Bauhaus-Universität Weimar
Informatik in der Architektur | InfAR

Arbeitspapiere
Working Papers

Nr. 16, January 2017

Pedestrian movement graph analysis

Ekaterina Fuchkina

ISSN 2191-2416

Bauhaus-Universität Weimar, Professur Informatik in der Architektur, Belvederer Allee 1, 99421 Weimar
Fon: +49/3643/584201, caad@architektur.uni-weimar.de, http://infar.architektur.uni-weimar.de

Ekaterina Fuchkina
Pedestrian movement graph analysis

Supervisors:
Vertr.-Prof. Dr.-Ing. Sven Schneider
Ing. Arch. Martin Bielik
M.Arch. Abdulmalik Abdulmawla

Weimar 2016
Arbeitspapiere (Working Papers) Informatik in der Architektur, Nr. 16
Herausgegeben von Prof. Dr. Dirk Donath und Jun.-Prof. Dr. Reinhard König
ISSN 2191-2416

Bauhaus-Universität Weimar
Professur Informatik in der Architektur und Junior-Professur Computational Architecture
Belvederer Allee 1, 99425 Weimar
http://infar.architektur.uni-weimar.de
www.uni-weimar.de/computational-architecture

Pedestrian movement graph analysis

Ekaterina Fuchkina
ekaterina.fuchkina@uni-weimar.de
Professur Informatik in der Architektur
Fakultät Architektur, Bauhaus-Universität Weimar, Belvederer Allee 1, 99421 Weimar, Germany

Abstract

Development of sustainable urban environments assumes processing of large amount of data from various sources. It could be field study observations, results of simulations or information provided by modeling. This paper focuses on processing modeled data of pedestrian movement based on existed axial maps of particular environment. Introduced component allows further analysis by calculation of set of metrics based on inverted graph, which is built from given paths.

Keywords: Space Syntax, Computational Design, Pedestrian movement

1. Component description

The "CityGraph" component is created for Grasshopper™ graphical algorithm editor in Rhino 5 environment. Component is a set of tools for analyzing a pedestrian movement. The main approach is based on representation of the city network as inversion of a simple axial map (Dawes 1926) and calculation of graph's various metrics, such as: Betweenness Centrality, Closeness Centrality, Gravity, Weighted Betweenness, Degree Centrality (Kalvo 2015).

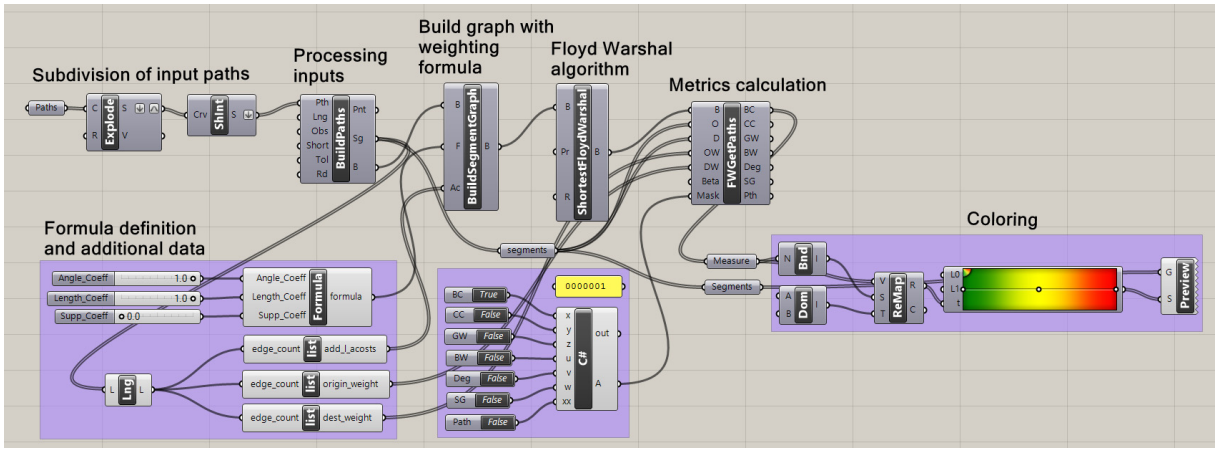


Figure 1: Overview of component setup

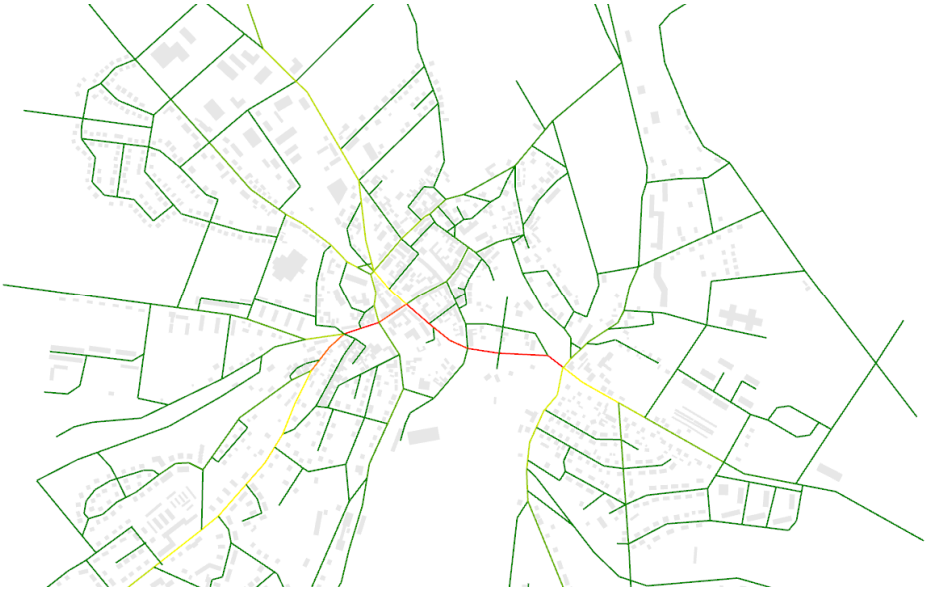


Figure 2: Graphical output of betweenness centrality calculation of city network

2. Conceptual description of algorithm. Motivations.

Due to huge amount of incoming graphical information about existing roads, sidewalks and footpaths it is hard to check if the data is correct and full. For example, cases on Figure 3 can arise.

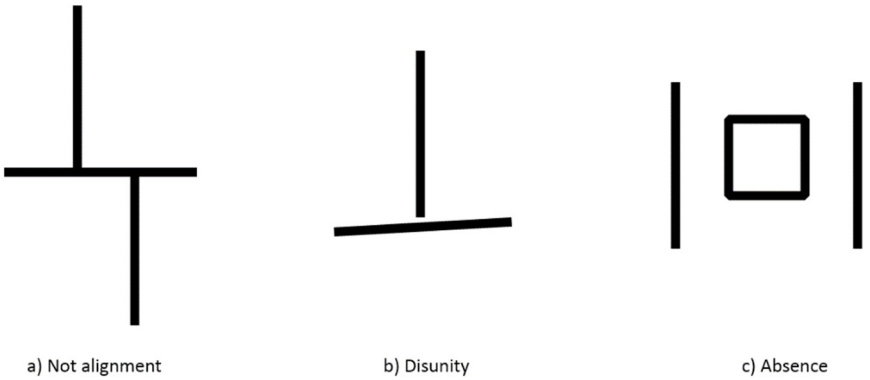
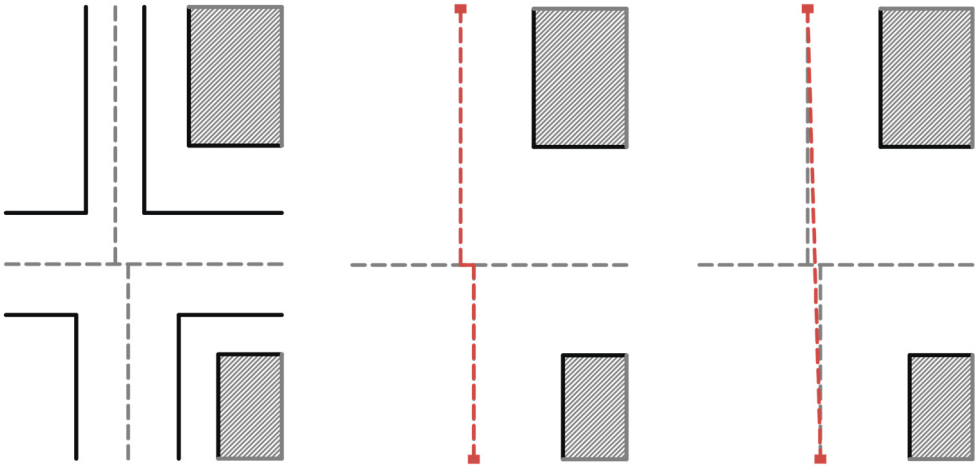


Figure 3: Problems with input data

2.1. Problem 1. Handle with incorrect and incomplete data

The first solution to problems of Figure 3 is - if some points of origin/destination are given and if direct line between them doesn't not intersect any obstacle, then this line can be named as solution for a shortest path task for these points, even if originally there is no such path information. Figure 4c shows that such solution provides more correct result (in terms of "the shortest" measure) and close to more realistic movement than Figure 4b.



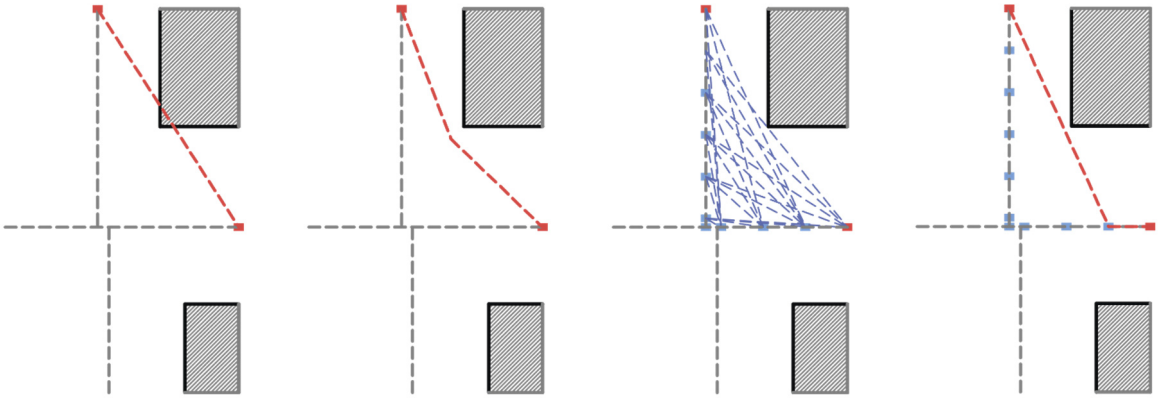
(a) Situation whith center lines, which are used for calculations,

(b) Red polyline is the shortest path between points based on given geometry

(c) Red line is the shortest path between points based on generated geometry

Figure 4: Problem of shortest path calculation with incorrect data

However, if in-between given points there are obstacles (Figure 5a) - other approach should be used, which is more general. The idea is to generate sample points (divide current path segments) by some distance and generate connection lines between them and existed ones (existed are points of end/start of path segments). This will give approximation of possible short movements through given space (or visibility from each point). Figure 5c shows new generated paths and Figure 5d shows shortest path between given points based on generated geometry. Hence, the denser sample points the more accurate the solution. This approach increases amount of path geometry data to process and to reduce it - radius parameter (Rd) is used (see section 3.1). Within the specified radius, points, to create new shortest (direct lines) paths, will be taken.



(a) Direct connection cannot be a shortest path because of obstacle

(b) Ideal solution for a shortest path task

(c) Generation of sample points (blue ones) and connection paths between them

(d) Shortest path solution based on generated geometry

Figure 5: General shortest path solution approach

2.2. Problem 2. Handle with unrealistic movements

On Figure 6a there are a two points to find a shortest path in-between and the right solution – the path is the shortest, but it can be assumed, that in a real situation, especially in unknown city, it is hard to follow such way, more easier to walk and orient on straight lines. For this purpose the definition of angular measure is introduced. The angular measure reflects how much the path is straight (how large, in terms of angles, changing of directions was along the path) and is combined with geodesic measure with some coefficient to control the influence. Figure 6b and Figure 6c show the solutions with different coefficients.

Figure 7 shows, that shortest path, where angular measure has highest coefficient, has geodesic length more than other solutions. More details on angular measure calculation see in section 2.4.

(a) Shortest path by geodesic solutinon with angular influence (0)

(b) Shortest path with half-angular influence (0.5)

(c) Shortest path with angular influence (1)

Figure 6: Difference between geodesic shortest path and path with angular measure influence

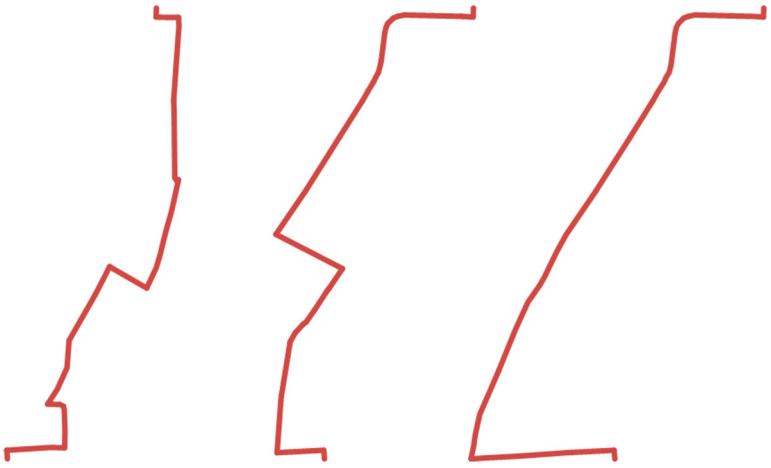


Figure 7: Shortest path with angular influence

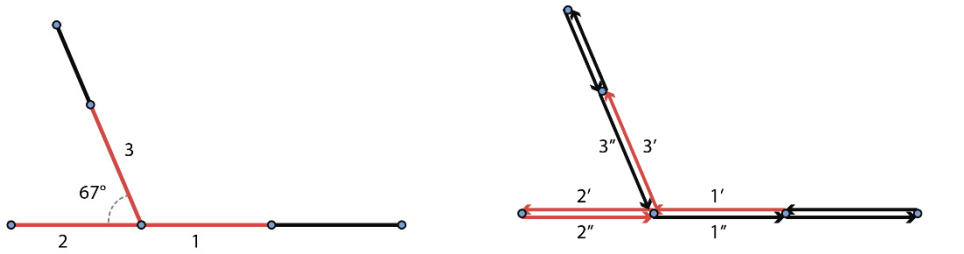
2.3. Problem 3. Handle with angular measure

In case, if for calculation of shortest path only angular measure is used (just sum of angles), it can be a situation, when neglecting a geodesic length cause a problem. For example, if it is needed to find the shortest path between segments 1 and 3 on Figure 8a using only angles as a measure, then it happens that a solution will the sequence 1-2-3 and not the expected 1-3, because the angular distance between segments 1 and 3 is 113°. The sum of angular distance between segments 1, 2 and 3 is 67°, which is less than in the first case.

$$dist(1,2) = 0°$$
$$dist(2,3) = 67°$$

In a real situation to repeat path 1-2-3 as on 8a it is needed to turn back on segment 2 to reach segment 3, which is 360and, hence, it is incorrect to take 1-2-3 path as the shortest one. To avoid this case - representation of directions of movement is included as shown on 8b, where path exists only between two segments if end and start (with arrow) points are coincident. Here the path 1′-3′ is the shortest. (Because the similar path as in the first case will be the sequence 1′-2′-2″-3′).

This approach needs the step when solutions are merged/reduced, because, for example, between 1 and 2 segments exist one shortest path (1′-2′), and also solutions for other directions of same edges, like between 1′ and 2″ (shortest path will 1′-2′-2″), and this should be removed.



(a) Shortest path between 1-3

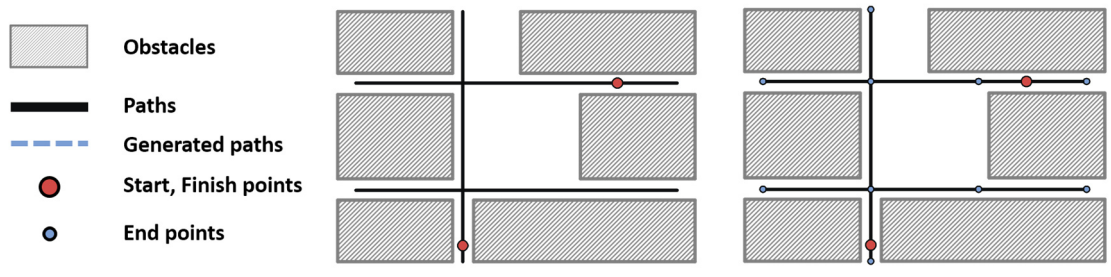
(b) Shortest path between 1-3 with taking into account directions

Figure 8: Directions influence

2.4. Algorithm illustration

1) Input paths and obstacles information. Figure 9a

2) Finding intersections of paths. (Only for illustration, because real component waits for paths already intersected to each other). Depending on parameter in component, segments can be further subdivided. Figure 9b shows that segments were subdivided by length of 1 unit.



(a) Input data

(b) Subdivision of segments

Figure 9: Algorithm illustration

3) Generating additional paths if corresponding parameter is switched on. Figure 10a shows that 4 paths were added.

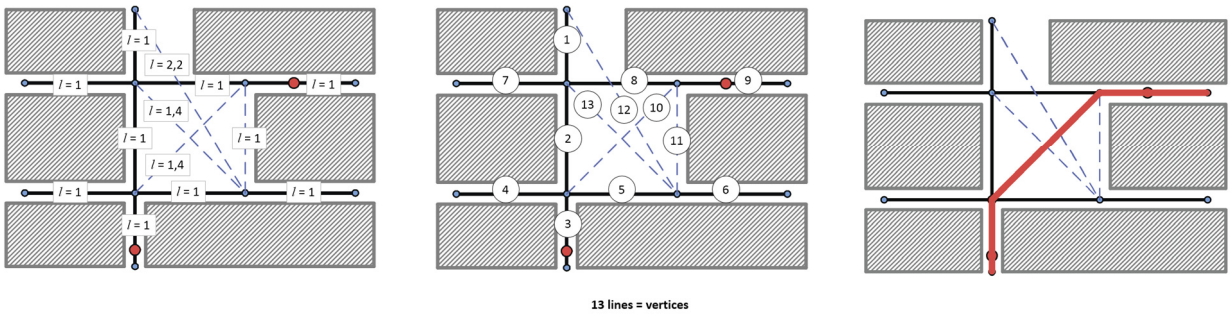
4) Doubling the geometry to introduce directions. Indexing each segment. Figure 10b shows indexes for segments without directions (It can be assumed that each segment actually is represented by two lines as id' and id")

5) Building inverted graph and assigning the weight/distance to the edges according to defined formula. See section 3.6. For example, we need to calculate the weight for the edge between vertex(i.e. segment) 3 and vertex(i.e. segment) 10, which more detailed are shown on Figure 11a.

According to default formula in section 3.6 we need maximum length among segments for normalization, which is the length of segment 12 and is 2.23 units. Length of segment 3 is 1 unit and segment 10 is of 1,4 unit length and we need for calculation the length between the center points. Angle between two segments is 45°. By formula, we get the weight, which is 0.66. For Floyd Warshal algorithm for all vertices shortest path distances are stored in matrix form as on Figure 11b.

$$weight(3,10) = 1.0 \cdot ((1 + 1.4)/2)/2.23 + 1.0 \cdot 45/360 = 0.66$$

6) According to the input data of origins and destinations shortest paths are extracted from matrix. Figure 10c.

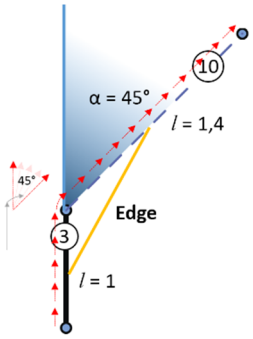


(a) Additional paths. Length markers

(b) Indexing

(c) Shortest path result

Figure 10: Algorithm illustration



(a) Weight calculation for edge 3-10

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	0,45	0,89	1,14	1,14	1,33	0,7	0,7	1,14	1,36	1,39	1,15	0,66
2	0,45	0	0,45	0,7	0,7	1,14	0,7	0,7	1,14	0,91	1,39	1,6	0,91
3	0,89	0,45	0	0,7	0,7	1,14	1,14	1,14	1,33	0,66	1,39	1,74	1,36
4	1,14	0,7	0,7	0	0,45	0,89	1,39	1,39	1,33	0,66	1,14	1,49	1,36
5	1,14	0,7	0,7	0,45	0	0,45	1,39	1,39	1,39	0,91	0,7	1,05	0,91
6	1,33	1,14	1,14	0,89	0,45	0	1,33	1,39	1,39	1,36	0,7	0,9	0,66
7	0,7	0,7	1,14	1,39	1,39	1,33	0	0,45	0,89	1,36	1,14	1,85	0,66
8	0,7	0,7	1,14	1,39	1,39	1,39	0,45	0	0,45	0,91	0,7	1,85	0,91
9	1,14	1,14	1,33	1,33	1,39	1,39	0,89	0,45	0	0,66	0,7	1,85	1,36
10	1,36	0,91	0,66	0,66	0,91	1,36	1,36	0,91	0,66	0	0,91	1,96	1,83
11	1,39	1,39	1,39	1,14	0,7	0,7	1,14	0,7	0,7	0,91	0	1,15	0,91
12	1,15	1,6	1,74	1,49	1,05	0,9	1,85	1,85	1,85	1,96	1,15	0	1,27
13	0,66	0,91	1,36	1,36	0,91	0,66	0,66	0,91	1,36	1,83	0,91	1,27	0

(b) Matrix of shortest path distances for Floyd Warshal algorithm

Figure 11: Building inverted graph

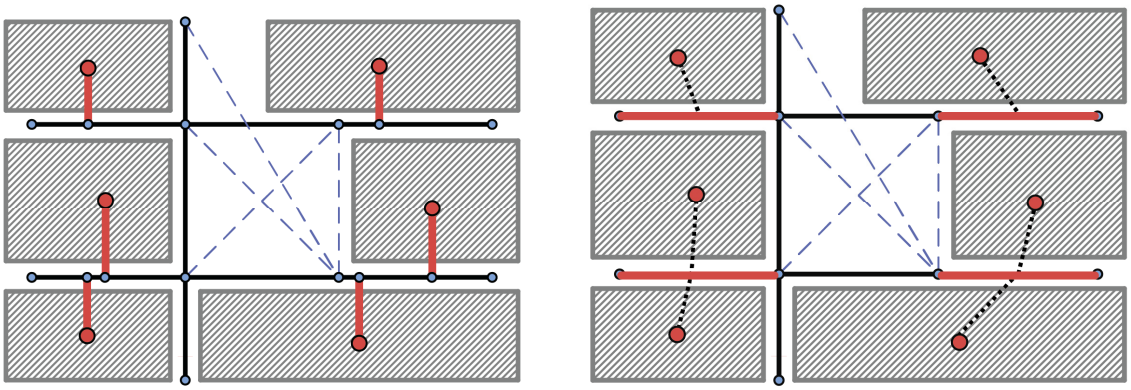
2.5. Definition of origins and destinations

For calculation of all to all results for origins/destinations (o/d) input existed segments can be taken.

In case more precise definition 2 ways are possible. For example, if take center points out of building contour as possible o/d, then we can build perpendicular segments to the closest ones, split it in point of intersection and use the perpendiculars as input for o/d (Figure 12a). The other way is by selected points choose closest segments (Figure 12b).

First approach generates a lot of additional geometry, in case of changing of selection points locations whole distance matrix should be rebuild (Section 3.2).

Second approach is easier to compute and changing the selection points do not leads the matrix re-computation (Section 3.3).



(a) Introduce new segments (b) Choose existed segments

Figure 12: Two possibilities of definition of origins and destinations

3. Grasshopper component setup description

3.1. Step 1: Processing inputs. BuildPaths

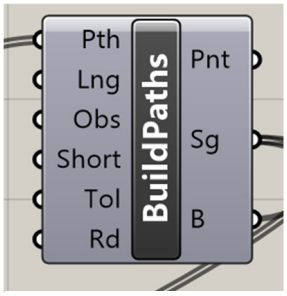


Figure 13: BuildSegmentGraph component

Input parameters:

Pth* - Path geometry. Non-intersecting (only from/to points) sequential lines are expected.

Lng - Length of segments subdivision (default = 10000). Used to make all segments of approximately equal length.

Obs - Obstacles geometry. Lines are expected. This geometry is used in case of additional short paths generation or visibility graph lines addition (if Short input parameter is true).

Short - Boolean. Defines is it needed to include generation of new paths/visibility lines or not.

Tol - Tolerance parameter (default = 0.1). Whenever two curves get close enough within tolerance, an intersection is assumed.

Rd - Radius parameter. If Short is true, then new paths are checked within this radius.

Output parameters:

Pt - Points extracted from segment ends.

Sg - Segments geometry.

B - "Builder" object. Serves as and object to pass information between steps/components.

3.2. Step 2-1: Build graph - 1st approach. BuildSegmentGraph

This approach assumes that origins/destinations will be segments itself. And further if on step 4 as origins or destinations points geometry will entered, then for this points existed segments will be assigned, according to which is closer to what.

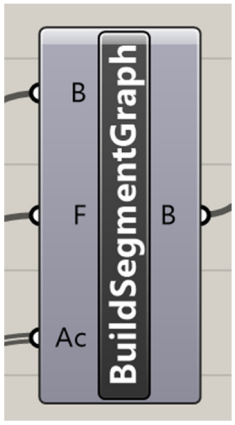


Figure 14: BuildSegmentGraph component

Input parameters :

B* - "Builder" object.

F - Formula definition.

Ac - List with information about additional cost/weight assigned to segments.

Output parameters:

B - "Builder" object.

3.3. Step 2-2: Build graph - 2nd approach. BuildFullGraph

This approach assumes that origins/destinations will be points and to define origin/destination segments perpendicular lines from this points to the closest existed segments are built and are added to the paths information.

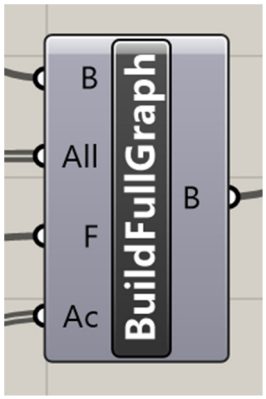


Figure 15: BuildFullGraph component

Input parameters :

B* - "Builder" object.

All* - All points which will further be assigned as origins/destinations. Usually can be obtained as center point of buildings.

F - Formula definition.

Ac - List with information about additional cost/weight assigned to segments.

Output parameters:

B - "Builder" object.

3.4. Step 3: Run Floyd Warshal algorithm. ShortestFloydWarshal

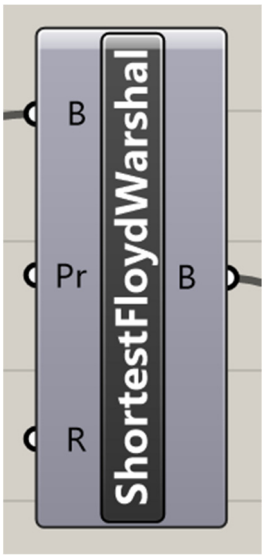


Figure 16: ShortestFloydWarshal component

Input parameters :

B* - "Builder" object.

Pr - Degree of parallelism. (default = 8)

R - Radius parameter to define within which distance calculate shortest paths.

Output parameters:

B - "Builder" object.

3.5. Step 4: Metrics calculation. FWGetPaths

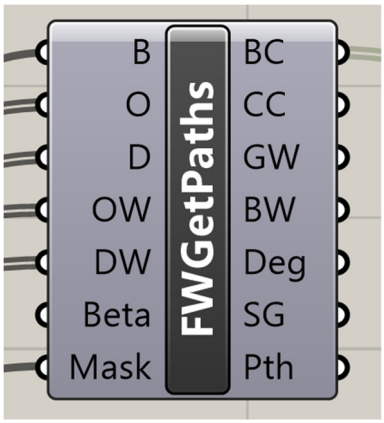


Figure 17: ShortestFloydWarshal component

Origins and Destinations could be specified as combined input, for instance, for O it is Points and for D it is lines, however, it is possible only for the 2nd approach. For the 1st approach only points are available for the input.

Input parameters:

B* - "Builder" object.

O* - Origins geometry (Lines/Points).

D* - Destinations geometry (Lines/Points).

OW - List of weights of origins. Used in calculations of Weighted Betweenness measure (See formula 6).

DW - List of weights of destinations. Used in calculations of Gravity and Weighted Btweenness measure (See formulas 5 and 6).

Beta - Parameter used in calculations of Gravity and Weighted Betweenness measure (See formulas 5 and 6).

Mask - String of boolean flags which defines the set of output parameters will be calculated. (Default "0011111", where the last "1" bit is corresponds to "BC" output, second is for "CC" and so on, the last "zero" bit is for "Pth" output.)

Output parameters:

BC - Betweenness Centrality measure. (See section 4.5)

CC - Closeness Centrality measure. (See section 4.5)

GW - Gravity measure. (See section 4.5)

BW - Weighted Betweenness measure. (See section 4.5)

Deg - Degree Centrality measure. (See section 4.5)

SG - Supplementary geometry is lines which will show to which segment point is corresponds if Step 2-1 is chosen and as origin/destination points geometry are entered.

Pth - Tree structure output of all shortest paths.

3.6. Formula definition for step 2

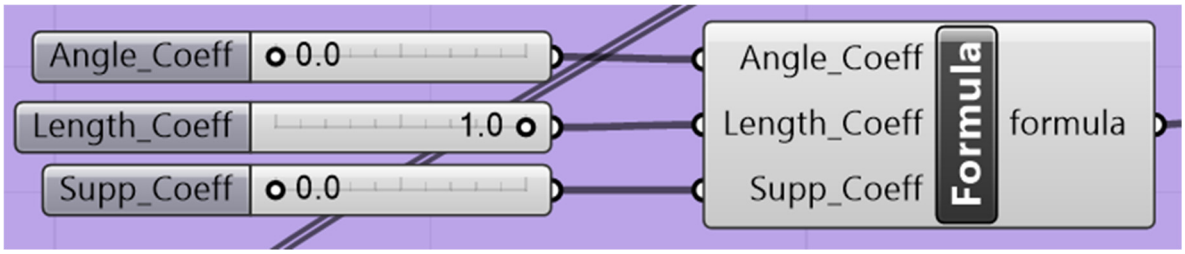


Figure 18: Formula definition

Formula definition assumes combination and manipulation of three parameters: Angular measure, Length measure and some user measure assigned to the edge (two adjacent segments as graph it is inverted axial map). All three parameters could be manipulated by coefficients. By default formula is:

$$\boldsymbol{Formula} = \mathbf{1.0} \cdot \boldsymbol{Length} + \mathbf{1.0} \cdot \boldsymbol{Angle} + \mathbf{0.0} \cdot \boldsymbol{UserParam} \quad (1)$$

where *Length* is the geodesic length between segments' center points normalized by largest length and *Angle* is angle in degrees between two segments normalized by 360. *UserParam* is not calculated by default, but it could be any list of user defined weights associated to the edge of inverted graph (see Figure 11a)

Figure 18 shows custom C# component "Formula", which allows to change formula equation. It takes 3 coefficients-parameters according to the parameters in formula and returns function object. Below is code of the component.

```
private void RunScript(double Angle_Coeff, double Length_Coeff,
double Supp_Coeff, ref object formula)
{
      this.ac = Angle_Coeff;
      this.lc = Length_Coeff;
      this.sc = Supp_Coeff;
      // return value should be a function with 3 double input values
```

```
      // and with output double value
      formula = (Func<double, double, double, double>) this.Result;
}

public double ac = 1;
public double lc = 1;
public double sc = 0;
public double Result(double length, double angle, double custom)
{
      return this.lc * length + this.ac * angle + this.sc * custom;
}
```

4. Component workflow description

4.1. Step 1: Processing inputs. (BuildPaths)

- Builder object is created
- Long segments are subdivided by defined length parameter(Lng)
- If (Short) parameter is true then new paths (visibility segments) are added
- All input segments are doubled to represent two possible directions of movement

4.2. Step 2-1: Build graph - 1st approach. (BuildSegmentGraph)

- Copy Builder object
- Assign id to every segment
- Build graph with weighting formula. Graph structure stored in next matrices:
  - adj_matrix - for each id pair the result of formula calculation is stored as double value.
  - adj_matrix_obj - for each id pair the Cost object is stored (angle and geodesic length separately)

4.3. Step 2-2: Build graph - 2nd approach. (BuildFullGraph)

- Copy Builder object
- Build perpendicular lines from points to the closest segments and subdivide the segments in the intersection point.

- Add doubled representation of new perpendicular lines; remove and then add again the subdivided lines (segments which are split by intersection point).
- Assign id to every segment
- Build graph with weighting formula. Graph structure stored in next matrices:
  - adj_matrix - for each id pair the result of formula calculation is stored as double value.
  - adj_matrix_obj - for each id pair the Cost object is stored (angle and geodesic length separately).

4.4. Step 3: Run Floyd Warshal algorithm. (ShortestFloydWarshal)

On this step Floyd Warshal (Silva 2014, Schiavoni 2008) algorithm is launched for the obtained matrices on step 2. Depending on is radius defined or not two slightly different algorithms are implemented. This step produces next matrices:

- dist_matrix - for each id pair shortest distance (sum of weights) is stored
- dist_real_length_matrix - for each id pair shortest geodesic distance is stored

After, all doubled segments and their values in matrices are merged into reduced matrices versions (See section 2.3).

4.5. Step 4: Metrics calculation. (FWGetPaths)

On this step different measures are calculated based on input of origins and destinations.

Betweenness Centrality (Freeman 1977)

Quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.

$$\boldsymbol{BC(v)} = \sum_{\boldsymbol{s \neq t \neq v \in V}} \boldsymbol{\sigma_{st}(v)} \tag{2}$$

Where σ_{st} is total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v.

Closeness Centrality (Bavelas 1950)

Is the average length of the shortest path between the node and all other nodes in the graph. Thus the more central a node is, the closer it is to all other nodes.

$$\boldsymbol{CC(s)} = \frac{\boldsymbol{\varphi_{st}^{2.5}}}{\sum_{t \neq s \in V} \boldsymbol{d(s,t)}} \tag{3}$$

where $d(s,t)$ is a distance (metric or non-metric) of shortest path from node s to node t and φ_{st} is total number of shortest paths which is reachable from s. In case of zero sum of distanses other formula is used (it happens when only angular measure is used and all paths are on straight line. Such case is rare, but appears during the radius filtering, which remains some parts of straight regions).

$$\boldsymbol{CC(s)} = \frac{\boldsymbol{\varphi_{st}}}{\mathbf{0.01}} \tag{4}$$

Gravity (Kalvo 2015)

The Gravity measure assumes that accessibility at Origin s is proportional to the attractiveness (weight) of destinations t, and inversely proportional to the distances between s and t.

$$\boldsymbol{GR(s)} = \sum_{t \neq s \in V} \frac{\boldsymbol{W_d[t]^{\alpha}}}{\boldsymbol{e^{\beta \cdot d(s,t)}}} \tag{5}$$

where $W_d[t]$ is the weight of Destination t, $d(s,t)$ is a distance (metric or non-metric) of shortest path from node s to node t. α is the exponent that can control the destination Weight or attractiveness effect, and β is the exponent for adjusting the effect of distance decay.

Weighted Betweenness (Kalvo 2015)

Take into account both attractiveness of Origin and Destination and accumulate it according to number of times a node acts in shortest path calculation between other nodes.

$$\boldsymbol{BW(v)} = \sum_{s \neq t \neq v \in V} \frac{\boldsymbol{W_d[t] \cdot W_o[s]}}{\boldsymbol{e^{\beta \cdot d(s,v,t)}}} \tag{6}$$

where $W_d[t]$ is the weight of Destination t, $W_o[s]$ is the weight of Origin s and $d(s,v,t)$ is a distance (metric or non-metric) of shortest path from node s to node t through node v.

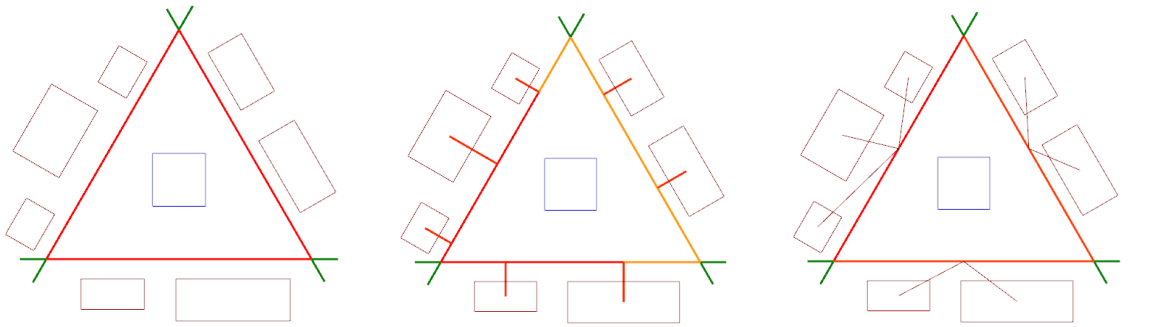
Degree Centrality (Diestel 2005)

Defined as the number of links incident upon a node (i.e., the number of ties that a node has).

$$\boldsymbol{DC(v)} = \mathbf{deg(v)} \tag{7}$$

5. Examples and tests

Consider a small situation, which will represent a part of possible city plan with buildings, some obstacle objects to avoid and given path lines.



(a) Segments as input

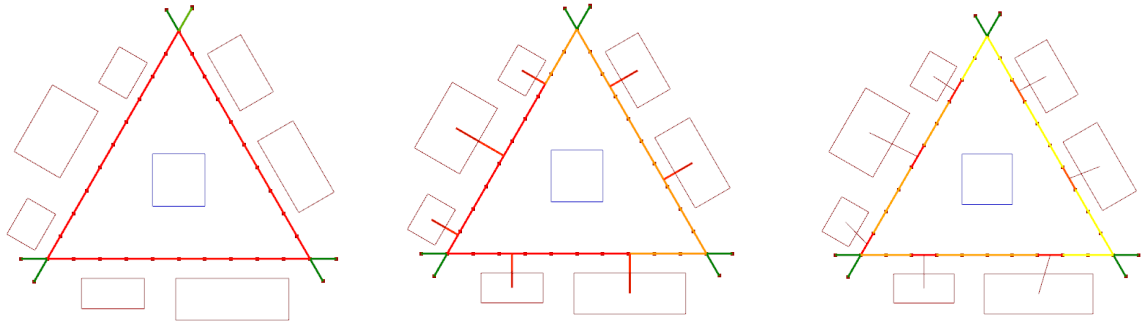
(b) Points as input (center of each building polygon) subdivide existed segments and generate new ones

(c) Points as input select existed segment, closest to each point

Figure 19. Definition of possible origins and destinations. Illustration of Betweenness Centrality measure for each case.

Figure 21 illustrates the result of component output with difference in input specification. All three cases eventually work with segments but the algorithm of determining which segments become origin or destination is different.

Figure 22 illustrates the result of component output for each input data case with additional subdivision of each segment. This step allows to achieve homogeneity in geometry and to provide more accurate results for cases illustrated in Figure 22a and Figure 22c.



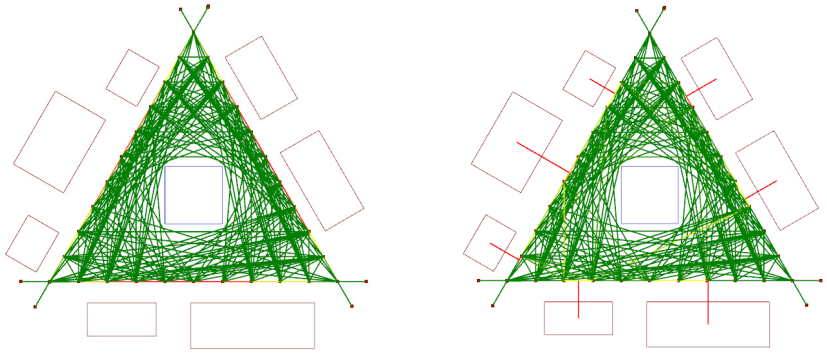
(a) Segments as input

(b) Points as input with subdivision

(c) Points as input with selection

Figure 20. Division of segments on equal parts of 1 unit length for each input case. Illustration of Betweenness Centrality measure for each case.

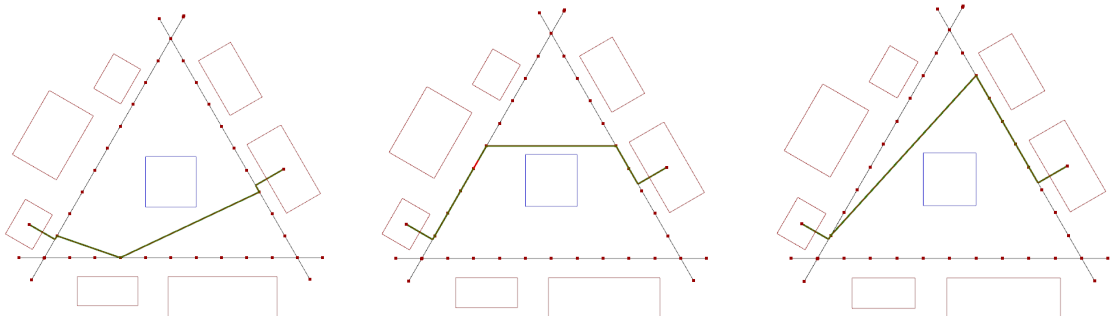
Figure 23 illustrates the result of component output with generation of additional paths geometry based on subdivision results. There is some object in the center, which specified as obstacle to prevent generation of new path through it.



(a) Segments as input (b) Points as input

Figure 21. Generating additional paths geometry

Figure 24 shows effects of generated paths on shortest path between two particular points for three situations: a – shortest path in terms of geodesic measure; b – in terms of combined angular and geodesic measures; c – in terms of only angular measure.



(a) Geodesic measure (b) Geodesic and angular measure (b) Angular measure

Figure 22. Shortest path between two particular points after additional path information is generated

Figure 25 shows the effect of radius application on generating process of new paths. It allows to control how far it is reasonable to generate new paths.

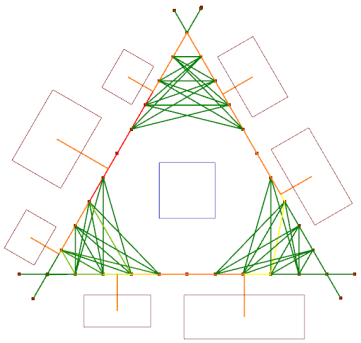


Figure 23. Radius application for generating new paths

Considering examples, which are closer to the real cases, there is provided result of analyzing of a city plan of Bad Berka town.

First example, demonstrates 1st approach and uses center points of buildings as origins/destinations (Figure 26). Time of calculation is more than 2 hours. Amount of segments(vertices of graph) is >7000.

Second example, demonstrates 2nd approach and uses only segments as origins/destinations (Figure 2). Time of calculation is approximately 9 seconds. Amount of segments(vertices of graph) is >1200.

Figure 24: Example of 1st approach. Betweenness centrality visualisation.

6. References

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. Sociometry, 35-41.

Bavelas, A. (1950). Communication patterns in task-oriented groups. Journal of the acoustical society of America.

Diestel, R., (2005), Graph Theory (3rd ed.), Berlin, New York: Springer-Verlag,

Silva, F. (2014) Floyd Warshall algorithm: All-Pairs Shortest Paths. http://www.dcc.fc.up.pt/~fds/aulas/CP/1415/Trab1/project1.html.

Dawes, M., & Ostwald, M. J. (1926). Precise Locations in Space: An Alternative Approach to Space Syntax Analysis using Intersection Points. Architecture Research, 3(1), 1-11.

Schiavoni, V., (2008). Floyd Warshall algorithm: optimization. http://www.jroller.com/vschiavoni/entry/a_fast_java_implementation_of

Kalvo R. Sevtsuk A. (2015) City Form Lab Help. https://urbanterrainsdigitallab.files.wordpress.com/2015/11/cityformlab_una_eng.pdf.