The walkable areas of which even the core vanishes are too narrow to be represented as free walkable space and will therefore be represented as fixed paths.

Research question two set out to connect a raster representation of free walkable space with a network of nodes and edges. The answer was found not in connecting the space itself but in calculating all path possibilities which the space had to offer. Since the connection points were known in which the fixed paths entered the free walkable space, paths could be calculated for each combination of points. This internal structure was then combined with the fixed paths to result into one pedestrian infrastructure.

The determination of costs or benefits for nodes, edges and raster cells is merely subjective. In this research no costs were assigned to nodes and the impedance for route calculation over edges is based on their physical length. Raster cells are assigned costs based on the assumed accessibility of the space they represent. Key is their interrelationship that should be logically related, i.e. high costs for inaccessibility and low costs for accessible space.

The unfavorable paths that result from conventional path calculation based on a Moore neighborhood in the raster environment can be optimized by assessing each source/target scenario separately. As the most optimal path is contained by its corresponding least cost corridor it can be used to redefine the cost grid and recalculate the path. This way a more accurate path is distilled.

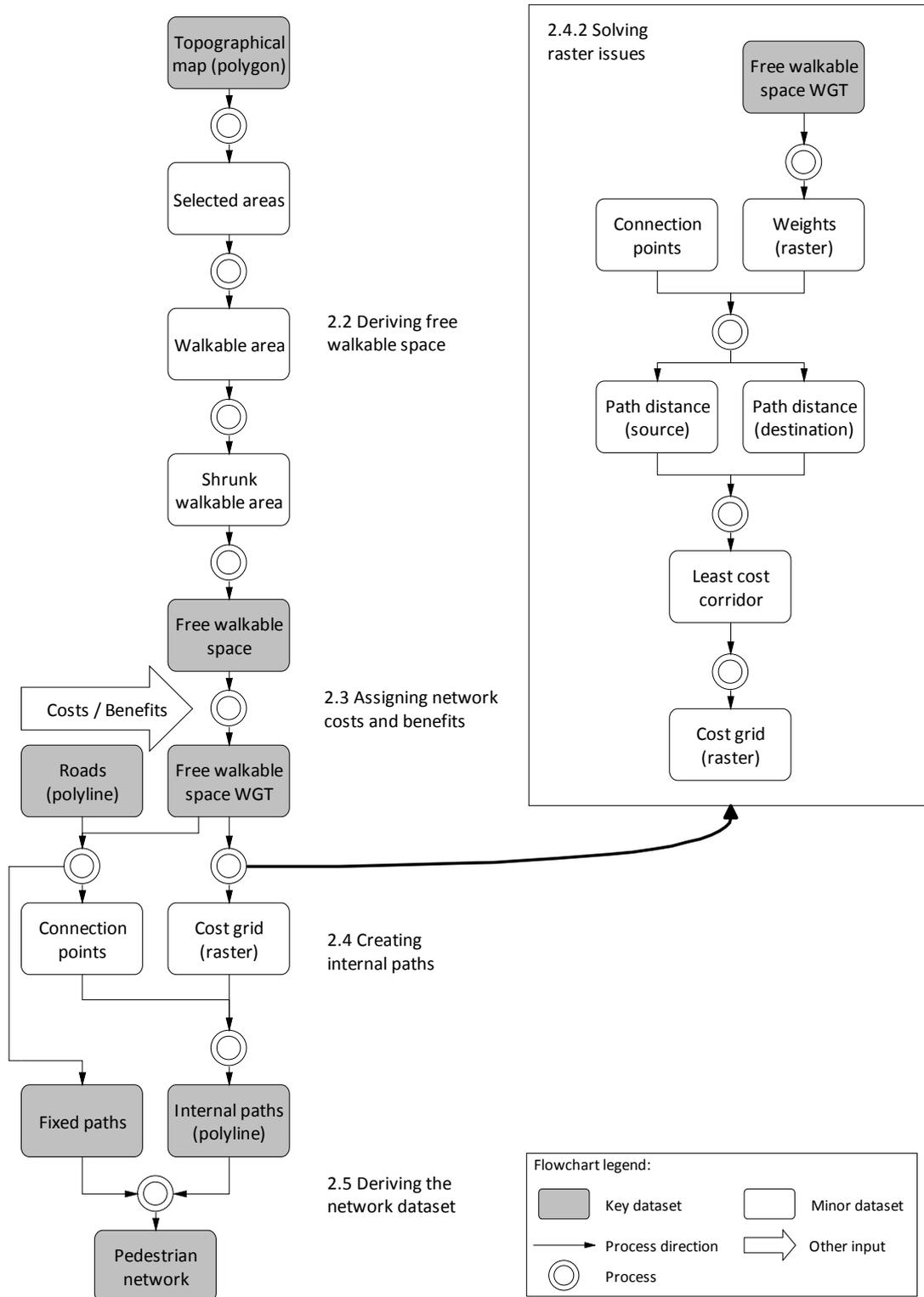
Optimal path calculation in a hybrid environment is done by calculating route possibilities through the raster environment in advance and updating the vector network with the newly calculated paths. With that a network dataset can be constructed on which Dijkstra's shortest path algorithm can be applied to find the optimal route between a start and end point.

Conclusively, the presented method is able to derive a network dataset suitable for pedestrian route planning from the TOP10vector topographical dataset of The Netherlands. Future research should focus on the derivation of fixed walkable paths which in this research shows some inconsistencies with the walkable area. Furthermore a closer look to raster path calculation is required.

7 References

- Corona, B., & Winter, S. (2001a). Datasets for pedestrian navigation. Institute for Geoinformation, Technical University Vienna.
- Corona, B., & Winter, S. (2001b). Guidance of car drivers and pedestrians. Department of Geoinformation, Technical University Vienna.
- Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1, 3.
- Elias, B. (2007). Pedestrian Navigation - Creating a tailored geodatabase for routing. Leibniz University of Hannover.
- Gaisbauer, C., & Frank, A. U. (2008). *Wayfinding Model For Pedestrian Navigation*. Paper presented at the 11th AGILE International Conference on Geographic Information Science, University of Girona (Spain).
- Google (2010). Google Maps, 2010, from maps.google.com
- Lynch, K. (1960). *The image of the city*. Cambridge: Massachusetts Institute of Technology.
- Montello, D. R. (2005). *The Cambridge handbook of visuospatial thinking*. Cambridge: Cambridge University Press.
- Rehrl, K., Leitinger, S., & Gartner, G. (2007). *The SemWay Project - Towards Semantic Navigation Systems*. Paper presented at the 4th International Symposium on LBS & TeleCartography, Hong Kong (China).
- Stark, A., Riebeck, M., & Kawalek, J. (2007, 6-8 September 2007). *How to Design an Advanced Pedestrian Navigation System: Field Trial Results*. Paper presented at the IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Dortmund (Germany).
- Walter, V., Kada, M., & Chen, H. (2006). *Shortest path analyses in raster maps for pedestrian navigation in location based systems*. Paper presented at the International symposium on 'Geospatial Databases for Sustainable Development', Goa (India).
- Wiener, J., Büchner, S. J., & Hölscher, C. (2009). Taxonomy of Human Wayfinding Tasks: A Knowledge-Based Approach. *Spatial Cognition & Computation*, 9(2), 13.

Annex A: Detailed process overview



Annex B: Python script

```
# Documentation
"JB_thesis_script.py"

# Settings
print "\nExecuting..."
# Import system modules
import sys, string, os, arcgisscripting, time, math, decimal

time0 = time.time()

# Create the Geoprocessor object
gp = arcgisscripting.create(9.3)
gp.SetProduct("ArcInfo")
gp.CheckOutExtension("spatial")

gp.OverwriteOutput = 1

# Set the Geoprocessing environment...
gp.XYResolution = "0.001 Unknown"
gp.scratchWorkspace = "D:\\JB_Thesis\\Data\\Application\\IA_MSc_Thesis_Jan_Bakermans_Temporary_GDB.gdb"
gp.MTolerance = ""
gp.randomGenerator = "0 ACM599"
gp.outputCoordinateSystem =
  "PROJCS[\"RD_New\",GEOGCS[\"GCS_Amersfoort\",DATUM[\"D_Amersfoort\",SPHEROID[\"Bessel_1841\",6377397.155,299.1528
  128]],PRIMEM[\"Greenwich\",0.0],UNIT[\"Degree\",0.0174532925199433]],PROJECTION[\"Double_Stereographic\"],PARAM
  ETER[\"False_Easting\",155000.0],PARAMETER[\"False_Northing\",463000.0],PARAMETER[\"Central_Meridian\",5.387638
  88888889],PARAMETER[\"Scale_Factor\",0.9999079],PARAMETER[\"Latitude_Of_Origin\",52.15616055555555],UNIT[\"Met
  er\",1.0]]"

gp.snapRaster = ""

gp.outputZFlag = "Disabled"
gp.qualifiedFieldNames = "false"

gp.extent = "DEFAULT"

gp.XYTolerance = "0.01 Meters"
gp.outputZValue = ""
gp.outputMFlag = "Disabled"
gp.geographicTransformations = ""
gp.ZResolution = ""
gp.workspace = "D:\\JB_Thesis\\Data\\Application"
gp.MResolution = ""
gp.ZTolerance = ""

# GDB settings
Component = gp.workspace + "\\IA_MSc_Thesis_Jan_Bakermans_Component_GDB.gdb\\"
Project = gp.workspace + "\\IA_MSc_Thesis_Jan_Bakermans_Project_GDB.gdb\\"
Temporary = gp.workspace + "\\IA_MSc_Thesis_Jan_Bakermans_Temporary_GDB.gdb\\"
Raster = gp.workspace + "\\raster\\"

print "Workspace is: " + gp.workspace

# Preprocessing

def DeriveFreeWalkableSpace ():
    # Free walkable space -----
    '''-----
    Input: Topographical map

    ? > walkable areas
    s > Shrunk walkable areas
    o > Free walkable space

    Output: Free walkable space
           Free walkable space (editable)
    -----'''

    prefix = "a03_"

    # Select walkable areas -----
    # TOOL 1: Make selection layer
    in_features = Project + "ps_TOP10_polygons"
    out_layer = Temporary + prefix + "walk_layer"
    gp.MakeFeatureLayer_management (in_features, out_layer)

    # TOOL 2: Select walkable areas
    in_layer = out_layer
    selection_type = "ADD_TO_SELECTION"
    where_clause = "\"TDN_CODE\" = '03103' OR \"TDN_CODE\" = '03143' OR \"TDN_CODE\" = '03243' OR \"TDN_CODE\" =
    '03343' OR \"TDN_CODE\" = '03403' OR \"TDN_CODE\" = '03413' OR \"TDN_CODE\" = '03433' OR \"TDN_CODE\" =
    '03473' OR \"TDN_CODE\" = '03533' OR \"TDN_CODE\" = '03603' OR \"TDN_CODE\" = '03903' OR \"TDN_CODE\" =
    '05023' OR \"TDN_CODE\" = '05213' OR \"TDN_CODE\" = '05263' OR \"TDN_CODE\" = '06513'"
    gp.SelectLayerByAttribute_management (in_layer, selection_type, where_clause)

    # TOOL 3: Copy selection to walkable area feature class
    out_feature_class = Temporary + prefix + "walk_select"
    gp.CopyFeatures_management (in_layer, out_feature_class)

    # TOOL 4: Dissolve all walkable area
    in_features = out_feature_class
    out_feature_class = Temporary + prefix + "dissolution"
    print ">>> Dissolving walkable area..."
    gp.Dissolve_management (in_features, out_feature_class)

    # TOOL 5: Shrink walkable area
```

```

in_features = out_feature_class
out_feature_class = Temporary + prefix + "minbuffered"
buffer_distance = -15
print ">>> Applying minbuffer..."
gp.Buffer_analysis (in_features, out_feature_class, str(buffer_distance), "", "", "ALL")

# TOOL 6: Expand walkable area
in_features = out_feature_class
out_feature_class = Temporary + prefix + "plusbuffered"
buffer_distance *= -1
print ">>> Applying plusbuffer..."
gp.Buffer_analysis (in_features, out_feature_class, str(buffer_distance), "", "", "ALL")

# Clip free walkable area to fit study area
# in_features = Temporary + prefix + "walkable_area"
in_features = out_feature_class
clip_features = Project + "ps_clipping_mask_study_area"
out_feature_class = Temporary + prefix + "plusbuffered_sa"
print ">>> Clipping study area..."
gp.Clip_analysis (in_features, clip_features, out_feature_class)

# TOOL 7: Multi to single part
in_features = out_feature_class
out_feature_class = Temporary + prefix + "free_walkable_area"
print ">>> Multi to single part..."
gp.MultipartToSinglepart_management (in_features, out_feature_class)

# TOOL 8 and TOOL 9: Add fields and calculate them to enable weight editing
in_table = out_feature_class
field_name_1 = "Area_ID"
field_name_2 = "WEIGHT"
field_type = "DOUBLE"
field_precision = 5
field_scale = 0
gp.AddField_management (in_table, field_name_1, field_type, field_precision, field_scale)
gp.AddField_management (in_table, field_name_2, field_type, field_precision, field_scale)
gp.CalculateField_management (in_table, field_name_1, "[OBJECTID]")
gp.CalculateField_management (in_table, field_name_2, 1)

# TOOL 10: Create back up to enable weight editing and preserve original
in_table = in_table
out_feature_class = Temporary + prefix + "free_walkable_area_BAK"
gp.CopyFeatures_management (in_table, out_feature_class)

def Internalpaths ():
# Internal paths -----
'''-----
Input: Free walkable space (editable)
Free walkable space
Roads (polyline)

Free walkable space + Roads (polyline)
s > Connection points
s > Fixed paths

Free walkable space (editable)
s > Free walkable space (raster)

Connection points + Free walkable space (raster)
s > Calculated paths

Calculated paths
s > Path cost table (table)

Connection points
s > Connections

Connections + Path cost table (table)
o > Internal paths (polyline)

Output: Internal paths (polyline)
Fixed paths
-----'''

prefix = "a04_"

# Fixed paths -----
# TOOL 11: Erase walkable space from roads
in_features = Project + "ps_TOP10_roads_block"
erase_features = Temporary + "a03_free_walkable_area"
out_feature_class = Temporary + prefix + "fixed_paths"
gp.Erase_analysis (in_features, erase_features, out_feature_class)

# Connection points -----
# TOOL 12: Find end points of fixed paths
in_features = out_feature_class
out_feature_class = Temporary + prefix + "path_ends"
point_location = "BOTH_ENDS"
gp.FeatureVerticesToPoints_management (in_features, out_feature_class, point_location)

# TOOL 13: Create layer to collect end points in
in_feature = out_feature_class
out_layer = Temporary + prefix + "path_ends_layer"
gp.MakeFeatureLayer_management (in_feature, out_layer)

# TOOL 14: Select end points that are <adjacent to> free walkable space
in_layer = out_layer
overlap_type = "WITHIN_A_DISTANCE"
select_features = Temporary + "a03_free_walkable_area"
search_distance = 0.015

```

```

gp.SelectLayerByLocation_management (in_layer, overlap_type, select_features, search_distance)

# TOOL 14: Add end points that are <within> free walkable space to the selection
in_layer = out_layer
overlap_type = "WITHIN"
select_features = Temporary + "a03_free_walkable_area"
selection_type = "ADD_TO_SELECTION"
gp.SelectLayerByLocation_management (in_layer, overlap_type, select_features, "", selection_type)

# TOOL 15: Copy selection to connection point feature class
out_feature = Temporary + prefix + "connection_points"
gp.CopyFeatures_management (in_layer, out_feature)

# Remove duplicate connection points
#rows = gp.UpdateCursor(out_feature)
#cur = rows.Next()
#a = []

#while cur:
#   if a.count([cur.shape.FirstPoint.x, cur.shape.FirstPoint.y, cur.shape.LastPoint.x, cur.shape.LastPoint.y,
#               cur.shape.length]) > 0:
#       rows.DeleteRow(cur)
#       cur = rows.Next()
#   else:
#       a.append([cur.shape.FirstPoint.x, cur.shape.FirstPoint.y, cur.shape.LastPoint.x, cur.shape.LastPoint.y,
#                cur.shape.length])
#       cur = rows.Next()

# Add area ID field
field_name = "Area_ID"
field_type = "SHORT"
gp.AddField_management (out_feature, field_name, field_type)

# Add point ID field
field_name = "Point_ID"
field_type = "SHORT"
gp.AddField_management (out_feature, field_name, field_type)

in_feature = Temporary + "a03_free_walkable_area_BAK"
rows = gp.updateCursor(in_feature)
cur = rows.Next()
areapointlist = {}
del_list = []

# Make point layer
in_feature = Temporary + prefix + "connection_points"
out_layer2 = Temporary + prefix + "point_layer"
gp.MakeFeatureLayer_management (in_feature, out_layer2)

while cur:
    # Make selection layer from area
    print "Area_ID:", cur.OBJECTID
    in_feature = Temporary + "a03_free_walkable_area_BAK"
    out_layer1 = Temporary + prefix + "frame_layer"
    where_clause = "\"OBJECTID\" = " + str(cur.OBJECTID)
    gp.MakeFeatureLayer_management (in_feature, out_layer1, where_clause)

    # Select points in point layer that belong to one area
    in_layer = out_layer2
    overlap_type = "WITHIN"
    select_features = out_layer1
    search_distance = 0.1
    gp.SelectLayerByLocation_management (in_layer, overlap_type, select_features, search_distance)

    # Calculate area ID for feature class
    gp.CalculateField_management (in_layer, "Area_ID", cur.OBJECTID)

    # For each area, store point ID and coordinates in a dictionary for future use.
    rows2 = gp.UpdateCursor(in_layer)
    cur2 = rows2.Next()
    areapointlist[cur.OBJECTID] = {}
    i = 1

    while cur2:
        # Calculate point ID
        cur2.SetValue("Point_ID", i)
        rows2.UpdateRow(cur2)
        print "> Point_ID:", i
        cur2 = rows2.Next()
        i += 1

    cur = rows.Next()

# Create Free walkable space (raster)-----
# Create raster from area
in_feature = Temporary + "a03_free_walkable_area"
value_field = "WEIGHT"
cell_size = 1
out_raster_dataset = Raster + prefix + "_areas"
gp.PolygonToRaster_conversion (in_feature, value_field, out_raster_dataset, "", "", cell_size)

# Create path feature class
out_path = Temporary
out_name = prefix + "internal_paths"
geometry_type = "POLYLINE"
gp.CreateFeatureClass_management (out_path, out_name, geometry_type)

# Set environment settings
gp.snapRaster = out_raster_dataset
gp.cellSize = 1
gp.mask = out_raster_dataset

```

```

# Set point dataset
point_dataset = Temporary + prefix + "connection_points"

# Calculate paths -----
# Buffer all points to cover more cells when rasterizing
in_features = point_dataset
out_feature_class = in_features + "_buff"
buffer_distance = 1
gp.Buffer_analysis (in_features, out_feature_class, buffer_distance)

# Add Range field
field_name = "Range"
field_type = "SHORT"
gp.AddField_management (out_feature_class, field_name, field_type)

rows = gp.searchCursor(point_dataset + "_buff")
cur = rows.Next()
area_list = []
while cur:
    aid = cur.Area_ID
    try:
        area_list.index(aid)
    except:
        area_list.append(cur.Area_ID)
    cur = rows.Next()

for aid in area_list:
    in_table = point_dataset + "_buff"
    out_view = Temporary + prefix + "temp_point_view"
    where_clause = "\"Area_ID\" = " + str(aid)
    gp.MakeTableView_management (in_table, out_view, where_clause)

    in_table = Temporary + prefix + "temp_point_view"
    out_table = Temporary + prefix + "temp_point_stat"
    statistics_fields = "Point_ID MAX"
    gp.Statistics_analysis (in_table, out_table, statistics_fields)
    rows = gp.SearchCursor(out_table)
    cur = rows.Next()
    range_max = int(cur.MAX_Point_ID)

    gp.CalculateField_management (out_view, "Range", range_max)

index = 1
gp.snapRaster = Raster + prefix + "_areas"
gp.extent = "148182.934 410391.759 150146.178 412199.376"
gp.cellSize = "1"
gp.mask = Raster + prefix + "_areas"

# Calculate range
in_table = point_dataset
out_table = Temporary + prefix + "MAX_Point_ID"
statistics_fields = "Point_ID MAX"
gp.Statistics_analysis (in_table, out_table, statistics_fields)

rows = gp.SearchCursor(out_table)
cur = rows.Next()

range_max = int(cur.MAX_Point_ID)

for From_ID in range(1, range_max):
    range_to = From_ID + 1
    for To_ID in range(range_to, range_max + 1):
        print "From: " + str(From_ID) + " To: " + str(To_ID)
        # <<< L O O P S T A R T S H E R E >>>
        in_features = Temporary + prefix + "connection_points_buff"
        out_feature_class = Temporary + prefix + "sources"
        where_clause = "\"Point_ID\" = " + str(From_ID) + " AND \"Range\" <> \"Point_ID\""
        gp.Select_analysis (in_features, out_feature_class, where_clause)

        in_features = Temporary + prefix + "connection_points_buff"
        out_feature_class = Temporary + prefix + "destinations"
        where_clause = "\"Point_ID\" = " + str(To_ID)
        gp.Select_analysis (in_features, out_feature_class, where_clause)

        ...

        in_features = Temporary + prefix + "dest"
        erase_features = Temporary + prefix + "sources"
        out_feature_class = Temporary + prefix + "destinations"
        gp.Erase_analysis (in_features, erase_features, out_feature_class)
        ...

        in_source_data = Temporary + prefix + "sources"
        out_distance_raster = Raster + prefix + "dist_s"
        in_cost_raster = Raster + prefix + "_areas"
        gp.PathDistance_sa (in_source_data, out_distance_raster, in_cost_raster)

        in_source_data = Temporary + prefix + "destinations"
        out_distance_raster = Raster + prefix + "dist_d"
        in_cost_raster = Raster + prefix + "_areas"
        gp.PathDistance_sa (in_source_data, out_distance_raster, in_cost_raster)

        in_distance_raster1 = Raster + prefix + "dist_s"
        in_distance_raster2 = Raster + prefix + "dist_d"
        out_raster = Raster + prefix + "corr"
        gp.Corridor_sa (in_distance_raster1, in_distance_raster2, out_raster)

        in_zone_data = Temporary + "a03_free_walkable_area_BAK"
        zone_field = "OBJECTID"
        in_value_raster = Raster + prefix + "corr"
        out_raster = Raster + prefix + "corr_min"
        statistics_type = "MINIMUM"
        ignore_nodata = "DATA"

```

```

gp.ZonalStatistics_sa (in_zone_data, zone_field, in_value_raster, out_raster, statistics_type,
ignore_nodata)

expression_string = "CON(" + Raster + prefix + "corr <= (" + Raster + prefix + "corr_min + 0.5), 1, 0)"
out_raster = Raster + prefix + "corr_1"
gp.SingleOutputMapAlgebra_sa (expression_string, out_raster)

in_conditional_raster = Raster + prefix + "corr_1"
in_false_raster_or_constant = "1"
out_raster = Raster + prefix + "corrs"
where_clause = "\"VALUE\" = 0"
gp.SetNull_sa (in_conditional_raster, in_false_raster_or_constant, out_raster, where_clause)

tempmask = gp.mask
gp.mask = Raster + prefix + "corrs"

expression_string = "FOCALSUM (" + Raster + prefix + "corrs, CIRCLE, 3)"
out_raster = Raster + prefix + "ma1"
gp.SingleOutputMapAlgebra_sa (expression_string, out_raster)

expression_string = "FOCALMAX (" + Raster + prefix + "ma1, rectangle, 5, 5)"
out_raster = Raster + prefix + "ma2"
gp.SingleOutputMapAlgebra_sa (expression_string, out_raster)

in_raster = Raster + prefix + "ma2"
reclass_field = "VALUE"
remap = "0 24 1:24 100 NoData"
out_raster = Raster + prefix + "reclass"
gp.Reclassify_sa (in_raster, reclass_field, remap, out_raster)

in_source_data = Raster + prefix + "reclass"
out_distance_raster = Raster + prefix + "euclidis"
gp.EucDistance_sa (in_source_data, out_distance_raster)

in_raster_or_constant1 = Raster + prefix + "euclidis"
in_raster_or_constant2 = 1
out_raster = in_raster_or_constant1 + "_1"
gp.Plus_sa (in_raster_or_constant1, in_raster_or_constant2, out_raster)

in_source_data = Temporary + prefix + "sources"
in_cost_raster = Raster + prefix + "euclidis_1"
out_backlink_raster = Raster + prefix + "bcklnk"
out_distance_raster = Raster + prefix + "dist"
gp.CostBackLink_sa (in_source_data, in_cost_raster, out_backlink_raster, "", out_distance_raster)

in_destination_data = Temporary + prefix + "destinations"
in_cost_distance_raster = Raster + prefix + "dist"
in_cost_backlink_raster = Raster + prefix + "bcklnk"
out_raster = Raster + prefix + "paths"
path_type = "EACH_ZONE"
destination_field = "Area_ID"
gp.CostPath_sa (in_destination_data, in_cost_distance_raster, in_cost_backlink_raster, out_raster,
path_type, destination_field)

# Create path based on minimum path cost from statistics table
in_raster = out_raster
out_raster = Raster + prefix + "paths_1"
#where_clause = "\"PATHCOST\" < " + str(minpathcost + 0.1)
constant = 1
gp.Con_sa (in_raster, constant, out_raster)

# Create polyline from path
in_raster = out_raster
out_polyline_features = Temporary + prefix + "temp_path"
simplify = "SIMPLIFY"
#raster_field = "VALUE"
gp.RasterToPolyline_conversion (in_raster, out_polyline_features, "", "", simplify) #raster_field)

# Explode path into points
in_features = out_polyline_features
out_feature_class = Temporary + prefix + "temp_path_points"
gp.FeatureVerticesToPoints_management (in_features, out_feature_class)

# Generate near table
in_features = out_feature_class
near_features = Temporary + prefix + "connection_points"
out_table = Temporary + prefix + "near_table"
search_radius = "1"
location = "LOCATION"
gp.GenerateNearTable_analysis (in_features, near_features, out_table, search_radius, location)

in_table = out_table
out_table = Temporary + prefix + "nearby_connection_points"
where_clause = "\"NEAR_DIST\" < " + search_radius
gp.TableSelect_analysis (in_table, out_table, where_clause)

in_feature = Temporary + prefix + "temp_path"

rows3 = gp.searchCursor(in_feature)
cur3 = rows3.Next()

while cur3:
    print cur3.OBJECTID
    # Alter begin and end point
    in_feature = Temporary + prefix + "temp_path_points"
    outDesc = gp.describe(in_feature)
    shapefield = outDesc.ShapeFieldName

    point = gp.createobject("point")
    pntarray = gp.createobject("Array")
    partarray = gp.createobject("Array")
    where_clause = "\"ORIG_FID\" = " + str(cur3.OBJECTID)

```

```

rows4 = gp.searchCursor(in_feature, where_clause)
cur4 = rows4.Next()
rows4_next = gp.searchCursor(in_feature, where_clause)
cur4_next = rows4_next.Next()
cur4_next = rows4_next.Next()
geometry = cur4.shape

# Alter begin point
in_table = Temporary + prefix + "nearby_connection_points"
where_clause = "\"IN_FID\" = " + str(cur4.OBJECTID)
rows5 = gp.searchCursor(in_table, where_clause)
cur5 = rows5.next()

point.id = 1
point.x = cur5.NEAR_X
point.y = cur5.NEAR_Y
pntarray.add(point)

# Add mid section points
i = 2
cur4 = rows4.Next()
cur4_next = rows4_next.Next()
while cur4 and cur4_next:
    geometry = cur4.shape
    point.id = i
    point.x = geometry.centroid.x
    point.y = geometry.centroid.y
    pntarray.add(point)
    i += 1
    cur4 = rows4.Next()
    cur4_next = rows4_next.Next()

# Alter end point
# geometry = cur4.shape
in_table = Temporary + prefix + "nearby_connection_points"
where_clause = "\"IN_FID\" = " + str(cur4.OBJECTID)
rows5 = gp.searchCursor(in_table, where_clause)
cur5 = rows5.next()

point.id = i
point.x = cur5.NEAR_X
point.y = cur5.NEAR_Y
pntarray.add(point)
partarray.add(pntarray)

# Insert final path in 'internal_paths'
in_feature = Temporary + prefix + "internal_paths"
rows5 = gp.insertCursor(in_feature)
cur5 = rows5.NewRow()

cur5.SetValue(shapefield, partarray)
print "adding: ", i
rows5.insertrow(cur5)

partarray.removeall()
pntarray.removeall()

cur3 = rows3.Next()

gp.mask = tempmask
del rows3, rows4, rows5, cur3, cur4, cur5

def PedestrianInfrastructure():
    # Pedestrian Network
    '''-----
    Input: Internal paths (polyline)
           Fixed paths

    Internal paths (poyline) + Fixed paths
    o > Pedestrian network

    Output: Pedestrian network
    -----'''

    prefix = "a05_"

    in_data = Temporary + "a04_fixed_paths"
    out_data = Temporary + prefix + "pedestrian_infrastructure"
    gp.Copy_management (in_data, out_data)

    inputs = Temporary + "a04_internal_paths"
    target = Temporary + prefix + "pedestrian_infrastructure"
    schema_type = "NO_TEST"
    gp.Append_management (inputs, target, schema_type)

'''Function section'''
DeriveFreeWalkableSpace()
Internalpaths()
PedestrianInfrastructure()

# Close down script
time1 = time.time()
elapsed = str(int((time1 - time0)/60)) + "min " + str(int((((time1 - time0)/60) - int((time1 - time0)/60))*60)) +
"sec "
print "Script successfully executed in " + elapsed
del gp, sys, string, os, arcgisscripting, time, math, decimal

```