POLISH ACADEMY OF SCIENCES
INSTITUTE OF GEOGRAPHY AND SPATIAL ORGANIZATION
00-818 Warsaw, Twarda 51/55

Wojciech Pomianowski, Jerzy Solon

GraphScape

Technical Report

edition 2

(software version 3.2)

GraphScape version 3.1.3 developed under the project 2012/05/B/ST10/02173
financed by the Poland's National Science Centre

Welcome

GraphScape is a software for landscape structure analysis and mobility modelling based on graph theory. It is intended to fill the gap between classic patch-metrics approach and functional connectivity approach. It works upon vector-based maps in widely accepted ESRI Shapefile format and easily interacts with ESRI ArcGIS platform and other GIS software.

GraphScape is focused on graph-related analysis, leaving other tasks like data preparation and results visualization to spreadsheet, database, GIS and statistical software.

Recent changes

ver. 3.2

- New handling of MST segmentation problem. Till ver. 3.1.3 MST building ended up with an error if complete MST could not be built. This happened when one or more Analysis Set nodes were isolated from the remaining MST body. Now, PARTIAL MST is built and missing edges are reported, along with topology error detection.
- Zero values in Transfer Resistance table (see MOBILITY FACTORS) are no longer converted to user-infinite resistance.
- MST Paths shapefile has additional “No” column with serial path number (see SAVING TO SHAPEFILES).
- Shortest MST path steps indicator is saved to Patch-related results shapefile.
- Non-matching class values in mobility files do not stop processing (see DEFINING MOBILITY). Instead, they are reported in Report Logbook.

Part I – Concepts

Processing flow and key data structures

GraphScape is composed of 4 functional modules, contained within a single executable program with windows-based interface:

1. Topology Builder
2. Mobility Model
3. Path Engine
4. MST Engine.

The processing flow is linear: once loaded, the data is subject to consecutive transformations until final results appear. Apart from setting various modelling options, the only operation to be performed is running the model with a "Run" button.

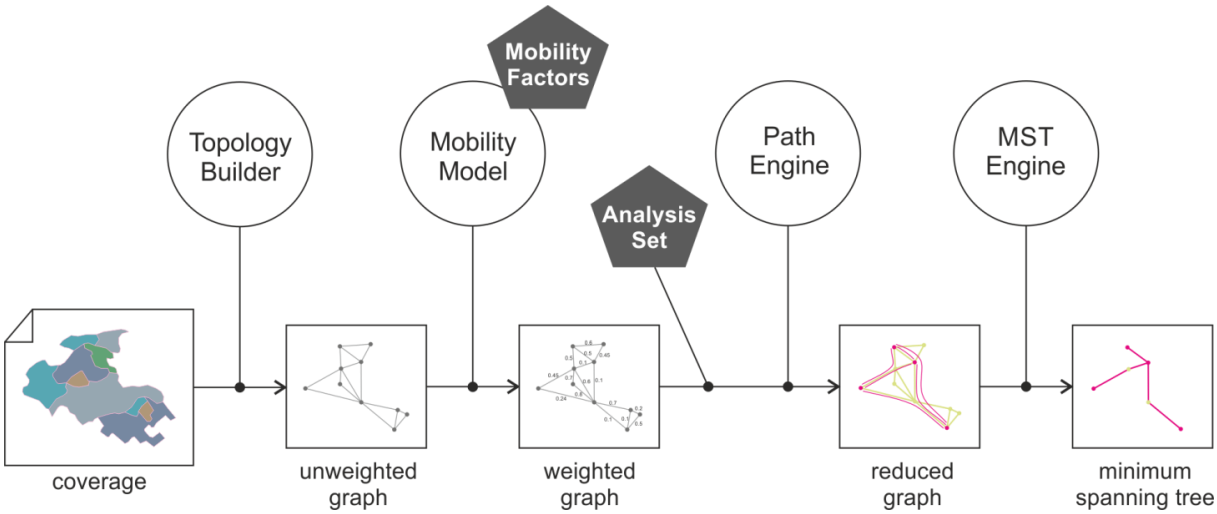


fig. 1 GraphScape architecture

As illustrated on fig. 1, input shapefile map is transformed into unweighted graph, then into weighted graph according to user-defined mobility factors. Next, for a given set of patches (called Analysis Set), reduced graph is prepared by Path Engine. Then, MST Engine finds Minimum Spanning Tree (MST) on a reduced graph. Finally, MST is presented on the background of the original map, and in form of listings and statistics. These may be carried to other software for further processing or saved to files. Intermediate data is also available for a better insight into the program operation or as a source for modeling performed externally. For instance, GraphScape may be used only for shapefile topology verification or as a source of graph data.

In the following chapters we will explore the concepts and algorithms used in GraphScape in accordance with four-step processing flow. Next, Part II will provide a practical guidance to the program operation.

Topology Builder

An input data provided to GraphScape is classified, continuous map of patches (areal units homogenous under study assumptions) covering a certain extent of earth surface. An alternative name **mosaic** will be also used. An accepted file format is **ESRI Shapefile**, which stores a set of objects with geometry and numeric or text attributes. At least one attribute should provide the data suitable for mobility modelling; other attributes may be present but are not further used. The overall shape of area of interest is arbitrary and some discontinuities (gaps) are accepted, unless they break the mosaic into separate, disjoint parts[1].

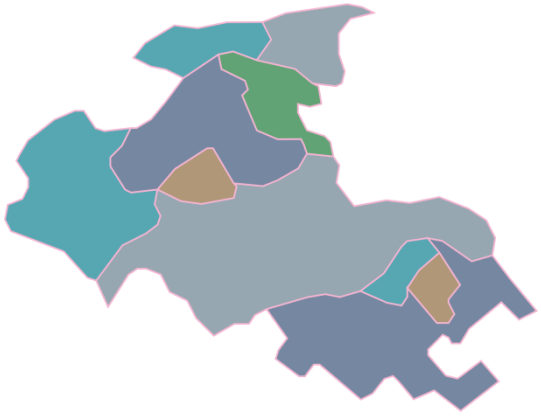


fig. 2 Input patch mosaic

A well known shapefile feature is that is has no provision for topology. For a contiguous patch map, the complete border of an each patch is stored, so borders of adjacent patches are effectively duplicated. No information on this adjacency relation is explicitly given.

The purpose of Topology Builder module is to discover the topological relations then to transform the mosaic into isomorphic structure of **graph**, suitable for modelling. To define this graph, we establish two mathematical mappings: one between patches and graph nodes and second between pairs of patches and graph edges.

A graph *G (V, E)* made of node set *V* and edge set *E* will be built in such a way that:

1. $v_1, v_2, \dots, v_n \in V \Leftrightarrow p_1, p_2, \dots, p_n \in P$
   There is a one-to-one mapping of node set *V* onto patch set *P*. In other words, every node has corresponding patch and every patch has corresponding node.
2. $E: \{e(v_a, v_b)\} \Leftrightarrow adj(p_a, p_b) = true$
   An edge exists between nodes v_a and v_b if adjacency relation holds true for a pair of patches (p_a, p_b).

Now, we will define adjacency relation *adj*, with a requirement coming from condition (2), that it must be defined for every pair of patches:

3. $\forall (p_a, p_b) \in (P \times P) \quad adj(p_a, p_b) = \begin{cases} true\ if\ shared\_border(p_a, p_b) \neq \emptyset \\ false\ if\ shared\_border(p_a, p_b) = \emptyset \end{cases}$

Shared_border is a function those arguments are two patches p_a and p_b, and the result is a set of multiline geometric objects (or empty set). Each object consists of a string (an ordered set) of points belonging to p_a and p_b. *Shared_border* may be viewed as a more sophisticated version of *adj* relation, providing surplus of information that, as we shall see later, will be valuable.

[1] For instance brown-colored patches on fig. 2 can be removed leaving unharmful gaps, but light-gray patches – not.

Declarative definitions given above are better explained by procedural description. A proprietary algorithm starts with matching all border points against each other. Points are considered equal if their both ϕ and λ coordinates are equal. Matching points are coalesced (joined into one object, with information on patch origin preserved). Next, coalesced points are traced into **shared borders** – strings of points with common left and right origin patch pair (see fig. 3).

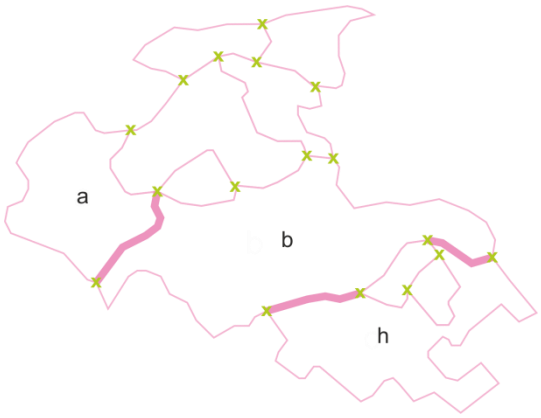


fig. 3 Shared borders identified

Resulting list of line segments contains objects representing complete borders (see border *a-b* on fig. 3) or parts of borders (*b-h*). Degenerated lines consisting of a single point are rejected.

At this stage, an algorithm builds an internal representation of a graph – an **Adjacency Matrix** (AM). It is binary, symmetric matrix of size *nodes x nodes*. Only below-diagonal elements are stored.

node (patch) id	1	2	3	4
1				
2	0			
3	1	0		
4	0	1	1	

fig. 4 Adjacency Matrix

Read straightforward, definition (3) suggests that shared borders should be grouped into sets for each patch pair to answer questions about adjacency relation. However, checking for non-emptiness of a set requires finding only a first element of a set. This observation leads to more effective procedure. AM is initially filled with 0's. Then is fed by running along the shared borders list. Given the list element `SB[i]`, with the attributes `left` and `right`, an assignment is `AM[SB[i].left,SB[i].right]=1` (multiple assignments do not change cell's content in any way). The remaining, unassigned AM cells denote non-adjacent patches. Shared borders list is kept as a supplementary structure, mainly for display purposes.

The resulting graph may be described as **unweighted**, because it has no extra numeric information on edges and **undirected**, because edges have no orientation towards any endpoint. This is the consequence of definition (3): the adjacency relation between two patches is necessarily symmetric.

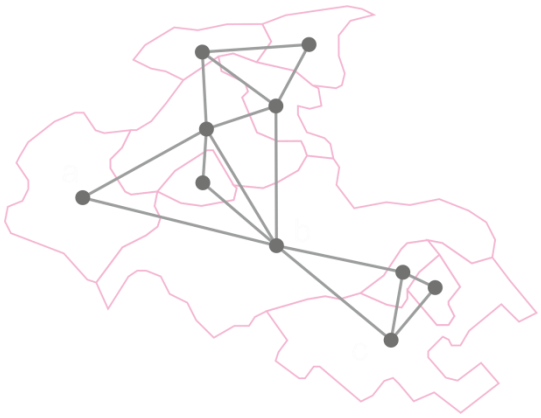


fig. 5 Patch mosaic transformed into graph

From now on, the terms *patch* and *node* will be used interchangeably, the former in the context of area-related or geometric properties, the latter – in the context of topologic or graph-related properties.

Mobility Model

Mobility Model is the most important GraphScape module, responsible for formulating mobility-related problem on the graph created by Topology Builder. At this stage the end-user creates a model of a **mobile agent**, that is the entity (living individual, a group, a species, a substance, etc.) performing a movement in the landscape. An assumption is that the movement is controlled exclusively by graph topology and properties assigned to graph nodes and edges, so the only way to express the knowledge about mobile agent is via three **mobility factors** – properties assigned to nodes and edges.

Mobility Model's purpose is to transform initial unweighted graph, which accounts for pure topology information only, to the **weighted graph**, which accounts also for other factors influencing mobility. A weight (numeric value) assigned to each edge will correspond to generalized "impedance of travel" between two nodes. Depending on the problem, it may represent time, cost, inverse probability of transfer, etc.

The internal representation of graph at the exit from Mobility Model is **Travel Matrix** (**TM**) – a floating point symmetric matrix of size *nodes x nodes*.

node (patch) id	1	2	3	4
1				
2	∞			
3	1.7	∞		
4	∞	1.1	1.9	

fig. 6 Travel Matrix

Compared to AM, not only value type is changed from binary to numeric, but also the interpretation changes. Low values represent quick travel; high values represent slow travel, with special case of infinity – no travel possible al all.

Please note, that on all illustrations after fig. 5, the length of a graph edge will represent weight and not geometric distance.

Mobility factors

The calculation of weights for Travel Matrix is based on three user-defined parameters. Two of them are related to sole properties of nodes (patches); one defines the relation between nodes. An effort is made to specify mobility factors in the most abstract way, so that they are devoid of direct

references to phenomenon being modeled, and their interpretation is solely based on the role in building the weights of the graph.

Patch resistance is resistance to travel – or alternatively – a cost of travel through a graph node. As graph node represents a patch with non-zero surface, this surface can incur a resistance to mobile agent. As a matter of convenience, this influence has been split into two factors. With **Patch Class Resistance** (**PCR**), the user assigns resistance to whole classes of patches (like a land-use classification type), with **Patch Individual Resistance** (**PIR**), the user assigns the resistance each patch individually. With both resistance factors, high values increase, low values decrease the weight assigned to Transfer Matrix cell. Apart from assignment method both factors share the same interpretation and performance in calculations. First factor is better suited to capture inherent behavior of mobile agent in relation to certain patch types so it is appropriate when this behavior is consistent within whole classes of patches. Second factor gives a way of incorporating unique properties of patches which are difficult to generalize and fit into classification scheme. The most notable property of this kind is geometric shape and PIR is the suitable way of introducing one of many available shape or size metrics into the modelling process.

PCR is supplied as an external table of type-value pairs. Because PIR is a property of each patch, the most natural way of providing this value is by one of the columns of shapefile coverage.

Type	*PCR*
A	0.0
B	0.0
C	0.7
D	0.0

fig. 7 Patch Class Resistance table

Transfer Resistance (**TR**) is a way of specifying a resistance to (or probability of failure of) travel from node to node. This factor can only be assigned class-wide, like PCR, so actually it describes a relation between classes of patches. Here again, high values increase and low values decrease the weight. Due to requirement of symmetricity of Travel Matrix, Transfer Resistance matrix, supplied as an external table, must be symmetric too (on fig. 8 cells below diagonal are omitted).

Type	A	B	C	D
A		1.0	0.8	1.1
B			1.0	1.0
C				1.5
D				

fig. 8 Transfer Resistance matrix

When building mobility factors tables, the user must take care to recognize the different logic behind patch resistance and transfer resistance. Transfer resistance should only account for the cost of crossing the borders or changing the environment during the movement. It must not be interpreted as a placeholder for “combined” characteristics of two adjacent patches.

Scaling of mobility factors

GraphScape accepts $\mathbb{R}_0^+$ (positive or zero) real numbers as mobility factor values. Negative values are converted to infinite resistance.

The safest and most standard way is to limit the resistances to 0..1 range. While 0 value

GraphScape assumes value 1.0 is neutral for three reasons:

1. It is default weight in step model (see below), so if two kinds of model (step and regular) are expected to give compatible results, mobility factors should be scaled to “tend” to neutral value 1.0.

2. It is default value for omitted mobility factors. If user-supplied mobility factor is mixed with default 1.0 factor, the former should be scaled to "tend" to neutral 1.0, otherwise one of them will have dominating influence on result.
3. It is default value for missing rows in PCR table and missing row/column combinations in TR matrix. If only few classes of interest are defined, the remaining classes will fall back to 1.0.

Avoid exceedingly big values so that no clash with infinity values occur (see DEFINING MOBILITY). A sum of resistances of all types along any path in the system should not exceed user-defined infinity.

Step model with border factor

Another option for defining mobility is not to use mobility factors at all. A fallback model employed when the user does not specify *any* factor is called **step model**. It is a derivation of an idea of path calculation in unweighted graph where, for lack of proper weights, 1.0 value is used as edge weight, so resulting graph may be called pseudo-weighted. In paths calculation, this logic leads to total length of path equal to the number of steps made between path origin and destination[2], which is the same as the number of edges or number or crossed borders. Step model imposes an extra homogeneity requirement on input map: patches should be reasonably equal-sized and uniform in terms of connectivity. Abnormally connective patches, often elonginated or star-like shape, will serve as movement conduits and attract shortest paths too strongly. The effect is more pronounced than in regular model, because there is no way of compensation with the help of mobility factors.

Pure step-based calculation has two fundamental drawbacks, which prohibit it's direct use in modelling.

1. The answer to the shortest path problem is not unique. There can be many paths of equal length between two nodes, with majority of alternative paths being a trivial variants of the base one. The number of possibilities increases as nodes become relatively far apart from each other.
2. Minimum spanning tree cannot be determined because all spanning trees have the same length (see MINIMUM SPANNING TREE for further explanation).

To overcome these problems, GraphScape makes use of additional (non-topological) criterion of the length of shared borders between patches. The assumption here is that the opportunity for transfer or probability of interaction between two patches increases when they share a long border. Out of two paths with the same number of steps, the one with longer sum of shared borders will be selected as shortest. Details of calculation are given in STEP FORMULA WITH BORDER FACTOR chapter.

PCR and TR assignment details

To be used in modelling, resistance values from external tables must be assigned to nodes and edges, based on type-matching.

A type value (e.g. "meadow") from PCR table is matched against type in patch table and appropriate values are assigned to individual patches. This is accomplished by a composition

```
node.PCR = read (match (node.type))
```

of two functions:

```
match: node.type → table.row
read: table.row → table.value
```

In case of Transfer Resistance, additional mapping is necessary and function chain is as follows:

```
edge.TR = read (match (links (edge)))
```

where

[2] This Step statistics is calculated anyway, even for well-defined mobility, and is included in final report.

```
links: edge → startnode, endnode
match: node.type, node.type → table.row, column
read: table.row, column → table.value
```

The functions described above define the assignment process, but are not invoked until total weight formula is applied.

Weight formula

Normal formula

Three mobility factors must be combined in order to arrive at a single weight value for Travel Matrix cell. For any i (origin) and j (destination) node pair, the formula is as follows:

$$w(i,j) = \left(\frac{PCR(i) + PCR(j)}{2} + \frac{PIR(i) + PIR(j)}{2} \right) TR(i,j)$$

def. 1 Weight formula

If one of mobility factors (PCR, PIR or TM) is not defined, then a value of 1.0 is used instead.

It is important to note the difference of attitude towards patch resistances and transfer resistance. Final *w (i,j)* value is assigned to graph edge, but patch resistances are not properties of an edge. They have to be derived from neighboring nodes. Formula implies that half of origin patch resistance and half of destination patch resistance give a total resistance to be fought against during crossing from origin to destination.

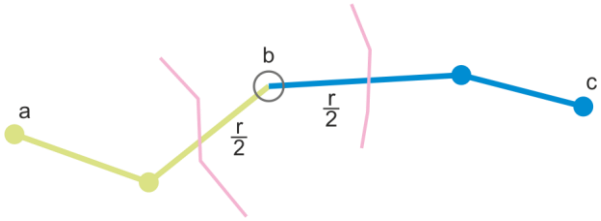


fig. 9 Patch resistances combined

The averaging formula assures that resistance is never lost when combining paths. For instance on fig. 9, a total path weight of a-c will be equal to the sum of a-b and b-c total weights.

The relation between patch resistance and transfer resistance is multiplicative so these two types contribute equally to the final result. The relation between two kinds of patch resistance is additive, so when both kinds are used, they may together outweigh transfer resistance – proper scaling is necessary.

Step formula with border factor

For this type of model, mobility is based only on topology and geometry, without any additional factors given by the user. The formula is:

$$w(i,j) = 1 - b(i,j)$$

def. 2 Weight formula for step mobility

where *b(i,j)* is **border factor** is between adjacent origin i and destination j patches:

$$b(i,j) = \frac{l(i,j)}{\sum l}$$

def. 3 Border factor formula

where *l(i,j)* is a length of a shared border between i-th and j-th patch. This value is divided by a total length of all shared borders in coverage, resulting in a very small fraction. During SP and MST computation, when weights are being summed, border factors decrease total weight. For two paths with equal number of steps (edges), it is the border factor that gives the winning path. The requirement must hold that Σb never exceeds 1.0 for any path; otherwise the step count would be artificially decreased by one. Border factor formula guarantees this, because there cannot be any path in a graph longer than the sum of all edges of the graph[3].

Travel Matrix assignment

In a final step of Mobility Model, an assignment to Travel Matrix occurs along the following formula:

$$TM(i,j) = AM(i,j)^{-1}\, w(i,j)$$

def. 4 Travel Matrix formula

Travel Matrix is a scalar product incorporating Adjacency Matrix and weight values calculated according to mobility definition. Inverted Adjacency Matrix term $AM(i,j)^{-1}$, returns ∞ for *AM* value of 0 (non-adjacent) and 1 for *AM* value of 1 (adjacent). The only purpose of this term is to incorporate topology information. Infinity values keep topology intact because they are invariant to any numerical operation. No matter how Mobility Model is defined, no edge can appear between non-adjacent nodes and no movement between them will ever occur (on the other hand, very high *W* values can also prohibit movement as if certain edges were removed from the graph). The formula implies that TM is sparse – it is mostly filled with infinity values and only cells for adjacent (i,j) pairs have numeric value.

In further processing, TM values are used whenever weights are needed. To keep consistency with graph terminology, in following text we will still use the term *weight* referring to graph property, but it must be born in mind that TM values will be used instead.

Path Engine

Analysis Set and reduction of graph

On exit from Mobility Model, GraphScape has Travel Matrix representing **full graph** for the whole coverage that is the information necessary for routing a mobile agent between graph nodes. However, mobile agent's movement is defined locally – only for adjacent nodes. It is still a far way from routing the agent between arbitrary, non-adjacent nodes, and this computation comes at high cost. On the other hand, we know that the major modelling concept of Minimum Spanning Tree is, by definition, based only on a subset of all nodes, so routing will be necessary only for a subset of pairs. The most economical way of proceeding is therefore to introduce the selection of nodes before routing.

At this stage, the user has an opportunity to specify nodes of interest, that is the nodes being a subject of analysis and relevant to the problem. The selection, usually a small fraction of original nodes, is called an **Analysis Set** and is created interactively (see BUILDING AN ANALYSIS SET chapter). Analysis Set, under the assumption of good connectivity, is a base for creation of a **reduced graph**.

[3] A border case is when a path spans the whole graph, but then there is no alternative path for comparison.

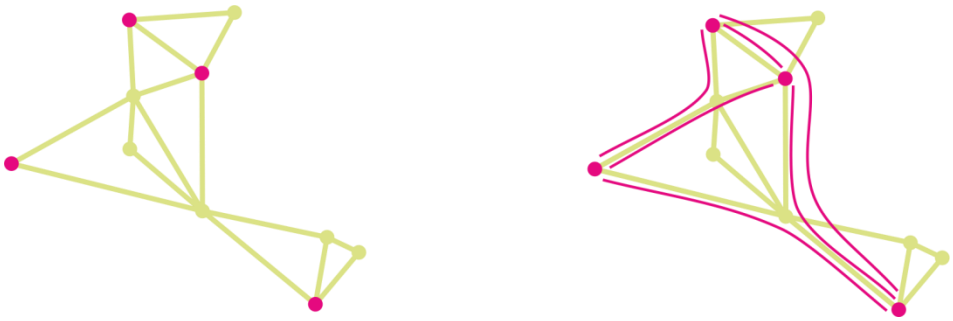


fig. 10 Graph reduction. Highlighted nodes belong to Analysis Set.

RG is derived graph and not a subgraph because it shares only nodes with original graph. It is however convenient to think of it as of subgraph, because subsequent operations will still refer to the full graph.

Starting from full graph *G (V,E)*, reduced graph *G'(V', E')* is specified as follows:

1. $V' \subseteq V$

   New set of nodes is drawn from a set of full graph's nodes. *V'* is equivalent to Analysis Set.

2. $\forall v' \in V'\ connectivity\ (v') = |V'| - 1$

   Each node *v'* has connectivity equal to total number of nodes minus one. In other words the graph is fully connected: an edge exists between every pair of nodes. The number of edges is *n(n-1)/2* for *n* equal to number of nodes.

3. $E' : \{e'(v'_a, v'_b)\} = shortest\_path(G, v_a, v_b)$

   New edge set *E'* consists of edges *e'* such that an edge between v'_a and v'_b is equal to the corresponding shortest path between v_a and v_b in full graph.

Restated in more job-oriented way, reduction is composed of picking up selected nodes and building all possible edges between them (for example, six on fig. 10). The edges are constructed from the shortest paths found in the full graph and this concept will be covered in the next chapter.

Computing shortest paths

A **path** in the graph is an ordered set of edges between given origin and destination nodes, with additional requirement that both edges and intermediate nodes being unique. A path can be alternatively represented as an ordered list of nodes and this representation will be also in use further. Many paths may exist between a pair of nodes and, in weighted graph, a **shortest path** is a path with a minimum sum of weights.

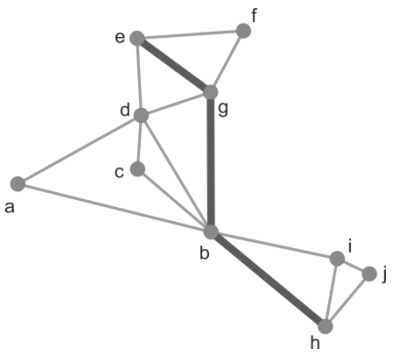


fig. 11 Shortest Path between nodes h and e

The logic of path weight computation occurring in SP algorithm is very simple: a path weight is the arithmetic sum of edges' weights.

$$W(path) = \sum_{e \in path} w(e) \quad = \quad \sum_{i=1}^{n} w(e_i)$$

def. 5 Path weight summation

This however leads to non-obvious considerations on weight function, discussed further in **WEIGHT FORMULATION UNDER GRAPH REQUIREMENTS**.

The job of Path Engine is to find a shortest path between every pair of nodes belonging to the Analysis Set. This is the most time-consuming computation performed by GraphScape and a lot of attention was paid to optimize this step. The computing problem is known under the name All-Pairs-Shortest-Paths and, like with many similar path problems, the core of the algorithm is Dijkstra's Shortest Path Algorithm (DSPA). One of the properties of DSPA is that it returns not only the total weight (the distance between pair of nodes), but also enumerates a list of edges or nodes constituting the path.

GraphScape implementation departs from classic textbook DSPA in three ways. First, it computes in a single step a complete set of paths from single origin node to all destination nodes (a tree) and this computation takes no more time than classic DSPA for a single pair. Second, the computation is split into multiple threads, each running on a separate processing unit. An algorithm has very good scalability, because each single-origin tree can be computed independently. Multithreading speed-up is almost proportional to the number of processor cores (typically 4-8)[4]. The third extension of DSPA is an introduction of a special weight pre-conditioning in case of step model, which effectively leads to bi-criterion DSPA.

SP algorithm results are stored in intermediate structure called Shortest Paths Matrix (fig. 12) with non-numeric elements. Here, row and column (origin and destination) headers contain only selected (new) nodes, but paths still refer to the nodes in full graph. This matrix represents a mapping between full graph and reduced graph.

	a	e	h
a	-		
e	e-d-a	-	
h	h-b-a	h-b-g-e	-

fig. 12 Shortest Paths Matrix

As we deal with a fully connected graph, SPM is dense. A presence of empty cell indicates that the graph is composed of more than one subgraph.

Finally, total weights of paths from SPM are calculated and stored as a **Reduced Travel Matrix**.

	a	e	h
a	-		
e	2.1	-	
h	6.0	6.2	-

fig. 13 Reduced Travel Matrix

It is easy to see that if step model is chosen, RTM can be calculated immediately from SPM by counting the path edges.

[4] Except for a very small Analysis Sets. If *a* is the size of Analysis Set and *c* is the number of cores, the speed-up factor is *min (a,c)*. For 6 objects in Analysis Set, on 8-core processor, the speed-up will be 6× and the processor utilization 75%.

Minimum Spanning Tree

Definition

A spanning tree T (V, E') is a subgraph of graph G (V, E) including all nodes of G and a subset of edges of G and having no cycles. For a given graph G there may be many spanning trees T composed of different edge sets E'. The number of edges of any T is equal to $|V| - 1$, and this is the least number of edges necessary to connect each and every node of G. For a weighted graph, the one with the least total weight is called **Minimum Spanning Tree** (MST).

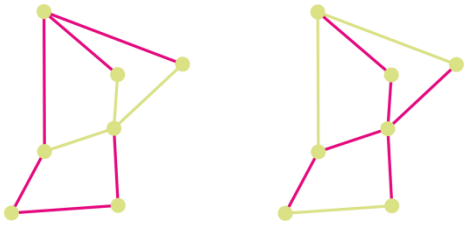


fig. 14 Spanning Tree (left) and Minimum Spanning Tree (right)

Weight summation for MST is carried out the same way as for paths:

$$W(MST) = \sum_{e \in MST} w(e).$$

def. 6 MST weight summation

As the number of edges of any T for a given graph G is equal, we cannot find MST in an unweighted graph. This limitation is also in effect for graphs, which use constant weights and are in fact not truly weighted. The consequence is that "pure" step model of mobility cannot yield minimum spanning tree at all[5]. In our modified step model (see STEP FORMULA WITH BORDER FACTOR), weights are no longer constant and totals are different for each T. This is why MST is valid.

MST Engine - Implementation

A graph of *n* vertices has n^{n-2} spanning trees, which is a big search space for brute-force algorithm (e.g. over 10^{32} possibilities for 25 nodes). GraphScape uses **Prim's algorithm**, which execution time never grows faster than n^2 and in practice is much lower. Compared to DSPA, speed requirements are lower because reduced graph is usually a fraction of the full graph. Therefore MST algorithm runs single-threaded.

Prim's algorithm's principle of operation is to build MST edge by edge, appending the shortest unused edge connecting to the new node (see Dao, 1974). This behavior is typical for *greedy* algorithms and more explanation is given in WEIGHT FORMULATION UNDER GRAPH REQUIREMENTS chapter.

A primary output of MST Engine is a list of edges of minimum spanning tree. Because the edges are made of paths of a full graph, they are reported under the name **MST paths**.

[5] In this situation MST algorithm, described further, chooses random spanning tree based on the ordering of nodes.

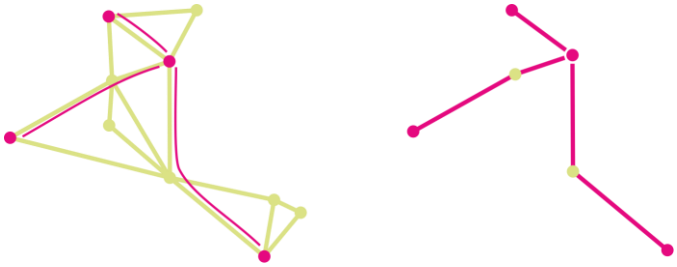


fig. 15 MST in reduced graph

The results should not be confused with superficially similar minimum spanning tree in full graph (fig. 16). The difference is that reduced graph MST edges are always terminated by Analysis Set nodes while this is not necessarily so for MST found in full graph. The latter MST could include out-of-interest nodes to improve connectivity of Analysis Set nodes.

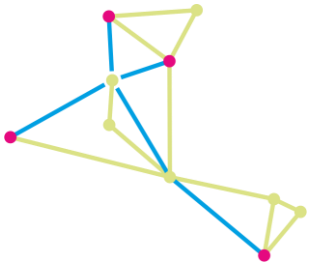


fig. 16 MST in full graph (not GraphScape output!)

Results

The most direct description of MST is a set of paths constituting MST edges. For the tree on fig. 17, the set is {a-b-c-d, d-c-b-e-f-h} in node-representation. Similar output is produced by GraphScape (see RESULTS ON MST TAB).

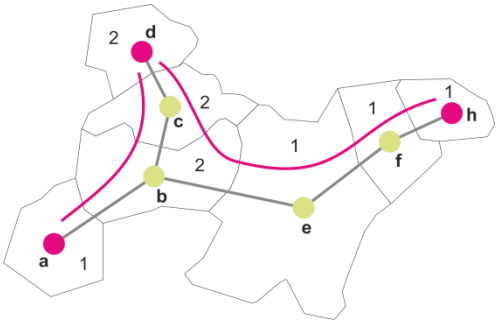


fig. 17 MST with transit density

A raw listing, although precise, does not advance our understanding of landscape as a whole. How do we transfer the information from MST to entire coverage? We might try to assign a Boolean value 1 to patches being crossed by a path and 0 otherwise, only to discover that many patches will have multiple paths crossing their territory, so more general indicator is necessary. **Transit Density** (TD) is the number of paths crossing a patch. Given node-representation P of shortest path of MST, and initial state TD_v=0, the algorithm for transit density is:

```
for all P ∈ MST
  for all v ∈ P
    TDv := TDv + 1
  endfor
endfor.
```

The algorithm leaves "untouched" nodes with their initial 0 values, so the result is valid for whole set V. By changing the reference from graph-oriented to patch-oriented, we obtain a new indicator which is easily mappable and comparable with other patch metrics.

A third method of revealing the information on MST is by **MST skeleton**: a straight line display of MST edges (see RESULTS ON MAP). This method is good in showing the extent and overall layout of MST, especially on complex coverages, but it does not tell the whole truth about MST composition. Two different trees may have the same skeleton and may only be distinguished by different TD pattern (compare fig. 17 and fig. 18).

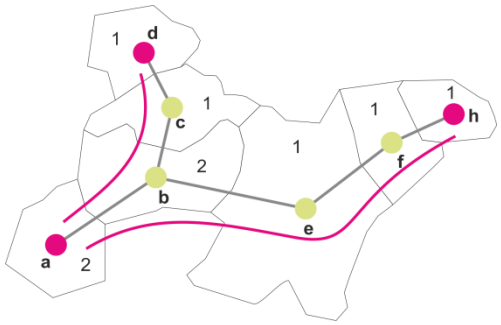


fig. 18 Another MST with the same skeleton but different composition.

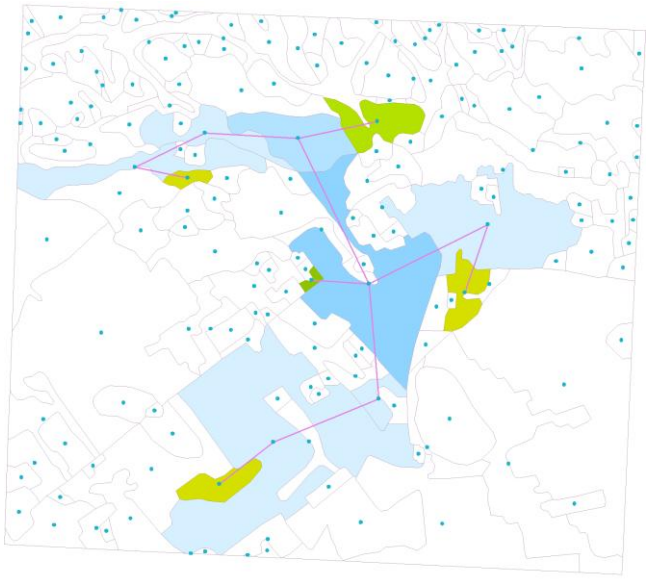


fig. 19 Transit density and skeleton lines on GraphScape map display

We may be also interested in a position of particular patch in MST or it's relation to the rest of MST. For every patch, a shortest edge (a path with least weight) is found in MST and two indicators: **Shortest MST Path Weight** and **Shortest MST Path Steps** are computed.

an indicator called is the weight of the

MST as optimization result

Minimum spanning trees were devised to optimize the wiring of electrical components and infrastructure network connecting a set of settlements. Based on most widespread interpretation of weight as a measure of cost or effort, MST is an optimum route system for a given set of nodes. It is

interesting to ponder upon the term "optimal" and possible beneficent of optimization. Two parties may be considered here:

- a mobile agent like electric current, car driver or roaming animal;
- a **provider** of a network connectivity[6] like printed circuit manufacturer, road authority or conservation authority.

In it's historical application, MSTs were used in minimizing electricity loss and this goal happened to be compliant with both parties, but it is not a general rule. A mobile agent seeks to minimize cost of traversal, a provider seeks to minimize upfront and running expenses for network connectivity. A provider situation is more complex, because it must not pursue cost-specified goal only: it has an obligation to maintain or improve connectivity. There are two approaches to include provider's perspective in the model. An **implicit provider's optimization** is already built into two-level GraphScape model structure. MST algorithm is proceeded by SP algorithm and MST is built of SPs. Shortest paths are clearly optimized for mobile agent, but minimum spanning tree is not influenced by mobility model. It is influenced only by it's definition, which guarantees connectivity and this is exactly the provider's obligation. It must be admitted that the weakness of any optimization based on MST is that connectivity is 0/1 property. There are no theoretical grounds to seek for better connectivity beyond it's minimal state.

Straightforward interpretation of mobility model explained earlier seems to be directed towards mobile agent, but in fact can be used more actively towards provider's goals. In **explicit provider's optimization**, we can try to include provider's goal: minimize expenses (we assume that connectivity goal will be included implicitly). The only way to do it is via weight function, which is constructed in mobility model. Since we have to fulfill both mobile agent and provider, mobility factors have to become a mix of environmentally and policy-oriented factors. Technically, we can proceed in two ways:

1. carefully craft a variable encompassing both goals and use it as a Patch Resistance (dropping Transfer Resistance entirely), or
2. use two variables: one for mobile agent's goal as Transfer Resistance and the other for provider's goal as Patch Resistance.

In both cases it is unlikely that provider's goal could be expressed as Transfer Resistance. It will most likely be coded into PCR for discrete variable and PIR for continuous variable.

A simplest example of provider's expense indicator, in the context of environmental protection, could be the area. The cost of establishing a protection over ecological corridor and the cost of excluding a swath of land from social activity is always related to the area. Other policy-related indicators like population (size, density), land ownership or zoning categories can be used. For instance if a corridor has to established on non-private ground, a providers expense C_p could be coded as 0 for non-private and "big enough" value for private ground. Then, for method (1), C_p should be mixed with mobile agent-oriented resistance R by addition, multiplication or any other formula, for instance $PIR=C_pR$. It is essential that the mixing formula and scaling of both factors yield proper balance, in line with the intention of optimization. Should the method (2) be used, C_p would be assigned to PIR straight away, and mobile agent cost would be expressed only by TR. In this case, we would have to rely on symmetric balance between factors, because multiplication is built into weight formula (DEF. 1).

Based on above guidelines, many policy constraints or goals may be included into optimization procedure by means of simple variables.

[6] Referred to as a property of a tree connecting a group of selected nodes, as opposed to *node connectivity* (a number of edges connected to a node)

Mobile agent ontology

MST may be considered an extension of a shortest path concept from two nodes to multiple nodes. In fact a minimum spanning tree for two nodes is equal to the shortest path. However, with increasing level of abstraction, the ontology of mobile agent changes.

The general rule is that MST, like any tree is **directionless** and this is why MST can be built in undirected graph. Though some MSTs will have a linear arrangement, many others will not. All trees with node count ≤ 3 are linear. Any tree with a node of connectivity ≥ 3 is be non-linear and cannot be traversed in single pass.

For shortest paths and linear trees, nodes can be ordered and followed in one direction, giving complete traversal of MST by single mobile agent (m.a.). Every m.a. can follow the same route and satisfy optimum: visit all nodes with minimum cost. For non-linear trees, it is impossible to visit all nodes without repeating certain parts of the walk. Some agents will visit parts of the tree optimally, and others may visit other parts optimally. From a viewpoint of a single m.a., either optimality or connectivity (ability to visit all nodes) is sacrificed. Since we cannot pinpoint any particular m.a. for which MST is optimal, we have to accept the **stochastic character** of mobile agent in MST. The optimum is achieved collectively: the total cost of movement approaches total weight when the number of traversals approaches infinity. If we include a notion of time to conceptual framework, this means that mobile agent is single entity, but given enough time to roam around MST. If time is not present, it means that mobile agent is a massive entity, with number great enough to move in all parts of MST.

Another conclusion is that total MST weight is only lower bound of total cost born by mobile agent. Actual cost are higher and depend on MST configuration.

Additional topics

Weight formulation under graph requirements

Good insight into *weight* concept is necessary before any work on a mobility model. The understanding of on weight coming from graph theory influences not only how weights are being constructed out of mobility factors, but also what are allowed values for these factors and how the final results should be interpreted.

The formulas for weight calculation given in def. 1 and def. 2 have been developed with graph use in mind. Many other, more sophisticated functions are conceivable as approximations of mobile agent's behavior, but not all of them would be suitable in the context of graph definition and algorithms. The requirements coming from Dijkstra SP and Prim MST algorithms[7] are the following:

1. All the information about mobility must be attributed to edge weights.
2. Weight measure must be valid locally and globally.
3. Weight values must be non-negative.

As may be seen in def. 1, two components involving PCR and PIR boil down to resistance averages for edge starting and ending node, so they may be transferred to edges. The remaining component (TR) is already an edge attribute, so the whole formula is effectively edge-related. The same is true for formula given in def. 2. In general, node characteristics are allowed as long as they are transferable to edges without loss of information or incurring spurious information.

A distinguishing feature of Dijkstra SP and Prim MST algorithms is that they try the first good solution (e.g. shortest edge) and stick to it as long as optimum condition is met. This class of algorithms is called **greedy algorithms**. To understand requirements (1) and (2), we will review shortest paths building in more detail. The algorithm builds path "attempts" in a series of steps. At each step, many candidate paths are elonginated and each path's summary weight is incremented (see fig. 20) by local weight value. There is no re-calculation of the whole path at any step of the procedure.

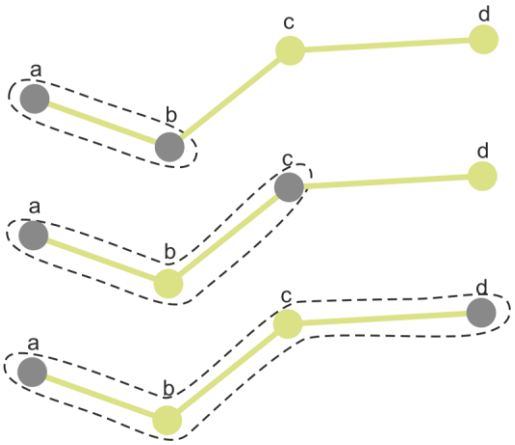


fig. 20 Steps of path weight summation. Terminal nodes dark.

The (local) weight of an initial step remains in the result until the end, and is included in the (global) weight of a winning path. We may be tempted to devise another strategy: first to find a shortest path and then correct its weight according to more sophisticated formula. Unfortunately this would not yield a shortest path anymore because we cannot influence the choice of the winning path beforehand.

The same reason precludes any formula employing two kinds of nodes: terminal (origin and destination) and intermediate. Let's inspect the role of certain node in the process of weight

[7] They are also necessary to carry out the mapping of shortest paths to reduced graph edges.

summation, for instance node *b* on fig. 20. On the first step, node *b* would be terminal, but on any subsequent step, it should be treated as intermediate. *A*-*b* weight calculation would need to be different for first step and for subsequent steps, and this is something both SP and MST algorithms cannot do. This kind of metric would be either valid locally or globally but not the both ways[8].

To explain non-negativity requirement (3), we will start with the observation of Dijkstra algorithm confronted with a **negative cycle**. If, during a course of path-building, an algorithm comes across a cycle of summary weight <0 (see fig. 21), then it is obliged to follow this cycle because negative value decreases total weight accumulated so far. Negative cycle wins with any positive edge and must be included in the path. After cycling, an algorithm is left at the same node and tries to add the same cycle to the path repeatedly, without ever reaching the terminal node.

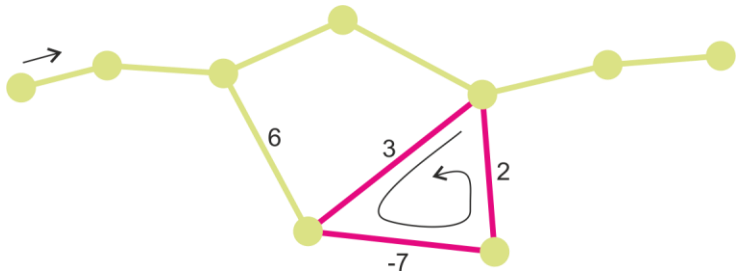


fig. 21 Negative cycle trap

An alternative Bellman-Ford algorithm detects negative cycle and stops gracefully, but also leaves the shortest path task uncompleted. Although negative values among graph edges do not necessarily imply negative cycles, there is high chance they will do, especially in sufficiently dense graphs, like mosaic-based. Therefore all negative weights must be excluded.

Weight measure under graph requirements

To be able to use shortest path and minimum spanning tree concepts, we must know what kind of weights is allowed as an input. A useful and simple way of telling this would be to answer the question “what scale of measurement is valid for weights?” Within a framework of scales developed by Stevens (1946), scales are defined by inherent relations of empirical objects and allowable transformations on corresponding numerical values. Transformations must keep inherent relations intact. Originally, scales framework has been used to build a numeric representation of certain empirical phenomenon. Here, it must be applied in the reverse way: starting from operations performed on numerical representation, we will try to find **secure** (withholding transformations) measurement scale.

First of all, nominal scale must be rejected because the only empirical operation defined by this scale is equation. It does not define > operation required by algorithms.

An **ordinal scale** is defined by inherent order relation and its ability to withstand any monotonic increasing transformation φ. This means that any pair of values a and b having the relation $a \geq b$ would still keep this relation after the transformation: $\varphi(a) \geq \varphi(b)$. A constant addition is among this kind of transformations, so to perform a test we will modify the formula from def. 5, by adding a constant C is to each elementary weight. The formula changes to

$$\sum_{i=1}^{n} w(e_i) + C \;=\; nC + \sum_{i=1}^{n} w(e_i)$$

def. 7 Weight constant addition test

[8] If actually used in shortest path calculation, this metric would result in all nodes along path being treated as terminal.

for a path made of n edges. Two paths of equal summary weights, made up different number of edges x and y respectively, would have their summary weights increased by different xC and yC. Weights observed on numeric axis would exhibit not only shifts but also swaps depending on C magnitude[9]. For instance, for $x>y$ and $C>1$, a sum for x edges would gain more than the sum for y edges. As illustrated on fig. 22, $C=10$ added to path weights results in red path being artificially shorter than blue path. In general, summary weight becomes spoiled with cardinality (number of elements) of a path, so SP algorithm may be deceived into choosing most simple paths instead of shortest paths.

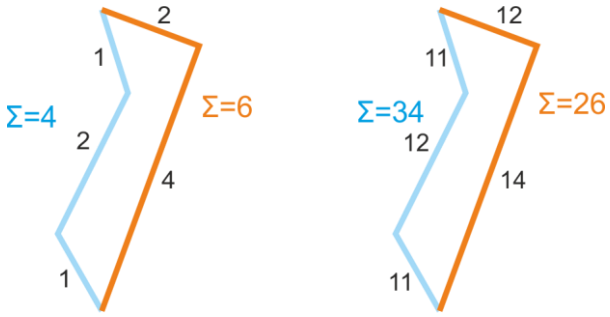


fig. 22 Constant addition results in wrong shortest path

The conclusion is that ordinal scale is not secure for SP definition and algorithm. It is interesting to note that it is still secure for minimum spanning tree. The reason is that spanning tree is, by definition, composed of all edges of the graph, so comparing any two spanning trees of the same graph, we have the same nC term in def. 7. The same observation is made by Roberts (1990), but on different grounds.

A similar test may be constructed for constant multiplication. An example on fig. 23 shows blue path shorter then red path before and after constant multiplication by $C=10$.

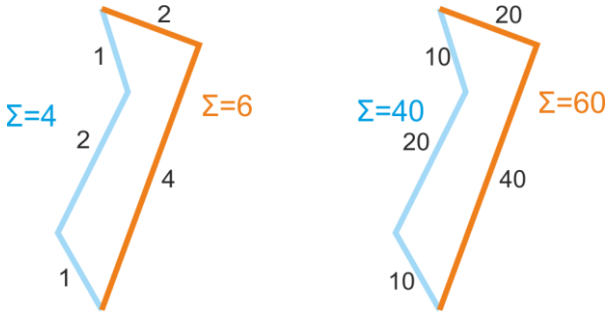


fig. 23 Constant multiplication results in proper shortest path

A formula describing summation of this kind is:

$$\sum_{i=1}^{n} Cw(e_i) = C\sum_{i=1}^{n} w(e_i)$$

def. 8 Weight constant multiplication test

This time, an extra factor C is brought before summation without n multiplier, so this transformation is not dependent on cardinality of the path. All paths will have the same C factor, which cancels out

[9] This observation gives an additional support for non-negative weights requirement of SP algorithm: negative weights cannot be neutralized by adding a constant.

during “>” comparison. This transformation preserves order. The applicability of constant multiplication is typical for a **ratio scale**. Actually, the condition defined for this scale is more demanding than necessary: it is said to preserve a ratio of any two values under constant multiplication so that $a/b = \varphi(a)/\varphi(b)$. It is not difficult to prove that meeting ratio condition satisfies also ordering condition.

Part II – Operation

Installation

GraphScape will run on Microsoft Windows platform versions 7 and higher. We expect it to run on any out-of-the-box machine regardless of particular configuration because the dependencies with other components are minimal. In particular, GraphScape does not use .NET, C++ runtime, Java, Flash Player or any other third-party components. No internet connection is required and no WWW browser components, plug-ins or extensions are involved.

GraphScape is distributed as a Windows ZIP archive numbered after the version (e.g. `3-1-3.zip`) with two files inside:

- `graphscape.exe` – the main executable,
- `graphscape.avm` – map template.

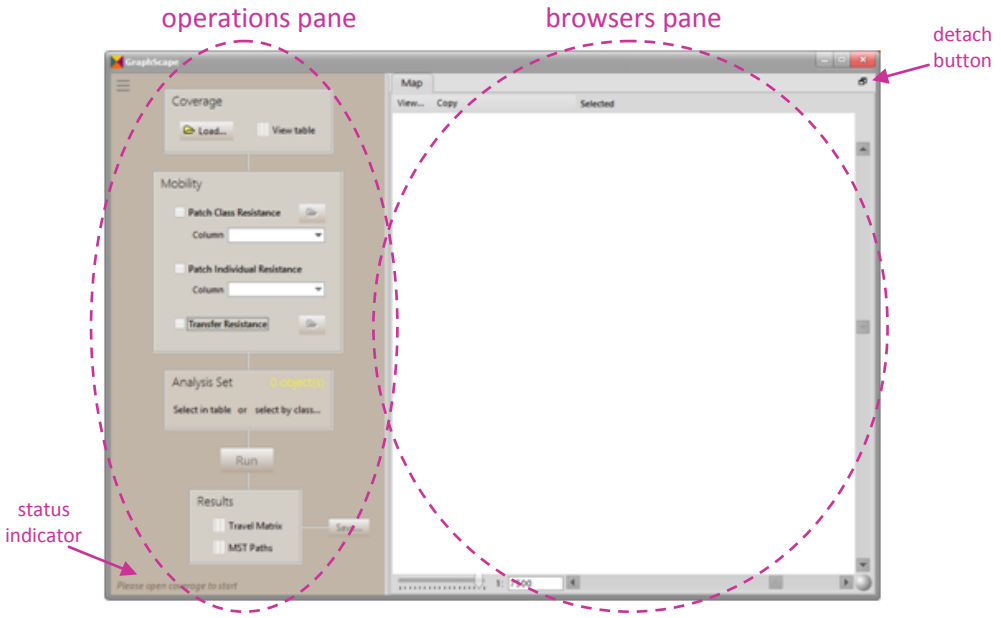
Additionally, `graphscape.pdf` (this manual) is available as a separate download.

There is no installation procedure proper. Please extract the files to any folder you like and double-click `graphscape.exe` to start working.

For more security-savvy users: it is wise to switch to administrator account and create a folder in a place with read-only access for ordinary users (e.g. under `Program Files` folder).

Program layout

On the left side of the main window you will see an **operations pane** with buttons and other controls for setting up and running a model. **Browsers pane** on the right side holds a map and a series of data tables. Initially only a map is visible, other browsers appear on demand. The browsers are displayed in a tabbed manner and if you need to see them side by side, press detach button – this will transfer current tab to a separate window.



The operations pane is designed to resemble a flowchart of a typical flow of work, with panels representing the following steps:

1. Loading a coverage
2. Defining Mobility
3. Selecting an Analysis Set
4. Running the model
5. Inspecting and saving the results.

While working, please pay attention to **status indicator** at the bottom of operations pane. This is where GraphScape reports currently performed operation. When the program is idle, status message guides you to the next step (e.g. “Open coverage to start”). During the time-consuming operations, the area is highlighted and the progress bar appears here along with estimated pending time.

Loading and viewing a coverage

Load... button in Coverage panel brings on a standard open file dialog box for selecting an ESRI Shapefile (SHP). GraphScape reads geometry as well as all shapefile attribute columns. After a successful load and topology build, the coverage will be displayed on the map. To view the contents of a shapefile in a table, press View table button in Coverage panel.

Coverage table

This table shows all the information GrapScape collects about patches. Shapefile data forms a major part, but not whole of a table. It extends from column number 2 and ends at geom.sub (number of shape parts) and geom.points (number of shape points) columns.

Map | Coverage | MST

▲ Obj	ASet	FIRST_NAZ	NUMER	AUTO_ID	AREA	PERIMETER	geom.type	geom.sub	geom.poin	transit d	patch cla	connect c	shortest
0		Vaccinio	9a	1	1836.348	174.415	Powierzch	1	8	0	-	4	∞
1		Peucedano	11a	2	5357.65	453.283	Powierzch	1	14	0	-	3	∞
2		Peucedano	11a	3	1698.266	252.291	Powierzch	1	10	0	-	2	∞
3	*	Querco-Pi	10b	4	84074.375	2409.668	Powierzch	1	65	0	-	12	∞
4		Querco-Pi	10c	5	25777.787	1136.333	Powierzch	1	50	0	-	4	∞
5	*	Querco-Pi	10d	6	19743.57	698.999	Powierzch	1	19	0	-	2	∞
6		Peucedano	11a	7	12035.863	459.882	Powierzch	1	9	0	-	4	∞
7		Peucedano	11c	8	18806.525	629.737	Powierzch	1	15	0	-	4	∞
8		Peucedano	11c	9	7224.051	414.646	Powierzch	1	14	0	-	3	∞
9		Peucedano	11c	10	3281.1	282.158	Powierzch	1	6	0	-	3	∞
10		Peucedano	11c	11	13250.455	770.563	Powierzch	1	25	0	-	5	∞

fig. 24 Coverage table

Column number 1 (ASet) shows which patches belong to **Analysis Set**, that is a group of patches chosen for MSP analysis. Included objects are marked with *. Please refer to BUILDING AN ANALYSIS SET chapter for further information.

Rightmost columns display various information related to patches produced later than on coverage load. They are described in RESULTS IN COVERAGE TABLE chapter.

The reason why original data is mixed with result data is that grouping them in one place gives better handling for sorting, visual inspection and export operations. Any original attribute can be combined with result data within one exported block. The most typical case is including object Id attribute with result columns to facilitate further processing in GIS software.

To sort a table, click on column’s header. To reverse the sort order, click again. Little triangle next to column’s title indicates sorting order. If you want to restore original order as read from the shapefile, click on a gray header column Obj.

To copy a block of data from a table, select cells and press Ctrl-C or right-click for drop-down menu. The data can be then pasted into Excel or GIS package.

The coverage table is click-synchronized with a map.

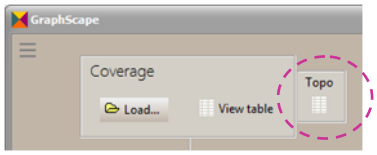
Shapefile requirements

1. Geometry projection Longitude-Latitude (unprojected geographic coordinates).
2. Geometry of polygon or multiple polygon type.
3. Continuous coverage.
4. Exact node matching on neighbour polygons. A shared border between two adjacent polygons must have:
   4.1. the same number of points,
   4.2. numerically equal coordinate values of corresponding points.
5. Single polygon should not exceed approximately 10000 points. This is not strict requirement, but larger polygons will considerably slow down map redraw and navigation.

Except for above technical requirements, It is advised that the coverage is as homogeneous as possible, that is contains patches of similar size and connectivity. Please inspect connect count column in coverage table. In case of planar graph these two variables are related: variability in size will always induce variability in connectivity.

Topology building and errors

Topology building phase starts immediately after a shapefile has been loaded and does not require any user intervention. As soon as topology has been build, an extra little pane shows up next to the Coverage pane. Press Topo button to view Topology Tab.



Except for rare cases of total failure, which can be caused only by insufficient memory, GraphScape will build a topology from any coverage, even not topologically correct. The reason is that the program cannot evaluate the severity of errors, especially in relation to whole coverage extent and complexity. Some situations which may indicate errors are reported, but these may actually be no errors at all, but particularities of a coverage accepted by the user. It is strongly advised to perform all necessary topology checks in native GIS prior to importing a coverage to GrapScape. Nevertheless, working on a new coverage imported to GraphScape should start with topology inspection.

The top area of Topology Tab reports two kinds of possible errors[10]:

- Solitary objects – patches with no neighbour,
- Orphan points – border points belonging to only one patch.

Solitary objects are patches that violate "continuous coverage" requirement (see SHAPEFILE REQUIREMENTS). Please switch to Coverage Tab and inspect connect count column - any patches with 0 value are considered solitary. A small number of solitary patches may be acceptable, provided that they shall not be included in Analysis Set. Unfortunately, solitary objects are only a special case of a general problem – graph segmentation. Isolated sub-graphs may be present even if they consist of

[10] The same message is included in final report.

non-solitary patches, for instance 3 patches forming an isolated group will all have connect count = 2. On other occasion, a coverage may be split into approximately equal parts with no connection between each other. GraphScape does not perform full contiguity check and this kind of error would appear only when Analysis Set objects would happen to belong to detached subgraphs. The user should prepare the coverage with this issue in mind.

Orphan points may indicate violation of "Exact node matching" requirement. Any point on any polygon boundary which does not match another point on neighbour boundary is considered an orphan. By this definition, it is clear that a polygon mosaic may have the orphan points on its outer boundary, so the presence of orphans does not constitute an absolute error. The presence of orphans on the inner borders of a coverage most likely signals an error. It is usually related to the history of coverage preparation, especially merging the coverage from parts of different projections or different digitizing source or method.

Orphan points are always accompanied by the gaps in shared borders (see fig. 27). Unfortunately, the program is not able to produce anything like a "list of gaps" or handle them in any automatic way. Depending on the location within a coverage, lack of shared border may be classified as error or not. This decision must be made by the user.

To see the location of orphan points, switch to Map Tab and turn on View > Diagnostics and watch for triangle symbols on boundaries.

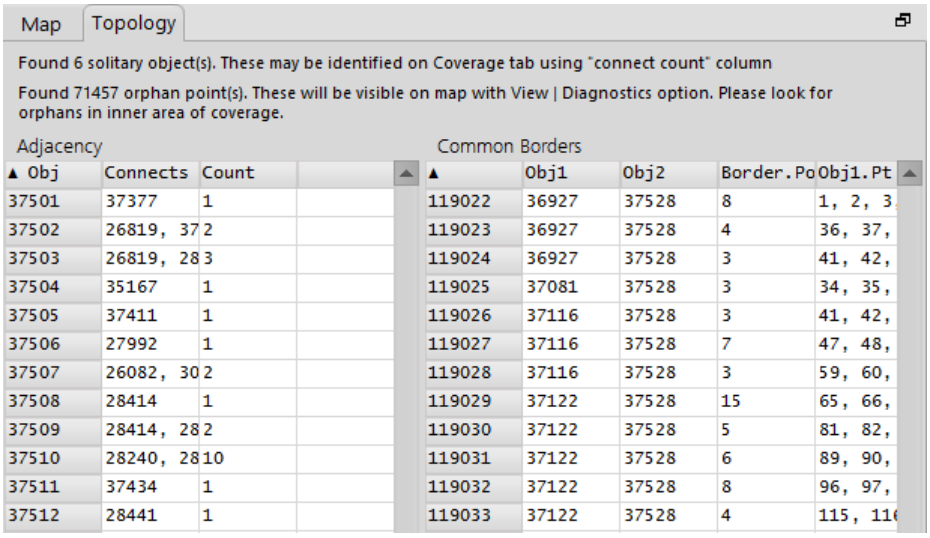
Modelling with imperfect coverage

Graph segmentation errors (including solitary objects but also bigger detached parts of coverage) will show up when building the MST (Minimum Spanning Tree).Two cases must be considered. If objects selected to the analysis belong to different subgraphs (corresponding to different detached parts of coverage), there will be no path (infinite distance) between some of them and an obvious error will occur. More difficult to diagnose is a case when segmentation does not break down MST, that is when Analysis Set nodes belong to the same subgraph. MST found this way may be distorted. To understand this, it is necessary to go back to the generation of shortest paths, which later become elements of MST. A path generated in segmented graph may be longer than the one generated in complete graph, because there are less edges to choose from. Simply put, missing connections result in missing opportunities to route a shorter path.

Shared borders gaps result in shorter then true shared borders between adjacent patches. In severe case this may result in complete lack of shared border and consequently – no connection. This kind of error is unacceptable for MST modelling. In less severe case, false length of border can influence the shortest path generation and, consequently – MST structure. This may happen only under step model (see STEP MODEL WITH BORDER FACTOR). Other than that, gaps may be tolerated.

Topology tables

Topology building results may be viewed in tabular form in two tables. They are of interest for those who want to investigate the problems in depth before sending a coverage back to GIS for repair.



Obj	Connects	Count
37501	37377	1
37502	26819, 37	2
37503	26819, 28	3
37504	35167	1
37505	37411	1
37506	27992	1
37507	26082, 30	2
37508	28414	1
37509	28414, 28	2
37510	28240, 28	10
37511	37434	1
37512	28441	1

	Obj1	Obj2	Border.Po	Obj1.Pt
119022	36927	37528	8	1, 2, 3
119023	36927	37528	4	36, 37,
119024	36927	37528	3	41, 42,
119025	37081	37528	3	34, 35,
119026	37116	37528	3	41, 42,
119027	37116	37528	7	47, 48,
119028	37116	37528	3	59, 60,
119029	37122	37528	15	65, 66,
119030	37122	37528	5	81, 82,
119031	37122	37528	6	89, 90,
119032	37122	37528	8	96, 97,
119033	37122	37528	4	115, 11

fig. 25 Topology tables

Adjacency table shows a list of patches with three attributes:

- patch number (column Object) – same as in original coverage,
- a list of adjacent patches numbers (column Connects),
- a number of adjacent patches (column Count).

Shared borders table shows the following attributes:

- shared border number,
- Obj1 and Obj2 columns – "left" and "right" patch for shared border,
- number of constituent points (column Border. Point Count)
- a list of constituent points (column Obj1.Pt) – with the numbering relative to Obj1 (e.g. when Obj1=166 and list={34, 35, 36} then shared border is made of points 34, 35 and 37 of polygon 166)

Shared borders table is click-synchronized with a map, but selected border will be highlighted only under diagnostic view.

Please note that cells containing very long lists in both tables will appear blank on screen, but will copy to the clipboard with full contents. Copying too many rows can bring importing program (esp. Excel) to its knees!

Working with map

GrapScape Map Tab provides basic GIS capabilities for viewing and analyzing imported coverage and modelling results. It does not provide any editing and user-defined symbology, so any serious work should be carried on with true GIS at hand.

Normal view

At the default state map is configured to display the following layers:

- Coverage polygons with thin, purple borders, normally filled white, except:
  - patches belonging to Analysis Set – filled translucent yellow,
  - patches belonging to MST – filled gradually from pale to dark blue, according to Transit Density value.

- Nodes: point representation (centroids[11]) of patches: blue dots,
- MST skeleton – thick magenta lines spanning between nodes.

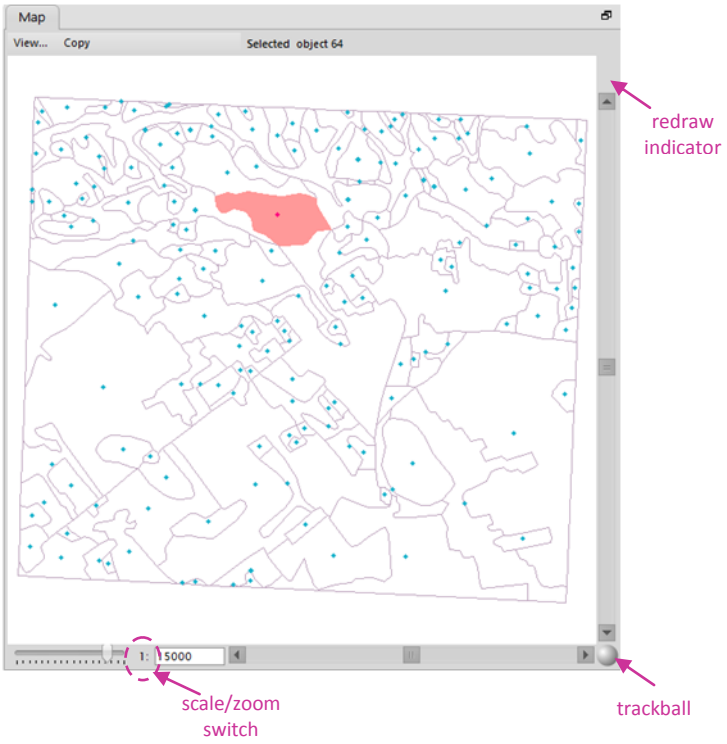


fig. 26 Map with navigation controls. Initial view after loading a coverage.

In initial view (after loading a coverage), there will be no Analysis Set or MST-related coloring or objects.

You can adjust map layers visibility with View... button at the top of map frame. Available options are:

- Coverage,
- Nodes + MST skeleton,
- Diagnostics (see DIAGNOSTICS MODE).

Big coverages take long time to draw. When a map is being refreshed you can see a red light in upper-right corner of map frame (fig. 26). Wait until red light is off to be sure that map contents is valid.

Navigation

Map panning can accomplished by:

- scroll bars,
- mouse on trackball (see fig. 26): press left button and drag,
- mouse on map: press roll and drag.

GrapScape map responds to zooming in two ways. In default state zooming results in scale change. Lines and symbols on map keep their widths and sizes. When scale/zoom switch (initially labeled 1:) is pressed, map enters optical zoom mode resembling working with a magnifying glass. Use it if symbology seems too heavy or too thin for your needs.

Zooming can be accomplished by:

[11] Centroids are generated as gravity centres of polygons, however when gravity center falls outside of the polygon, it is moved to the nearest point on the boundary.

- track bar in left-bottom corner,
- numeric value[12],
- mouse on trackball: left button zooms in, right button zooms out,
- mouse on map: roll forward to zoom in, roll back to zoom out.

Please note that after working with some non-map controls, moving mouse back to map does not re-activate mouse roll operations. Click on Map Tab then.

Selecting objects

Three kinds of objects can be selected on map with a mouse-click:

- patches,
- shared borders (only in Diagnostics mode),
- MST skeleton elements.

Only one object can be selected at a time. Selected object is highlighted with red color.

Map is click-synchronized with appropriate tables: Coverage table on Coverage Tab, Shared Borders table on Topology Tab and MST Paths table on MST Tab. You can browse through objects on map and observe their properties in tables. To make use of this feature it is best to detach tabs into separate windows. Synchronization is two-way: selection in a table causes selection on a map too.

Selecting objects is not the same as building an Analysis Set, but is often the first step in this process.

Copying map

Press Copy button at the top of map frame. The contents of the map will be copied to clipboard as a raster image with 200 DPI resolution. The size of a picture is equal to currently selected paper size of your printer.

This is a quick way of sending analysis results to general-purpose software. If you plan to carry on more demanding work on the results, vector-based export will be more suitable (see SAVING RESULTS).

Diagnostics mode

In this view, additional symbols are displayed:

- orphan points (orange triangles)
- common boundaries (thick green lines)

Additionally, selected objects are shown with more detail (border points are displayed) and are highlighted with a red frame rather than filled.

[12] Minimum predefined scale for map is 1:2500. If you need larger scale it must be entered numerically.

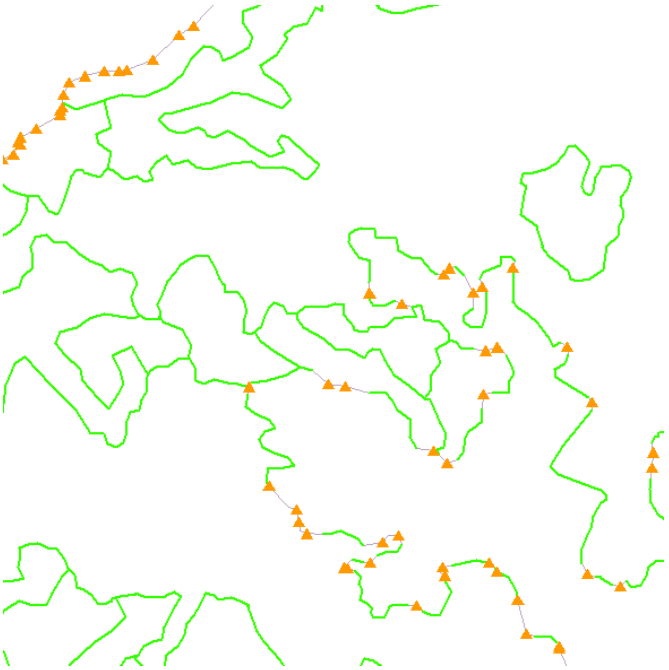


fig. 27 Map diagnostic view with orphan points (triangles), shared borders (thick green lines) and underlying original coverage borders (thin purple lines)

The visual contrast between shared borders and underlying original borders helps to find gaps in shared borders. In perfect situation, original borders should be completely covered with shared borders except for the outer boundary of coverage. On fig. 27, the top-left string of orphan points lies on the outer boundary, while the right-bottom group of orphans must be attributed to digitizing errors. Most notorious errors come from minute displacements of border points because they are hard to notice during digitizing. In this case, what you see as singular orphan points are actually pairs of points.

The gaps are always accompanied by orphan points. In larger scale, it is more convenient to look for orphan points which will guide you to the problem area. Please note, that although the gaps look like clickable objects, they are only borders of underlying patches.

Defining Mobility

GraphScape uses three Mobility factors to determine the cost of travel between patches. These correspond to three checkboxes in Mobility panel (see fig. 28):

- Patch Class Resistance (PCR),
- Patch Individual Resistance (PIR),
- Transfer Resistance (TR).

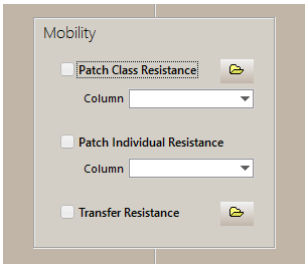


fig. 28 Mobility panel

Be sure to read about mobility formulations in **MOBILITY MODEL**, because this chapter will only help you with technicalities.

Any mobility factor checked will be in use in further modelling. If all checkboxes remain unchecked, a fallback type of mobility called **step model** will be used (see **STEP MODEL WITH BORDER FACTOR**).

After checking appropriate resistance type, you must provide a source of resistance values.

- For **PCR**, you select a text file with two or more columns. First column has to contain a class-identifier values (usually textual, but numeric is also accepted). Second column contains numeric resistance values. The file may have ore than one resistance column to facilitate quick switching of alternative versions of resistance. First row of the file must contain names of columns. A complete file content looks like this:

```
c_id  mam    rept
9a    0.2    0.6
11a   0.25   0.66
10b   0.1    0.06
10c   0.3    0.006
```

  After picking a file, please select one of the columns from a drop-down list.

- **PIR** values are provided as an extra column in a coverage. Please select an appropriate shapefile column from a drop-down list.
- For **TR** values, you select a text file with a matrix layout. The file must have the same number of rows and columns. First row and first column must contain class values, the rest of the file contains resistance values. Upper-right half values must be filled, lower-left half has to contain empty cells, for example:

```
c_id  9a     11a    10b    10c
9a           0.5    0.2    0.3
11a                 0.7    0.9
10b                        0.9
10c
```

  Please note that the empty space in the last row is composed of 4 empty cells.

All negative resistance values (either PCR, PIR or TR) are converted to infinite resistance, so they have the effect of completely blocking the movement. Technically, value 1E+30 is assigned as user-defined infinity to distinguish them from topology-related infinity 1E+306. For display purposes, user-infinity is just a numeric value, while topology infinity is "∞".

PCR table and TR matrix class values should match values in the coverage. A non-matching value can indicate typing error or that the table/matrix is designed to work with different coverage. In a Report Logbook (see **RESULTS ON MST TAB**) you will find will an appropriate comment with missing values listed.

Preparation of resistance files

Files may be prepared in basic text editor like Notepad, saved as text from Word document or from Excel spreadsheet. Make sure they are TAB-separated (have a TAB character used to separate items in row), but this is the default option when data is formatted as a table. For numeric values decimal sign can be either comma or period.

Building an Analysis Set

Analysis Set is a user-defined set of patches which is the input to MST modelling. In Analysis Set Panel you will find a current size of a set (e.g. "0 object(s)"), along with two buttons activating two ways in which a set can be built.

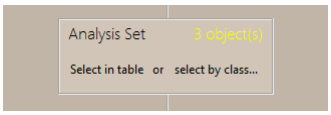


fig. 29 Analysis Set panel

When pressed, Select in table label simply directs you to Coverage table (see ABOVE). Here, the first column named ASet, shows patches belonging to an Analysis Set with an asterisk "*". To add a patch to the AnalysisSet, click on a row and press right mouse button and use Analysis include command. Multiple patches can be added when a block of rows is selected. This, combined with sorting, allows you to include patches of common characteristics or similar value (e.g. size, connectivity). Other commands available are: Analysis exclude, Analysis toggle and Analysis clear all. Any changes to the Analysis Set are reflected in the panel and on map:

fig. 30 Map with 5 patches in Analysis Set

An opposite way of working is to select patches on map (see SELECTING OBJECTS) and toggle ASet flag in coverage table. It is most convenient then to detach coverage table and place it side-by-side with map.

Another way of building an Analysis Set is available by pressing Select by class label. This method is suitable for categorial data, e.g. a type of patch. In a dialog box (fig. 31) you can choose a column of a coverage and see all column's values listed along with frequencies (object counts). Please note that any previous manual selections will be cleared.

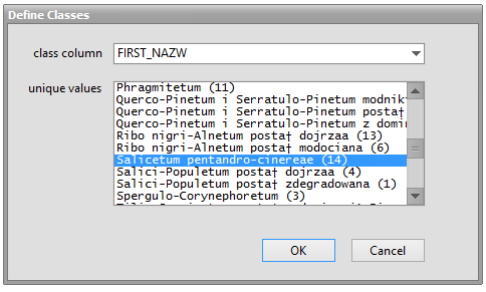
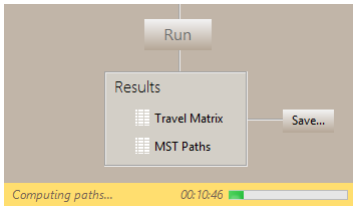


fig. 31 Selecting Analysis Set by class

With both methods, the columns of a coverage largely dictate your options to select patches in systematic way. It pays to have as rich coverage as possible.

Running and viewing results

Run button starts computation performed by Shortest Paths Engine and MST Engine and leading to final results.



The program runs multithreaded and the program window remains responsive for most of the time so it is possible to browse data or navigate map. If you want to cancel an operation, click on the progress bar[13].

Immediately after the run operation is completed, results appear in several places:

- on MST tab
- on map
- on Travel Matrix tab
- in Coverage table

To open MST tab or Travel Matrix tab press appropriate button in Results panel below the Run button.

Results on MST tab

MST Tab is divided into two areas. The upper part is a MST Paths table, the lower – Report Logbook.

MST Paths table gives a detailed breakdown of minimum spanning tree. Every row describes a single path, which constitutes an edge of MST. Terminal nodes are listed in Obj1 and Obj2 columns. A complete list of nodes is in Path column, followed by summary Weight and No Steps. Nodes are numbered sequentially, starting from 0, like in Coverage Table. Whole table or selected cell block can be copied to the clipboard with a right mouse click. Copying may be the only way to reveal paths too long to display in a table cell.

Minimum Spanning Tree Paths

▲	Obj1	Obj2	Path	Weight	No Steps
0	74	2084	74, 44, 21.	9999528	2
1	2084	2104	2084, 1931.	9999888	2
2	74	5279	74, 44, 34.	9999034	5
3	5279	25423	5279, 2877.	9998496	8
4	25423	25253	25423, 301.	9999063	2
5	25423	37390	25423, 371.	9999346	2
6	25423	27765	25423, 371.	9999386	2
7	25423	25316	25423, 371.	9999394	2
8	25423	27662	25423, 371.	9999437	2
9	25423	28122	25423, 371.	9999452	2
10	25423	25174	25423, 371.	9999457	2
11	25174	24985	25174, 251.	9998590	2
12	25423	19225	25423, 371.	9999555	2
13	25423	20613	25423, 371.	9999567	2

fig. 32 MST Paths list

A report block is added to **Report Logbook** after every run and remains with consecutive number, until Clear Results button is pressed. A report block starts with general information, which describe coverage and analysis parameters (not very interesting by itself but useful when archived). Then, two kinds of statistics follow:

- weights statistics (count, sum, mean and standard deviation) of paths and
- the same statistics but with steps as a weight measure.

[13] Certain steps in computation sequence cannot be interrupted immediately (e.g. memory allocation for big data sets). A break request will be delayed but still honoured.

The text may be easily copied to other document via clipboard (right mouse click). Statistics data is TAB-separated and can be easily converted into tables in importing program. Below is an example of the report:

```
GrapScape 3.1.3

Coverage C:\Sys\Run\GraphScape\Data\frag_2.shp
Topology finished in 00:00:11, peak memory 209MB


Run 1 ************* 21/03/2016 17:16:35 ************************
completed in 00:03:23, memory now 416MB
Coverage C:\Sys\Run\GraphScape\Data\frag_2.shp: 32527 objects
Analysis Set by column CODE_06 value 222: 243 objects
Travel Matrix from steps+border factor

MST Paths Weight Statistics
-------------------------------------------
Count      Sum     Mean    Std.Dev.
242        565.978 2.33875 0.931998

MST Paths Steps Statistics
-------------------------------------------
Count      Sum     Mean    Std.Dev.
242        566     2.33884 0.932017
```

Partial MST

Sometimes, single MST cannot be determined due to segmentation. MST breaks into two or more parts (a special case may be single-node part) because would-be edges connecting these parts have infinite weight. The reason may be either

- coverage topology error (no shared border between parts of coverage),
- infinite weight resulting from negative PCR, PIR or Transfer Resistance values in combination with particular node position.

In these cases, the report will include the following message:

```
Complete MST cannot be found: XXX edge(s) missing
```

GraphScape will build a **partial MST**, actually a forest of trees consisting only of valid edges (paths with finite weight). Infinite-weight edges will be excluded and XXX will be the number of missing edges. Strictly speaking, there are no “missing nodes”, because even a single node is proper component of a forest. If you are interested in finding detached nodes, please review COVERAGE TABLE for nodes with `Analysis Set = TRUE` and `Transit Density = 0`.

fig. 33 Partial MSTs with 2 and 3 parts

It is helpful to know if MST breakdown is related to coverage topology error. GraphScape will perform a quick check of topology and if no connection is found among Analysis Set nodes, it will come up with the following message:

```
Isolated sub-graph detected (coverage topology error). Suspected edge is
XXXX-YYYY
```

XXXX and YYYY are node numbers, which can be looked up in the coverage table. The edge identification is heuristic and may miss the real culprit on ocassion.

Please note, that

- both reasons may present at once,
- topology errors are checked only between Analysis Set nodes.

The most effective method to spot down topology error is to switch temporarily to step model (uncheck all mobility factors). This eliminates all user-assigned infinite resistance values from the model.

Results on map

MST is difficult to display on a map and two cartographic methods are used side by side:

- cartogram of Transit Density – visible as different shades of blue over the patches,
- MST skeleton – thick magenta lines spanning between nodes.

A patch with a slightest blue is crossed by one path, the darker the color – the more paths. The blue is transparent to allow Analysis Set objects' yellow to come through. This way terminal patches of MST (always crossed by at least one path) receive shades of green.

MST skeleton helps to follow paths visually. It is composed of straight lines connecting patch centroids. Of course, in case of complicated patch background, the lines will sometimes stray or cross neighboring patches.

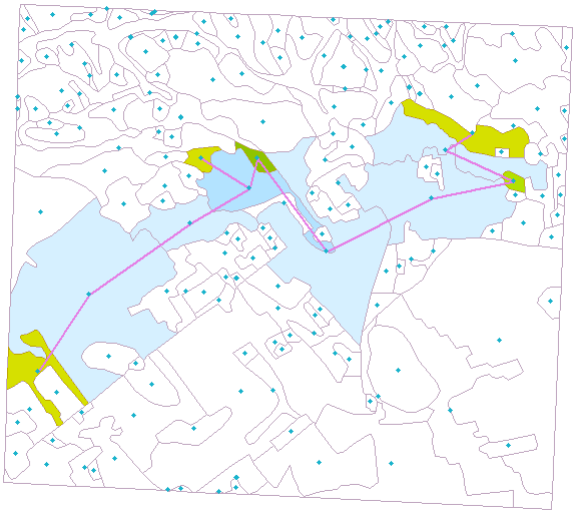


fig. 34 Map with results. See MST skeleton (magenta lines) and transit density (blue color over patches).

Quite often the paths are partially overlaid and difficult to tell apart. This is natural, because they tend to follow the same patches, especially those close to important key nodes. Also, it may be difficult to recognize how far a single path extends. Remember then, that it is possible to click on any skeleton line and get it selected not only on map but also in the MST table. The other way around is to select a row in a table and see it highlighted on map. Two methods combined will give you full flexibility in browsing through a MST.

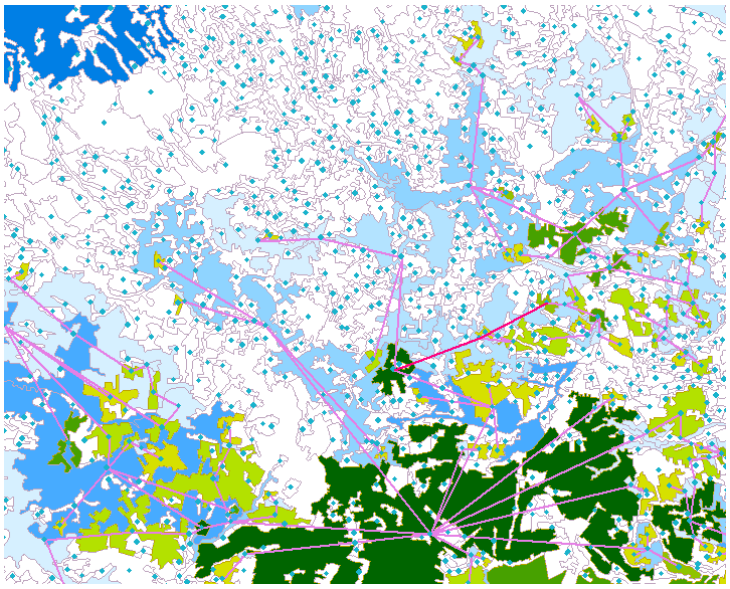


fig. 35 More complex results on big coverage. Single path highlighted.

Travel Matrix

Travel Matrix table gives a glimpse of low-level data directly used as an input to shortest paths computation. It may be useful for verification purposes, because it shows the cumulative effect of all mobility factors. The other reason for interest would be to gather raw data for some other external modelling.

Map | Travel Matrix

▲	0	1	2	3	4	5	6	7
0	–	∞	∞	∞	∞	∞	∞	∞
1	∞	–	∞	∞	∞	∞	∞	∞
2	∞	∞	–	∞	∞	∞	∞	∞
3	∞	∞	∞	–	∞	0.9993961	∞	∞
4	∞	∞	∞	∞	–	∞	∞	∞
5	∞	∞	∞	0.9993961	∞	–	∞	∞
6	∞	∞	∞	∞	∞	∞	–	∞
7	∞	∞	∞	∞	∞	∞	∞	–
8	∞	∞	∞	∞	∞	∞	∞	∞
9	∞	∞	∞	∞	∞	∞	∞	∞
10	∞	∞	∞	∞	0.9995592	∞	0.9993179	∞
11	∞	∞	∞	∞	∞	∞	∞	∞
12	∞	0.9993562	∞	∞	∞	∞	∞	∞
13	∞	∞	∞	∞	∞	∞	∞	∞

Please be aware that columns and rows are filled with valid data only for pairs of objects in Analysis Set. Other rows/columns may also contain some valid cells, but this happens only when a given pair connection is a part of some shortest path, so finding ∞ value in these rows/columns does not definitely mean lack of connection. If your goal is to obtain a complete travel matrix, you must run a model with all objects included in Analysis Set.

By sorting a valid column (click on column header), you can view all connections of a single object.

As this matrix is potentially very big (n^2 cells), be careful when selecting the whole table for copying to the clipboard.

Results in Coverage Table

geom.point co	transit density	patch class resistance	connect count	shortest MST path
19	7	1	4	2
67	4	1	14	∞
44	2	1	11	2
14	2	1	5	2
43	2	1	7	∞
50	2	1	10	∞
66	2	1	11	∞
12	2	1	3	2

A number of columns of Coverage Table are related to results and become filled with data only after the model run. These are:

- transit density – a number of MST paths crossing the patch (see RESULTS),
- patch class resistance – a resistance value assigned from mobility (see MOBILITY FACTORS)
- connect count – number of connecting patches
- shortest MST path weight – a summary weight of a shortest path connecting the patch with the rest of MST.
- shortest MST path steps – a number of steps of that path.

Please note that shortest path is always found based on weight and not number of steps.

Clearing results

Clear Results button on the right side of Report Logbook clears the contents of the logbook and MST Paths table above. Transit density coloring and MST skeleton lines disappear from the map. Transit density and Shortest MST Path columns of coverage table are cleared.

Also, result columns gathered through all runs and prepared for saving in shapefiles are deleted.

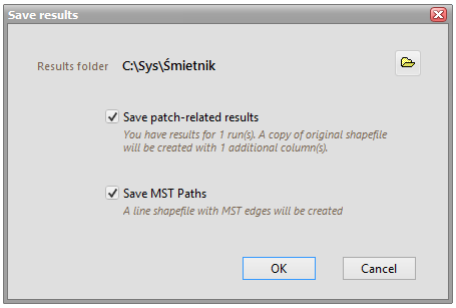
Saving results

GraphScape does not directly save text or tabular data to files – you must use a clipboard for transfer. It is easy, because all tables provide uniform way of doing this. You can select a range of cells and press Ctrl-C key or right-click for drop down menu and use Select All or Include headers options. Report Logbook behaves like any other simple editor and it's contents may be copied too.

Map data can be saved in two ways. Press Copy button on upper part of map frame to copy map image in 200 DPI resolution to the clipboard. For more flexibility, vector format GIS data can be exported to shapefiles.

Saving to shapefiles

Press Save button next to Results pane to see the following dialog box:



Since there are multiple files to be saved (depending on the number of runs), GraphScape approach is that you provide only the folder for storing the files, and the filenames are assigned automatically.

There are two groups of result files available.

Patch-related results shapefile is an exact copy of original coverage, with additional columns for

- transit density (named `td<number>`),
- shortest MST path weight (named `smstpw<number>`),
- shortest MST path steps (named `smstps<number>`),

With every run, 3 new columns are added, but only one file is created under the name `<coverage-name> Results.SHP`.

MST Paths shapefile is an equivalent of MST Paths table (see **RESULTS ON MST TAB**), but with geometry added. Geometry is of linear type and contains skeleton lines, just like visible on the map. Every run is saved in a separate file and the names for the consecutive files have the following syntax: `<coverage-name> MST Run <number>.SHP`.

Actual saving takes place after OK button is pressed. Existing files are overridden without warning, but as long as the results are still in Report Logbook, next save will include them anyway. If you want do discard previous runs, press Clear Results.

Note: infinity values are saved as 1E+19.