

Minimum-cost network-layout tool with multi-sinks and multi-sources

Version 4: 24/01/2025

Owner and author of the description

Petra Heijnen p.w.heijnen@tudelft.nl

1. Introduction

Networked infrastructures (gas pipes, water pipes, electricity cables, glass fibre, (rail) roads) form the backbone of our society as they provide essential utilities and services. In a densely built and highly urbanised environment, such as the Netherlands, the roll-out of new networks encounters physical or legal boundaries that make the planning of such networks a problematic task. Furthermore, suppose the network has to connect several processes from independent organisations. In that case, the actual commitment of these parties and the network capacities they require can remain uncertain for a long time. This uncertainty makes the planning process cumbersome and requires tools that can deal with this uncertainty.

The need for planning tools is becoming more prominent as, for example, the energy sector develops several initiatives that require building new networks. Increasing awareness of future energy scarcity and the adverse effects of electricity generation have led to projects in which pooled demand or supply of energy leads to economies of scale. Examples of such networks are heat pipelines to rationally deal with excess heat in industrial areas, biogas pipelines that collect output from individual farmers or the hydrogen backbone that can partly use the existing natural gas network.

The common characteristics of these networks are:

1. The network transports a commodity from one or multiple sources to one or multiple sinks. Representative examples are the collection of biogas from farms and CO₂ from industry or the distribution of heat, natural gas to houses and hydrogen to industry.
2. The network building cost depends mainly on the pipeline length (e.g. digging cost) and the pipeline capacities (e.g. materials cost).
3. In the exploratory or design phase of the project, the project owner might not know all participants nor the capacities they require. Moreover, the demand and supply of the different participants may vary over time.
4. The area around the network poses limitations on the possible routing of pipelines. They have to follow streets or avoid existing buildings. Or there are obstacles like rivers or mountains or zoning rules (e.g. protected natural areas).

Network design problems have been intensively studied in literature before. However, none of the existing algorithms covers all of the issues mentioned above. Network owners can use the method discussed here to find a minimal cost network that connects multiple sources with multiple sinks taking into account the demand and supply patterns over various time steps. Depending on the time, nodes can be suppliers at some times and consumers at other times. Routing can (if needed) be limited to

specific connections or around obstacles in the area. Part of the network can already exist, and new connections can use existing ones if sufficient capacity remains and the reuse is cost-effective.

1.1. Model assumptions

Before running the model, you need to be aware of the following modelling assumptions.

- The algorithm looks for a minimum-cost network without any redundancy. So, the new connections in the final and intermediate results will have a tree (or forest)-topology.
- Existing connections however may contain loops and do not need to have a tree (or forest)-topology.
- The algorithm uses a generalized cost function, taking into account (among others) the length and capacity of the new connections. This cost is by no means a realistic estimation of investment cost and is only used to compare the different topologies to find the one with the lowest cost.
- The nodes to be connected in the network can have different roles. They can be production, consumption, throughput or storage nodes.
- A node can be producer at one time and consumer at another.
- The demand and supply of the production and consumption nodes can be given for different time steps.
- The capacity of all connections in the network will be determined in such a way that the network can maximize supply in each time step, but not more than that.
- If in one time step, there is a surplus of supply, this can be stored in storage nodes. This storage can be used in another time step when there is a shortage of supply. This only makes sense if the time steps succeed each other over time.
- Storage nodes need to have a current storage level and a maximum capacity.
- If there are no storage nodes or the capacity of the storage nodes is not sufficient, then remaining supply is just lost. However, one can include a connection to an infinite backbone that can always supply remaining demand or take up a surplus of supply.
- All connections in the network can be used in two directions.
- The final minimal-cost-network found by the model might be sub-optimal since finding the optimal network is hard, and we use heuristics to end up with a near-optimal network in reasonable time.

2. Installation of the software

The software is written in Python 3. To use the model, you should install Anaconda. You can find Anaconda on <https://www.anaconda.com/download/>. Choose the latest Python 3 version available. We will use the Python-environment Spyder that comes with Anaconda.

2.1. Extra Python modules

Anaconda comes with a long list of packages. However, you need to add the extra modules below to use the model. Type in the Anaconda Prompt:

- `pip install pyvisgraph==0.2.1`

- `conda install shapely==2.0.5`

And any other module that is called for but not automatically installed on your laptop. These will be mentioned in the Python console when you try to run the code.

2.2. Model download

You can download the model on <https://gitlab.tudelft.nl/pheijnen/optimal-network-layout>. Download the directory *ONLT*, with all files included, on your laptop.

2.3. Input preparation

As input, the model uses an Excel file with a specific format. There is one sub-directory '*case_study*'¹ in your main folder '*model*'. This directory contains the Excel files '*input_data_simple.xlsx*' and '*input_data.xlsx*' for a working example. These input files can be adapted or copied.

The Excel file might contain several worksheets that will be discussed below.

Worksheet 'terminals'

The first worksheet is called 'terminals'. This worksheet is obligatory. Without this worksheet the ONLT will not work.

In graph theory, the term *terminals* refers to the nodes that need to be connected in the network. In our case, these are the production and consumption nodes. This worksheet contains the labels of the terminals (from 0 to N) (column 1) and the coordinates of the terminals (column 2). You can add extra nodes by adding extra rows to this worksheet and label the rows with the consecutive numbers. You can also delete nodes by deleting the last rows. Please assure that the terminal labels are consecutive numbers from 0 on.

The next columns give the demand and supply of the terminals in each time step. If a node is a consumer, it absorbs from the network, indicated by a *positive* number. If a node is a producer, it supplies to the network, indicated by a *negative* number. Each terminal should have at least one supply or demand unequal to zero in one of the time steps to be connected to the network. You can add or delete time steps. If you delete a time step you should delete the entire column, not only the demand or supply values.

The Excel file '*input_data_simple.xlsx*' contains the minimum amount of input the ONLT can work with (see Figure 1). This example has 8 terminals, with a demand or supply in one time step. The volume-unit and time-unit you choose for this demand and supply is your own choice, but should be consistently used for all other relevant input.

¹ You may give the folder and the input file different names. Be sure to adapt these names in the file '*user_interface.py*' from where you run the model.

	A	B	C	D
1	terminals	coordinates	timestep 0	
2	0	(35,43)	-4	
3	1	(10,41)	-5	
4	2	(83,15)	2	
5	3	(11,8)	-5	
6	4	(46,66)	2	
7	5	(72,3)	-4	
8	6	(17,81)	-5	
9	7	(88,45)	-1	
10	8	(93,3)	20	
11				
12				
13				

Figure 1: Worksheet 'terminals' from 'input_data_simple.xlsx'

Do not change the name of this worksheet or the given format.

Worksheet 'storage' (optional)

The next worksheet is called 'storage'. In this worksheet, you can indicate storage nodes in your network. Storage nodes have a location ('coordinates'), a current storage level ('amount') and a maximum capacity ('capacity').

In case you want to connect the network to an 'infinite' backbone, you can set the capacity of this fictive 'storage' node and the current amount in storage to a very large number.

The Excel file 'input_data.xlsx' contains a worksheet storage (see Figure 2) with two storage nodes. Both are empty at the start and have a capacity of 50 in the same volume-unit as used for the demand and supply of the terminals.

	A	B	C
1	coordinates	amount	capacity
2	(50,0)	0	50
3	(80,80)	0	50
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			

Figure 2: Worksheet 'storage' from 'input_data.xlsx'

Worksheet 'routing_network' (optional)

If the network connections can only follow predefined paths, you should fill the worksheet 'routing_network'. The worksheet contains pairs of nodes, given by their coordinates, that can be directly connected (see Figure 3). The routing network should include, in at least one of the coordinate pairs, the coordinates of the terminals and if applicable also the coordinates of storage nodes. The

routing network can also include other nodes. These extra nodes will be assumed to have no demand or supply and are automatically added to the routing network based on the given coordinates. These nodes might be used in the final network, but they don't have to be included.

For all connections in the routing network the assumption is made that they can be used in both directions.

	A	B	C	D
1	node 1	node 2		
2	(35, 43)	(25, 36)		
3	(35, 43)	(25, 51)		
4	(35, 43)	(34, 31)		
5	(25, 36)	(10, 41)		
6	(25, 36)	(25, 51)		
7	(25, 36)	(34, 31)		
8	(61, 37)	(67, 50)		
9	(61, 37)	(79, 38)		
10	(25, 51)	(10, 41)		
11	(25, 51)	(3, 58)		
12	(52, 53)	(46, 66)		
13	(52, 53)	(61, 59)		
	terminals	storage	routing_network	

Figure 3: Worksheet 'storage' from 'input_data.xlsx'

Worksheet 'obstacles' (optional)

Another possibility is that the network needs to avoid obstacles. If that is the case, you need to give information on these obstacles in the worksheet 'obstacles'. Define each obstacle as a polygon by its corner points. Each column defines a new obstacle (see Figure 4). You can add or delete obstacles.

Obstacles should not overlap. If obstacles overlap or have an edge in common, you should combine the different obstacles into one. The polygons do not have to be convex but should not be self-intersecting.

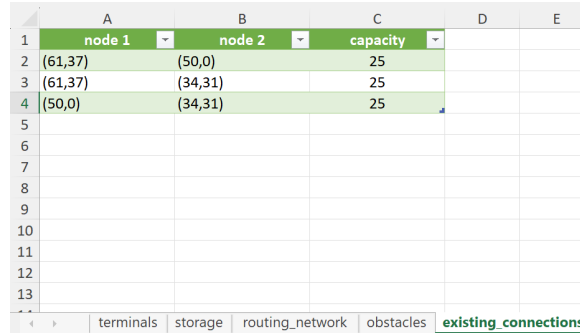
	A	B	C
1	obstacle 0	obstacle 1	obstacle 2
2	(0,10)	(40,5)	(10,60)
3	(0,30)	(40,50)	(10,80)
4	(30,30)	(60,50)	(40,80)
5	(30,10)	(60,30)	(40,60)
6		(100,30)	
7		(100,20)	
8		(60,20)	
9		(60,5)	
10			
11			
12			
13			
	terminals	storage	routing_network
			obstacles

Figure 4: Worksheet 'obstacles' from 'input_data.xlsx'

If routing is restricted to specific network connections (see worksheet 'routing_network'), you should incorporate the possible obstacles in the routing network by leaving out the connections that are not possible. You cannot ask the ONLT to take into account both obstacles and the routing network at the same time.

Worksheet 'existing_connections' (optional)

There may be existing connections or even an entire network to connect new lines to. You define these existing connections in the worksheet 'existing_connections'. Define each connection by the coordinates of its endpoints and the capacity that can be reused. The capacity needs to be given in the same volume-units per time-unit as used for the other input. Again, the assumption is made that the existing connections can be used in both directions.



	A	B	C	D	E
1	node 1	node 2	capacity		
2	(61,37)	(50,0)	25		
3	(61,37)	(34,31)	25		
4	(50,0)	(34,31)	25		
5					
6					
7					
8					
9					
10					
11					
12					
13					

Figure 5: Worksheet 'existing_connections' from 'input_data.xlsx'

Nodes in the existing connections might be terminals or storage nodes, but this is not required.

Existing connections might go through obstacles, but their end points should lie outside obstacles.

2.4. Input requirements

To summarize, you should keep in mind the following model requirements.



- Worksheet names should not be changed.
- All terminals in the worksheet 'terminals' should be numbered 0, 1, 2, 3, etc.
- Supply is given as a negative number, demand as a positive number in a volume-unit per time-unit.
- The terminals and storage nodes should be part of the routing network, otherwise they cannot be connected to the network.
- Obstacle restrictions should be incorporated in the routing network if both apply.
- The routing network needs to be connected, otherwise part of the nodes cannot be reached.
- Existing connections can pass obstacles, but their end nodes cannot lie within obstacles.
- If there is a routing network and there are existing connections, the end nodes of the existing connections should also be part of the routing network.
- End nodes of existing connections can be terminals or storage nodes, but they don't have to.

3. Model execution

All the options and model steps will be discussed in detail in this chapter.

3.1. Run the model – no storage, routing restrictions, obstacles, or existing connections

Open the file *user_interface.py* in Spyder. If the input-file *input_data_simple.txt* is well defined, and in the given folder, default *case_study*, you can run all code in the file *user_interface.py* by clicking the

green triangle () in the upper menu bar. The model runs in different steps. Each next step will save its output to the *output_data.xlsx*. This means that you can exclude previous steps in new model runs to save time, since the output of these previous steps can be found in the *output_file*. You can exclude a step by adding # before the command line or by only executing the cell you need ()

We will discuss each step now:

Set input parameters

Before running the model, you need to give some extra information in Step 0.

- *folder* gives the folder's name in which you have saved the input file. The default is 'case_study'.
- *inputfile* defines the input file in which your input data is given. The default is 'input_data.xlsx'.
- *outputfile* defines the output file to which all output data is written. The default is 'output_data.xlsx'.
- *beta* (β) is the capacity-cost exponent and has a value between 0 and 1. The default value is 0.6.
- *routing* is set to True if routing restrictions by a given routing network need to be applied. The default is False.
- *obstacles* is set to True if routing restrictions by given obstacles need to be applied. The default is False.
- *existing* is set to True if existing connections can be used in the new network. The default is False.
- *spc* defines the absolute cost for adding a new splitting point. The default value is 0, which means that splitting points can be added for free.
- *upc* defines the relative cost for using capacity of existing connections compared to building a new connection of the same capacity. The default value is 0, which means that there are no cost for using existing connections.
- *cpc* defines the relative cost for extension of the current available capacity of existing connections compared to building a new connection of the same extension. The default value is 1, which means that extending current available capacity is just as expensive as building a new connection.
- *node_size* sets the size of the nodes in every network plot. If you have a large network with a large number of nodes, you can set the *node_size* to a smaller value. The default value is 100.
- *show_output* determines whether intermediate results, like plots, are shown or not. Showing all intermediate results can significantly increase the runtime but gives you more information on what happens. The default value is False.
- *extreme_length* defines which connections are considered when searching for new connections. Connections that are longer than the set value are excluded from the search. The default value is infinite. Setting this parameter to a smaller value can reduce the runtime.
- *cost_deviation* defines a stopping criterion. If the cost improvement between the previous and the new network is smaller than the set value, the search will stop looking for improvements. The default value equals 0. Setting this parameter to a higher value can reduce the runtime.

The cost-function on which the model assesses the quality of the different network topologies is defined by

$$C(G) = \sum_{e \in E_n(G)} l_e q_e^\beta + spc \cdot s(G) + \sum_{e \in E_o(G)} l_e (upc \cdot \min(q_e, r q_e)^\beta + cpc \cdot \max(0, q_e - r q_e)^\beta),$$

$E_n(G)$ is the set of all new edges (connections) in the network G , l_e is the total length of the edge e (e.g. the Euclidean distance based on the coordinates of the edge end points), q_e is the edge's capacity. For every new to build connection the cost equals $l_e q_e^\beta$.

β is the capacity-cost exponent, indicating how profitable it is to join edges. If capacity does not count in the network cost, then $\beta = 0$, if connections of double capacity are twice as expensive, then $\beta = 1$. The value of β is typically somewhere around 0.6.

You can add extra cost for the splitting points $s(G)$ in the network. If connections can only be split at existing nodes, you should set spc to a high value to make extra splitting points very unattractive. The cost for all splitting points together equals $spc \cdot s(G)$.

Let $E_o(G)$ be the set of all existing connections. For all edges $e \in E_o(G)$ the current capacity is denoted by $r q_e$. Reusing part of this capacity can add extra cost equal to $upc \cdot q_e$ for $q_e \leq r q_e, e \in E_o(G)$. If $upc = 0$ then reusing existing capacity brings no extra cost. If $upc = 1$, reusing existing capacity is just as expensive as building a new connection.

Also, an extension of the current (remaining) capacity $r q_e$ of existing connections can add extra cost. If $cpc = 1$, then extending the current capacity of an existing connection is just as expensive as building a new connection. If $cpc < 1$, an extension of the existing connection is cheaper. If an extension is impossible, you should set cpc to a high value to make this option very unattractive. The extra cost for extending existing capacity equals $cpc \cdot (q_e - r q_e)^\beta$ for $q_e > r q_e, e \in E_o(G)$.

The cost function in the model is only used to compare different solutions and is by no means realistic. To make the total cost more realistic, you need to multiply it with a reality factor.

Be careful to take into account the units as used in the input data. So, if coordinates are given on a meter-grid and the flow through the network in litre per second, then the cost reality factor should be given in $(\text{€} \cdot s)/(m \cdot l)$

Step 0: Preparation

Step 0 is a preparation step. All data is read from the input-file and stored in a dictionary. Besides the ONLT tests whether the demand and supply sum up to zero in each time step. If so, this will speed up the search for minimal cost network.

The procedure *demand_over_nodes()* determines the total number of time steps T given in the input file and shows a bar chart (Figure 6) with the total demand (in blue) and total supply (in red) in VU (= Volume Unity) for all demand and supply nodes in the network summed over all T time steps. In the example '*input_file.xlsx*', there are 9 terminals (labelled from 0 to 8). Each terminal is a producer in some time steps and a consumer in other time steps.

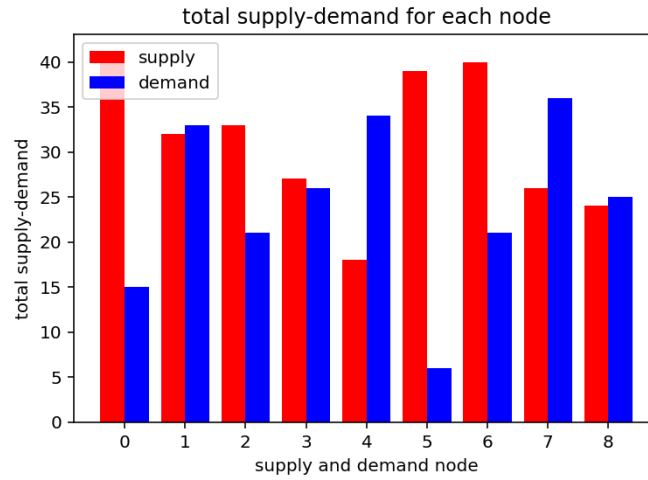


Figure 6: Bar chart with total supply and total demand of each terminal over all time steps

The procedure *demand_over_time()* shows a bar chart (Figure 7) with the total demand (in blue) and total supply (in red) in *VU* (= Volume Unity) for all time steps summed over all terminals.

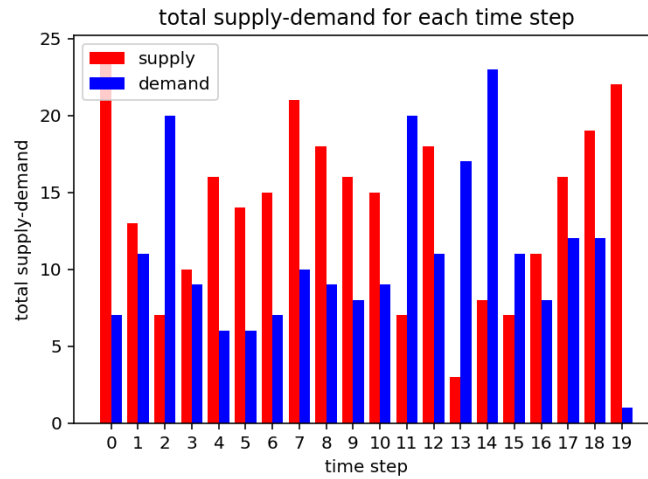


Figure 7: Bar chart with total demand and total supply in each time step from all terminals

In this example, there are supply and demand values for 20 different time steps. In some steps the supply is much higher than the demand and vice versa, so storage need to be used in order to supply as much as possible.

The procedure *plot_demand()* displays the supply and demand over all timesteps for every individual terminal (Figure 8).

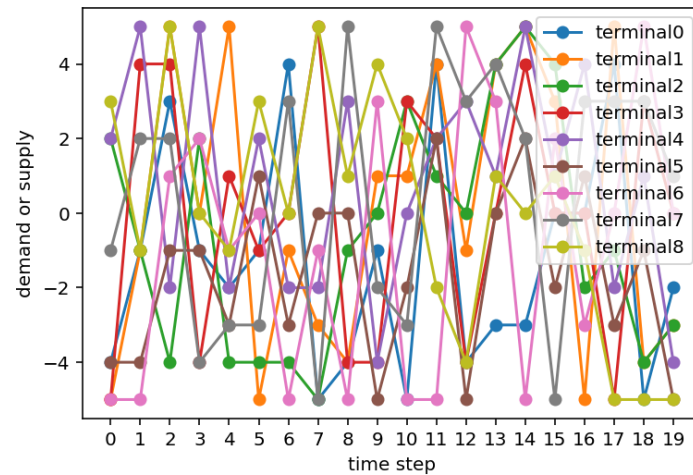


Figure 8: Plot for demand and supply over all time steps for each individual terminal

Step 1: Determine minimum spanning tree

The starting network will be the minimum spanning tree. It is the minimum-length network that connects all nodes. Just enough capacity is assigned to the connections in the network to satisfy the most demand by supply or storage at every time step and to put the remaining supply in the nearest possible storage. So the capacity (in VU) to a connection is the maximum over all time steps of the sum of all flows (in VU) over that connection.

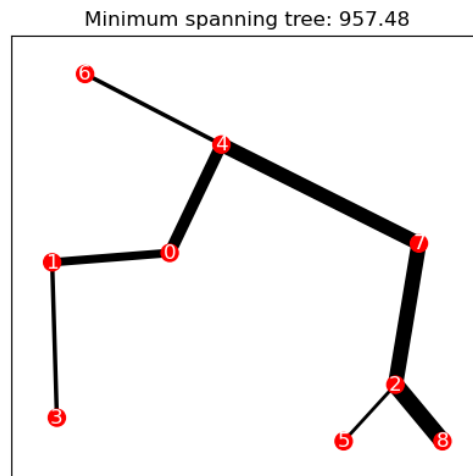


Figure 9: Minimum spanning tree with sufficient capacity assigned for 'input_data_simple.xlsx'

The procedure shows a graph (Figure 9) of the determined network, where the thickness of the connections is relative to the assigned capacity. The network shows the terminals in red.

The total costs for this network can then be calculated by the formula given before. These costs are also given as output and equal 957 MU (monetary unit).

The network that results from this step is called MST. Information about this network is saved in the original folder *case-study* in the file *output_data.xlsx* but can also directly be called for in the Spyder console. The network contains different types of output:

End nodes of the connections

Each connection is defined by its two end nodes. Information about the edges is stored in the output file *output_data.xlsx* in the worksheet *MST_edges* where the edges are uniquely defined by their end nodes in the first two columns labelled 'node1' and 'node2'. All edges in the network can also be found by typing in *MST.edges()* in the Spyder console.

Length of the connections

The length of each connection can be found by *nx.get_edge_attributes(MST,'weight')* or *MST.edges(data='weight')*. It is the Euclidean distance between the coordinates of the connected nodes. Both capacity and length of the connections are used to calculate the total cost. This same information is stored in the output file *output_data.xlsx* in the worksheet *MST_edges* in the column 'length'.

Capacity of the connections

You can ask for the capacity of each connection by typing *nx.get_edge_attributes(MST,'capacity')* in the Spyder console, which in this case gives the following output:

```
{(0, 1): 10, (0, 4): 14, (1, 3): 5, (2, 8): 20, (2, 5): 4, (2, 7): 18, (4, 6): 5, (4, 7): 17}
```

This output is a dictionary. For each connection (edge) in the network it shows the assigned capacity.

You can also use *MST.edges(data='capacity')*

Also this information is stored in the output file *output_data.xlsx* in the worksheet *MST_edges* in the column 'capacity'.

Current capacity of the connections

Existing connections can be reused in the network. The capacity of these existing connections is also stored in the output file in the worksheet *MST_edges* in the column 'current'. For new connections this value equals 0.

Cost of the connections

The cost of each connection is calculated by $l_e q_e^\beta$. For existing connections there might be extra costs. This cost can be found by *nx.get_edge_attributes(MST,'cost')* or *MST.edges(data='cost')*. Also this information is stored in the output file *output_data.xlsx* in the worksheet *MST_edges* in the column 'cost'.

Coordinates of the nodes

The coordinates of the nodes can be found by *nx.get_node_attributes(MST,'coord')* or *MST.nodes(data='coord')*. In this case, the coordinates will be equal to those given in the input file. However, when extra splitting nodes are included, their coordinates can also be found in this way. Information about the nodes is stored in the output file *output_data.xlsx* in the worksheet *MST_nodes*.

Type of nodes

Each node in the network will belong to one of the following types:

- *terminal*: the supply and/or demand nodes that need to be connected.
- *storage*: the storage nodes

- *steiner*: the extra splitting nodes in the network, also the extra nodes in the network coming from the routing network.
- *split*: the additional splitting nodes on existing connections.
- *corner*: the nodes in the network that are needed to turn around obstacles.
- *existing*: the nodes in the network that are endpoints of existing connections but are no terminals.

The type of each node in the network can be found by `nx.get_node_attributes(MST, 'stamp')` or `MST.nodes(data='stamp')`. In the initial minimum spanning tree in Figure 9, all nodes are terminals. Also this information is stored in the output file in the worksheet *MST_nodes*.

Cost of nodes

As said before, splitting nodes might bring extra investment costs. Information about the cost of the nodes can be found by `nx.get_node_attributes(MST, 'cost')` or `MST.nodes(data='cost')`. In the minimum spanning tree in Figure 9 all nodes are terminals, so all cost equal 0. This information is also stored in the output file in the worksheet *MST_nodes*.

Cost of network

You can find the network cost by typing `MST.graph['cost']` in the console. This cost is the sum of all the edge and node cost. This information is not separately stored in the output file.

Each network found in the following procedures has these edge, node or graph attributes. And the same information about these networks is stored in the output file.

Be aware that every time you run the model the output file is overwritten. So, if you want to keep the output of different runs, you should define different names for the output files at the start.

You can now use the following steps to improve the results of the minimum spanning tree.

Step 2: Determine minimum-cost-spanning tree

The minimum-spanning-tree solution did not consider the capacity cost for building a specific connection. By rewiring the connections in the minimum-spanning-tree, a better solution might be found. The rewiring process is a heuristic process and does not guarantee the optimal solution to be found since earlier decisions may restrict later choices.

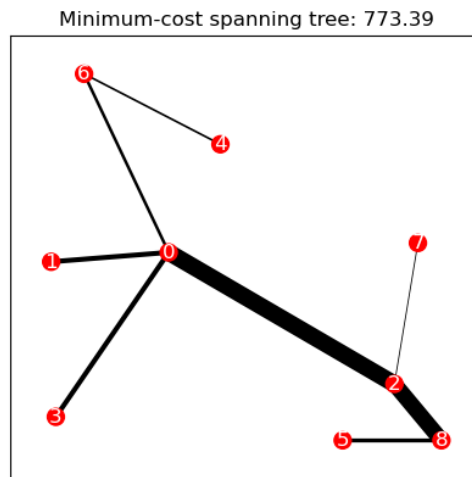


Figure 10: Minimum-cost spanning tree for 'input_data_simple.xlsx'

Figure 10 shows the minimum cost spanning tree that is found with a total cost of 773 MU.

In every step, you can choose to show the intermediate output (*output = True*) or not (*output = False*). The default is *False*.

The intermediate steps show one profitable edge replacement at a time (see Figure 11).

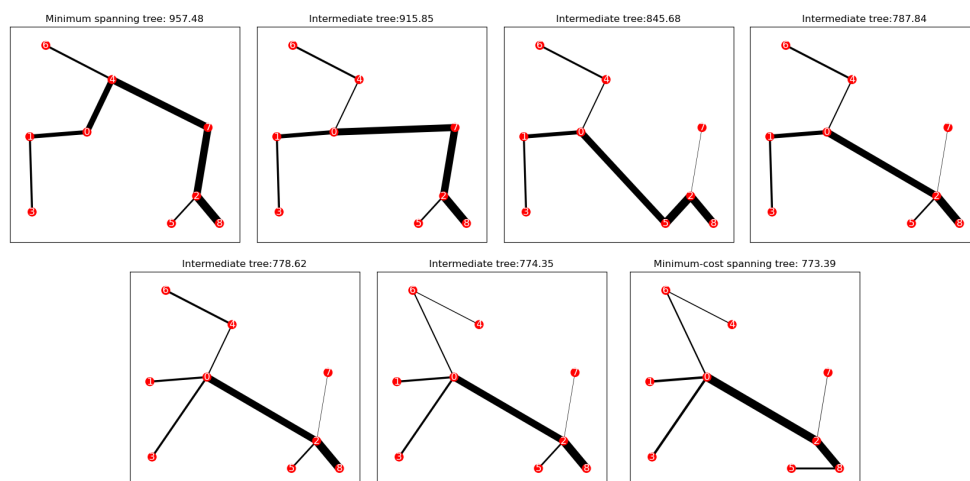


Figure 11: Intermediate steps for finding the minimum-cost spanning tree for 'input_data_simple.xlsx'

The final minimum-cost spanning tree is called FT. You can obtain the same information from this network as earlier for the minimum-spanning-tree MST. This information is also stored in the output file in the worksheets *FT_nodes* and *FT_edges*.

Finding the minimum-cost spanning tree can take time if many nodes need to be connected or many time steps are given. In that case, it could be useful to set different values to the parameters *k*, *extreme_length* and *cost_deviation* as explained under **Set input parameters**. However, changing these parameter values, it is not guaranteed that the same network is found.

Step 3: Determine minimum-cost-Steiner tree

In some cases, shorter networks exist when allowing extra splitting points in the connections. Since length generally contributes significantly to the building costs, this might reduce the network costs. Adding extra splitting points in the network might also introduce new costs for building these splitting points. So a good balance has to be found.

This model step adds splitting points to the network if they give profitable improvements. In Graph Theory, these splitting points are called *Steiner nodes*, and the shortest length tree, including these Steiner nodes, is called a *minimum (length) Steiner tree*. Since length and capacity influence the cost, our method searches for the minimum-cost-Steiner tree.

Steiner nodes are added to the smallest angles in the minimum-cost spanning tree one by one as long as cost improvements can be found.

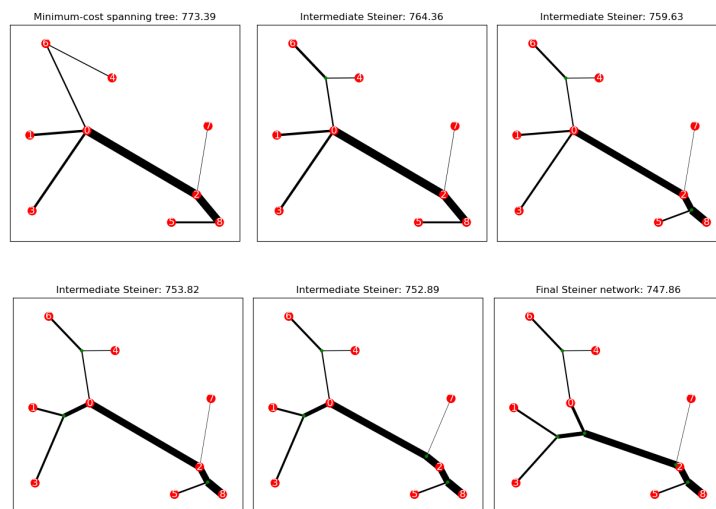


Figure 12: Steiner nodes added to the minimum-cost-spanning-tree found for 'input_data_simple.xlsx'

Figure 12 shows the intermediate networks in which Steiner nodes (in green) are added to small angles as long as they are reducing the cost.

When we start with the minimum-cost spanning tree, Steiner nodes are added to the final Steiner network one by one. The total cost of the final network is equal to 748 MU. An improvement of 25 MU compared to the minimum-cost spanning tree. However, in this example we assumed that there were no extra costs for splitting nodes by setting *spc* equal to 0.

If splitting cost is higher than zero, then fewer splitting nodes might be added, and a different Steiner tree might be found as the best result. Figure 13 shows the best Steiner tree found when splitting cost *spc* are set to 5.

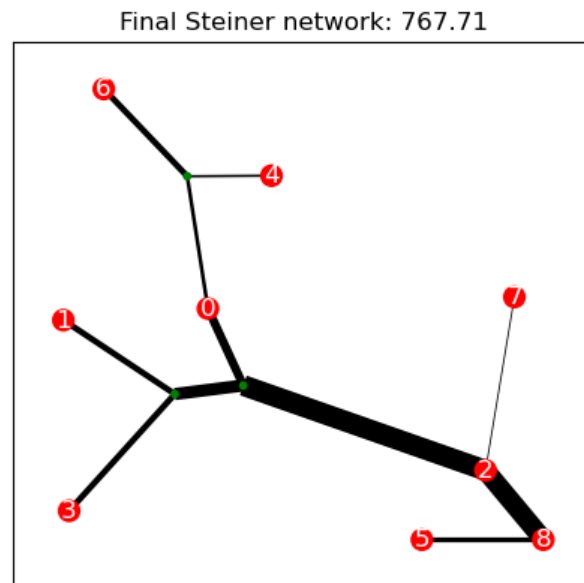


Figure 13: Best Steiner tree with $\text{spc} = 5$ for 'input_data_simple.xlsx'

Step 4: Last improvement round

In the last round, step 2 and 3 are repeated as long as better results are found. In many cases, the minimum-cost Steiner tree will not be improved in this last step. However, for the current example the best network is found as in Figure 14 at a cost of 760 MU.

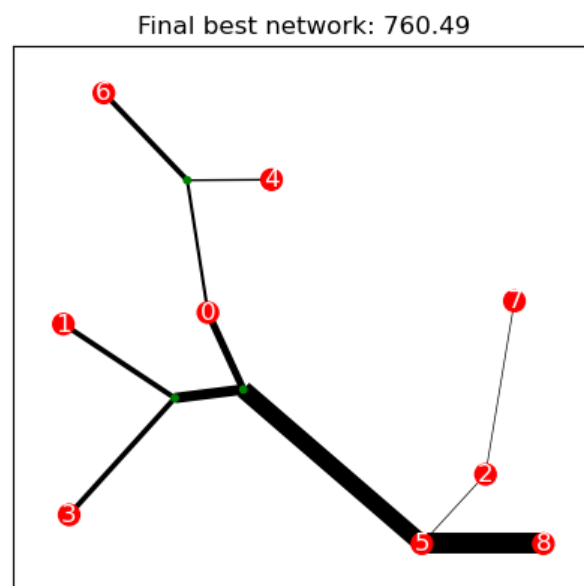


Figure 14: Best network for 'input_data_simple.xlsx' ($\text{spc} = 5$)

3.2. Run the full model – without any restrictions

The example in 'input_data.xlsx' is a full example with storage nodes, a routing network, obstacles and existing connections. Every time you run the model, you can choose which restrictions or extensions you would like to incorporate. When you run the model for the full example but without any

restrictions you will find the networks as in Figure 15. Unless mentioned explicitly, the models are run with the default parameter settings.

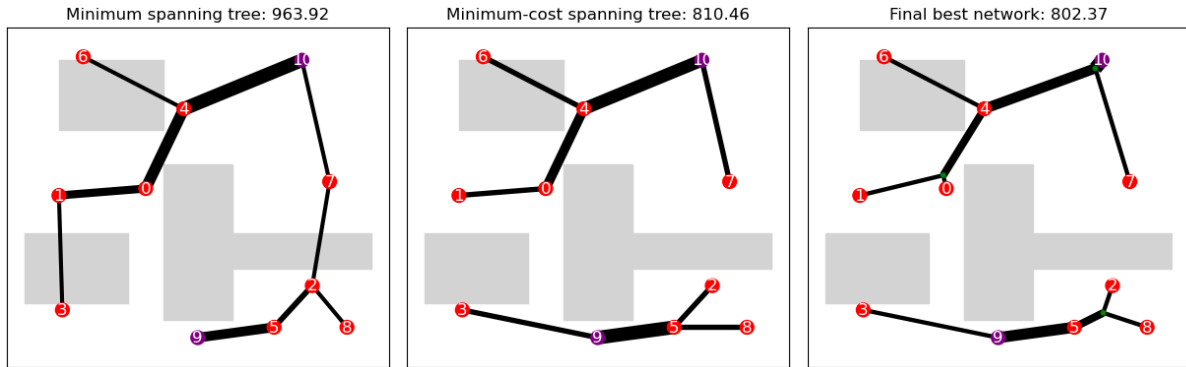


Figure 15: Minimum spanning tree, minimum cost spanning tree and best network for 'input_data.xlsx'

Although the obstacles (grey polygons) are not used to restrict the network, they are still shown in the plots to inform the user on how the networks will cross these obstacles.

The purple nodes in the plots are the storage nodes. The green node is an extra splitting node (Steiner node).

The results in this example show that the final network does not have to be one connected network. If supply and storage in part of the network is sufficient to satisfy demand in that part of the network, the final network might consist of some loose sub networks.

The storage nodes are used in this example. You can see the amount of storage in the different time steps by calling `gp.plot_storage(G,input_dict,title)` from the Spyder console. This plot is not given automatically. You can choose for which network you want to show the storage levels. E.g. `gp.plot_storage(NT,input_dict,"storage in final network")` gives the plot in Figure 16.

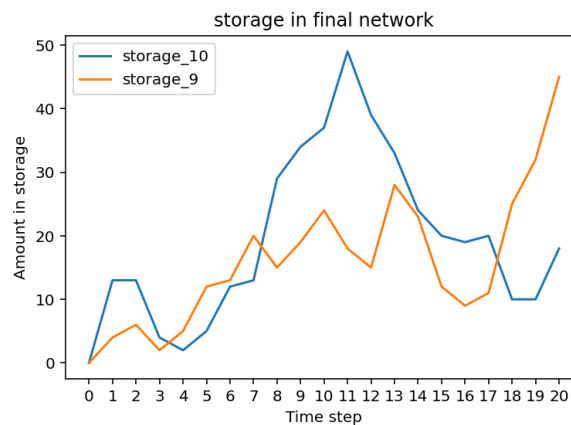


Figure 16: Storage levels over time for 'input_data.xlsx'

3.3. Run the full model – with routing restrictions

When routing restrictions apply, step 1, the initial topology will be restricted to only allowed connections.

Step 1 will result in the minimum-length-spanning tree that connects all demand and supply nodes and part of the Steiner nodes. In Graph Theory, the problem of finding this tree is called the *minimum-Steiner-tree-in-a-graph problem*. It differs from the Steiner tree we have seen before because now the Steiner nodes have fixed positions. Moreover, it differs from the minimum-length-spanning tree because more nodes than only the supply, demand and storage nodes are connected.

Finding the minimum-Steiner-tree-in-a-graph is again a hard problem for which several heuristics are developed. In the ONLT, we use one heuristic to find this Steiner tree. The outcome of this heuristic will be given as the result of Step 1.

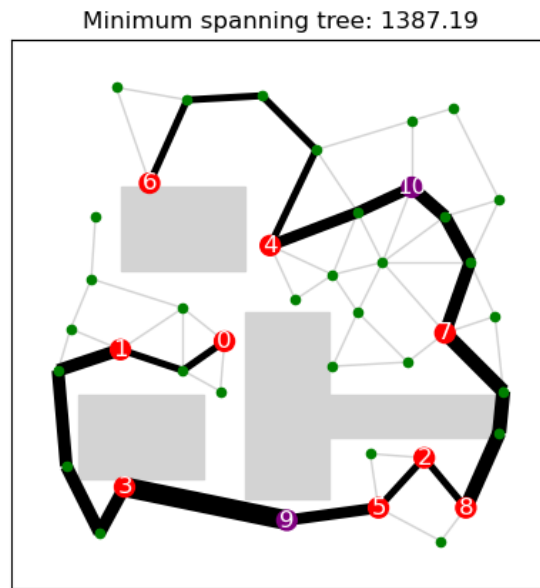


Figure 17: Minimum length Steiner tree in the restricted routing network for 'input_data.xlsx'

Figure 17 shows the minimum-length Steiner tree (for simplicity reasons here again called minimum spanning tree). The green nodes are also called Steiner nodes. These nodes can (but don't have to) be used in the network to connect the supply-demand nodes. The total cost for this network is 1387 MU. When routing is restricted, networks will, naturally, be longer and, by that, be more expensive. Although, these routing restrictions might reduce other unwanted costs, e.g. cost to acquire land.

In **Step 2**, the same procedure is used to find profitable edge changes. However, the potential new edges can again only follow the connections in the routing network.

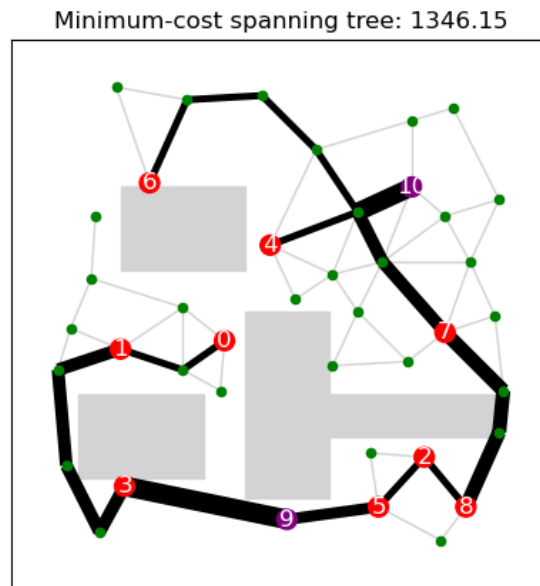


Figure 18: Minimum-cost Steiner tree in the routing network for 'input_data.xlsx'

Figure 18 shows the final minimum-cost Steiner tree. The total cost for this minimum-cost Steiner tree equals 1346 MU.

It has no use to apply step 3 and step 4 in the case of a routing network since splitting nodes cannot be freely positioned now, since they need to be cross points in the routing network and are already considered.

The cost for taking Steiner nodes from the routing network into the spanning trees are also based on the parameter *spc*. So if you want a network with a low number of extra points that is still restricted to the routing network you can set *spc* to a high value.

3.4. Run the model – with obstacles

When several regions cannot be crossed, the regions need to be defined as obstacles in the model. Connections cannot cross these obstacles but need to be redirected around them.

The same steps are performed as before, but, as can be expected, with different resulting networks and corresponding cost (see Figure).

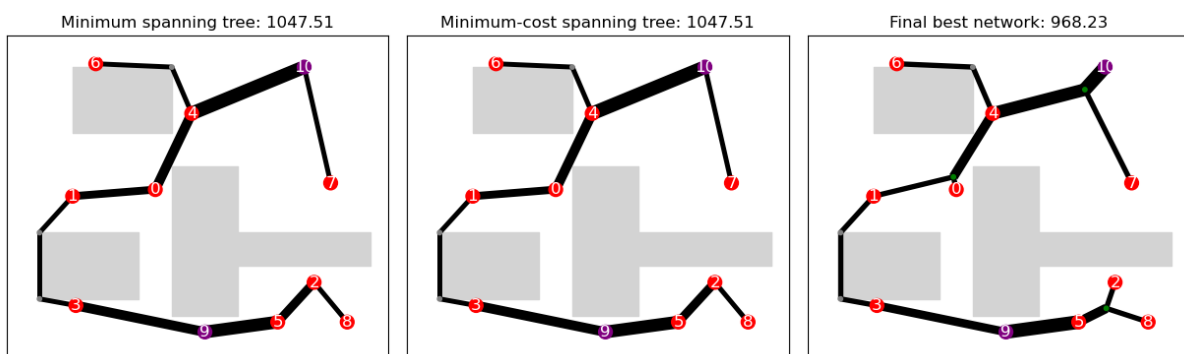


Figure 13: Minimum-spanning-tree, minimum-cost-spanning-tree and final Steiner network around obstacles for 'input_data.xlsx'

3.5. Run the model – with existing connections

The last option that can be chosen is the case where already existing connections with spare capacity are present in the network region. Although not required, the nodes that need to be connected can already be part of the existing connections, but that is not needed and is not the case in this example.

In this example, the existing connections have a spare capacity of 25 VU. Existing connections can be split if that is a profitable possibility. Splitting cost of an existing connection is assumed to be equal to the cost of splitting new connections and in the example is set to $spc = 0$.

Again, the same steps are executed to find the best network topology. As can be seen in Figure 19, re-using existing connections can significantly lower the investment cost. In the minimum spanning tree all existing connections will be connected. However in the minimum-cost spanning tree and the final network only existing connections that are profitable are present in the network. In this example, all three existing connections are reused. The total capacity of the existing connections is indicated by the light blue color. The dark blue line shows the amount that is used.

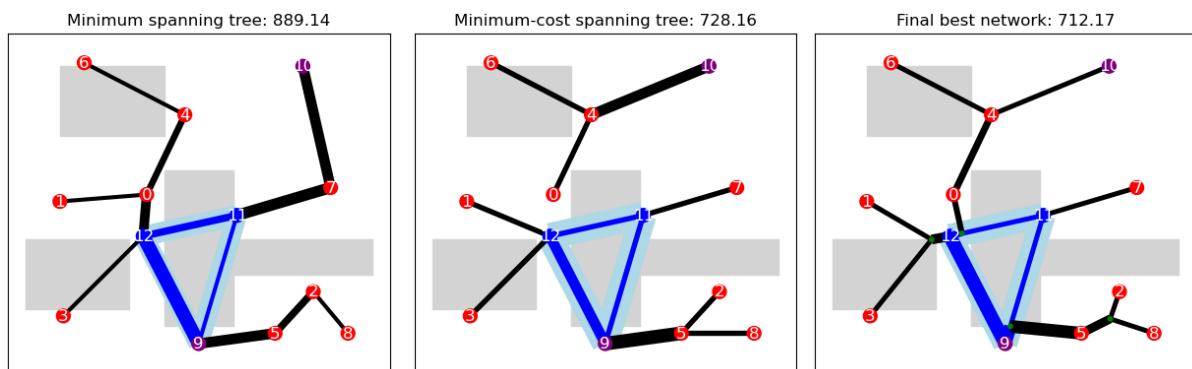


Figure 19: Resulting network when existing connections are present for 'input_data.xlsx'

In the previous networks the existing capacity could be used for free. However, when reusing existing capacity brings extra cost (e.g. $upc = 0.2$) the reuse might be limited (see Figure 20).

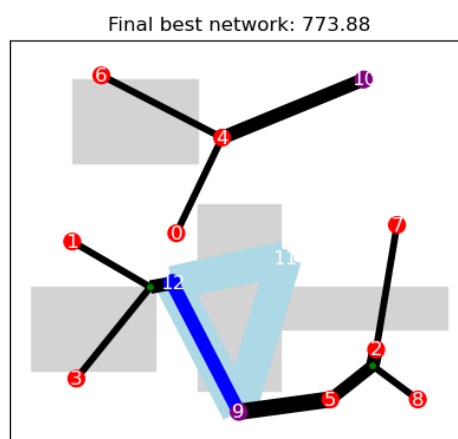


Figure 20: Network when reusing existing capacity costs 20% of building new capacity for 'input_data.xlsx'

When the existing connections do not have sufficient spare capacity, their capacity is expanded if needed, and the extra cost for this capacity extension is given in the output.

For example, if the spare capacity of the existing connections is only 2 VU, a more expensive network is found (Figure 21) including extra cost for extension, under the assumption that extension is just as expensive as building new connections ($cpc = 1$). The use of the existing connection is limited in this case. Existing connections that are extended are shown in purple.

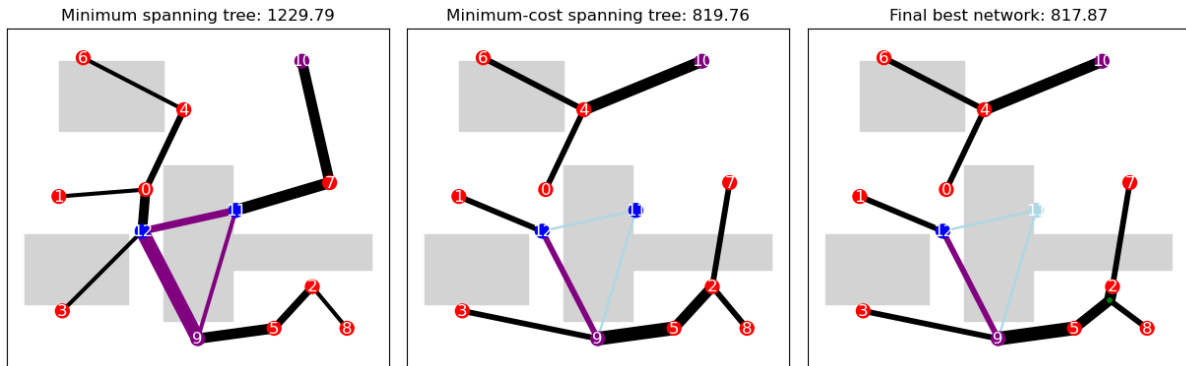


Figure 21: Optimal network when existing connections have little spare capacity for 'input_data.xlsx'

If an extension of the existing connections is far more expensive ($cpc = 2$), or perhaps even impossible, than building new connections, the existing connections are just ignored, as shown in Figure 22.

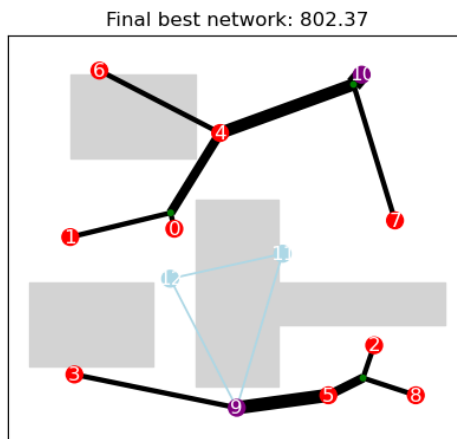


Figure 22: Optimal network when existing connections have little spare capacity and extension is expensive for 'input_data.xlsx'

3.6. Combination of options

The options discussed above can be combined. However, as said before, you cannot combine *obstacles* and *routing*. Instead, obstacle restrictions should be incorporated into the routing network. The other combinations are possible, and for the example case, the results of these combinations are shown below. E.g. a routing network can be combined with existing connections (see Figure 23).

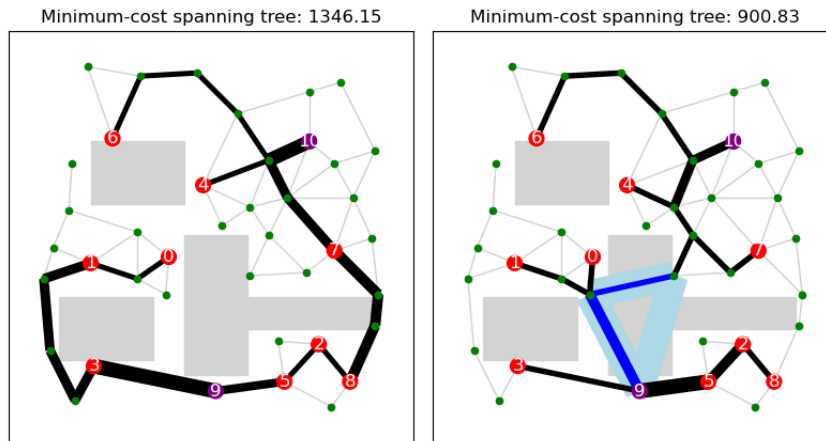


Figure 23: Best network in a routing restriction network with and without existing connections for 'input_data.xlsx'

The restrictions given by obstacles can also be combined with existing connections (Figure 24). These existing connections can even cross obstacles, since they are a given fact. However, their endpoints are assumed to lie outside the obstacles to make new connections to these endpoints possible.

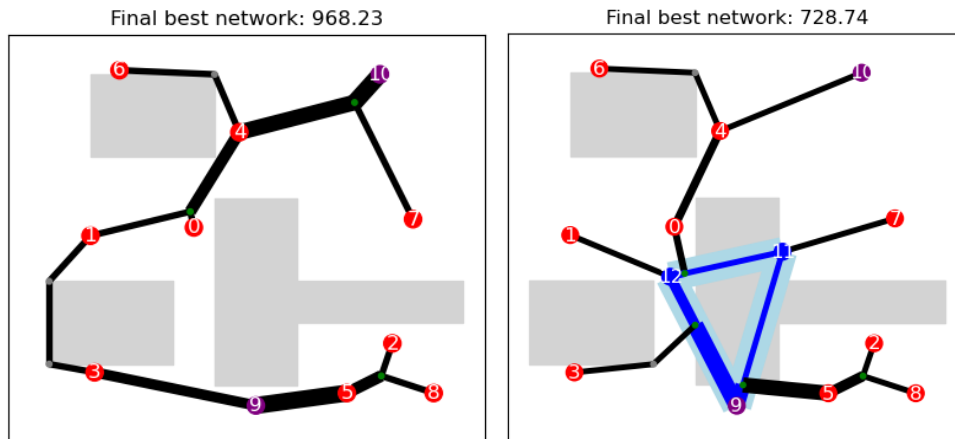


Figure 24: Best network with routing restrictions given by obstacles, with and without existing connections for 'input_data.xlsx'

3.7. Some final remark

For the simple example in 'input_data_simple.xlsx' the demand and supply was given only for one timestep (see Figure 25). In this example there is one big consumer (node 8) and the other nodes do have a moderate demand or supply.

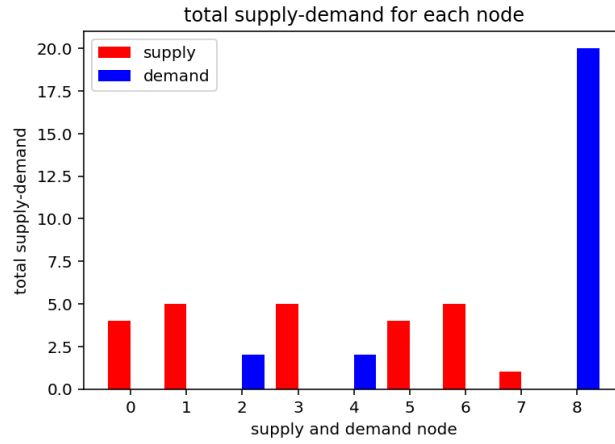


Figure 25: Demand and supply per node for 'input_data_simple.xlsx'

Choosing another value for the capacity-cost exponent will generally lead to totally different network topologies which can clearly be shown for this example. Figure shows the result when $\beta = 0, 0.5$ and 1.

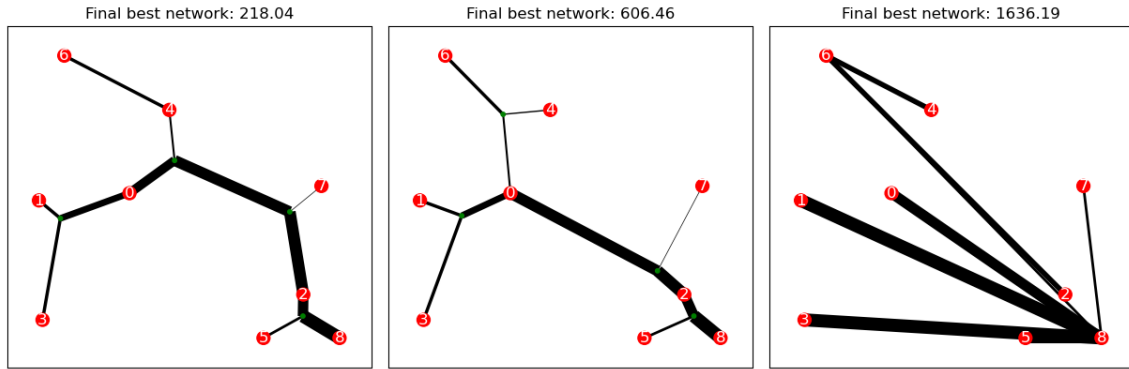


Figure 20: Final minimum-cost network with $\beta = 0, 0.5$ and 1, respectively for 'input_data_simple.xlsx'

When β is small, the capacity has only a minor influence on the cost and the optimal network is as short as possible to reduce the influence of the length on the cost. When β is large, building two parallel pipes is almost just as expensive as building one bigger pipe. In that case direct connections between nodes that exchange a large part of the flow make the network cheaper. This explains the direct connections to node 8 when $\beta = 1$. Since node 8 is the largest consumer, node 0, 1, 3, 5, 6 and 7 are almost entirely supplying node 8. Node 2 is on the path from 6 to 8, so can easily be supplied by 6 without much extra cost. And node 4, another consumer, is in the shortest way connected to one of the suppliers, here again node 6.

Remember that normally β is determined by the market cost for different pipeline diameters or capacities and **cannot be chosen**. Moreover, only network costs determined with the same value of β are comparable. It is meaningless to compare different network topologies on their investment costs for different values of β . Last but not least, the width of the connections in the network plots are relative to the connections capacities, but this width is set for each plot separately. So that the plot for $\beta = 1$ has more thick lines, does not tell that in that network capacities are larger. Exact capacity values can be found in the output file.

3.8. Speeding up the search

In case of many nodes and many time steps the time complexity of the tool might be high. To speed up the search (sometimes at the expense of optimality), one can set some parameters, as discussed before.

- The maximum length for new edges to enter the network (parameter: *extreme_length*) can be set to a smaller value.
- The required cost improvement (parameter: *cost_deviation*) to continue the search can be set to a higher ratio (between 0 and 1).

Another time-saving option is to skip the last step, the extra improvement round. In many cases this last round will not lead to any improvements, but it will take some time to come to that conclusion.

The inclusion of storage in the model might significantly increase the time complexity because of the higher degrees of freedom.

3.9. Manipulating the output

The output is mostly given as plots when executing the model, as shown in the previous examples. Moreover, the final network's cost and intermediate trees' cost are printed.

When the file *user_interface.py* is still open, the resulting graphs are also available as MST (minimal-length spanning tree), FT (minimal-cost spanning tree), ST (minimal-cost Steiner tree) and NT (final best network). As mentioned before, the node coordinates, type and cost and the edge length, capacity and cost of these 4 graphs are stored in the *outputfile* defined at the start.

The following procedures from the file *general_procedures.py* can also directly be called for in the Spyder console. Note that you should first open and run this file to make the procedures callable from the console.

coordinates_from_file(folder+inputfile)

Procedure to obtain the coordinates of the terminals

coordinates_storage_from_file(folder+inputfile)

Procedure to obtain the coordinates of the storage nodes

amount_storage_from_file(folder+inputfile)

Procedure to obtain the amount of storage at the start

capacity_storage_from_file(folder+inputfile)

Procedure to obtain the storage capacity

routing_network_from_file(folder+inputfile)

Procedure to obtain only the routing network as a graph. The coordinates of the nodes are available as node attribute 'coord'.

demand_from_file(folder+inputfile)

Procedure to obtain the demand at every time step as a nested dictionary

network_from_excel(output_path,title)

Procedure to obtain the minimum spanning tree, minimum cost spanning tree, minimum Steiner tree or the final best graph as graph from the output file given title = 'MST', 'FT', 'ST' or 'bestG' respectively.

draw_obstacles(input_dict)

Procedure to draw only the obstacles (no-go areas). The results is shown after plt.show()

draw_routing(input_dict,node_size=20)

Procedure to only draw the routing network. The node size can be adapted. The result is shown after plt.show()

draw_network(G,title,input_dict,routing=False,node_size=100,show_capacity=True)

Procedure to show one of the resulting graphs with or without routing network.

total_cost(G,beta,spc=0,upc=0,cpc=1)

Procedure to calculate the total cost for a network G with edge attributes (*capacity*, *weight* and, in case of existing pipelines, *current*) and node attributes (*stamp* for the different node types). beta is the capacity-cost exponent with a value between 0 and 1.

euclidean_weighted_graph(G)

Procedure to add length of the edges as edge attribute 'weight' to a graph G, based on the Euclidean distance between the end nodes of the edges. Coordinates of the nodes should be present as node attribute 'coord'.

plot_storage(G,input_dict,title)

Procedure the amount in storage over all time steps for a given graph.

4. Further reading

The model is described in more detail in the following papers:

A method for designing minimum-cost multi-source multi-sink network layouts

PW Heijnen, E.J. Chappin, P.M. Herder - Systems Engineering, 2019

<https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21492>

Infrastructure network design with a multi-model approach: Comparing geometric graph theory with an agent-based implementation of an ant colony optimisation

P Heijnen, E Chappin, I Nikolic - Journal of Artificial Societies and Social Simulation 17 (4), 1, 2014

<http://jasss.soc.surrey.ac.uk/17/4/1.html>

Maximising the Worth of Nascent Networks

PW Heijnen, A Ligtoet, R.M. Stikkelman, P.M. Herder - Networks and Spatial Economics, 2014

<https://link.springer.com/article/10.1007/s11067-013-9199-1>

The model is used in the following papers and theses:

Hydrogen infrastructure planning under uncertainty in an industrial port cluster

Lafeber, Meike (TU Delft Technology, Policy and Management, 2024)

<https://resolver.tudelft.nl/uuid:426038aa-bd50-4f20-8463-8149c8d6367e>

Cost optimal hydrogen transportation

Waal, Boris (TU Delft Technology, Policy and Management, 2024)

<https://resolver.tudelft.nl/uuid:d9f0aeb7-61a4-4666-b75c-81fd0f3d2848>

Optimizing district heating networks: Exploring the solution space

Piket, Martijn (TU Delft Technology, Policy and Management, 2023)

<http://resolver.tudelft.nl/uuid:89231e65-19f8-453b-bbdc-e8bed5d5952e>

Creating Clusters in a 5th Generation District Heating and Cooling Network

Burk, Sarah van (TU Delft Technology, Policy and Management, 2023)

<http://resolver.tudelft.nl/uuid:059d58aa-9163-444a-94e2-a76e48626683>

Improving the realisation of the Dutch hydrogen backbone

Nijmeijer, Wesley (TU Delft Technology, Policy and Management, 2023)

<http://resolver.tudelft.nl/uuid:28390cbf-9ad6-4565-ad80-2859edc9ae3b>

Constructing and analysing off-shore hydrogen system designs powered by wind farms in the North Sea

Tongeren, David van (TU Delft Technology, Policy and Management, 2022)

<http://resolver.tudelft.nl/uuid:9976f270-509a-4236-be65-0a71ac95b5e7>

Traversing obstacles – Designing energy infrastructure networks in a geographical cost-differentiated context

Assum, Joost van den (TU Delft Technology, Policy and Management, 2022)

<http://resolver.tudelft.nl/uuid:512431bf-13f5-4c6d-808a-8c961b780d48>

Repurposing Natural Gas Infrastructure for Hydrogen Transmission

Geutjes, Dennis (TU Delft Technology, Policy and Management, 2021)

<http://resolver.tudelft.nl/uuid:ba310d52-b968-463f-8269-d71d600ff662>

Towards a robust European hydrogen network

Huisman, Rowan (TU Delft Technology, Policy and Management, 2021)

<http://resolver.tudelft.nl/uuid:4f15082f-41fe-44fd-8263-1c8e3a53ab18>

Towards a sustainable district heating network in Rotterdam,

de Rooij, Lisanne (TU Delft Technology, Policy and Management, 2020)

<http://resolver.tudelft.nl/uuid:0dbce7db-a8ae-45c0-9491-8c7fefa9706b>

An approach for flexible design of infrastructure networks via a risk sharing contract: The case of CO2 transport infrastructure

Y Melese, P Heijnen, R Stikkelman, P Herder - International Journal of Greenhouse Gas Control, 2017

<https://www.sciencedirect.com/science/article/abs/pii/S1750583616305515>

Regional electricity distribution systems: Designing the future electricity grids of the Netherlands

Rasmussen, Ine Bjørkmann (TU Delft Technology, Policy and Management, 2019)

<http://resolver.tudelft.nl/uuid:1fd3a3ca-51a4-4402-a5de-7e0a35d85b41>

A Geographic Information Systems-based approach for the planning and evaluation of remote DC micro-grid topologies for rural electrification

Tagliapietra, Michele (TU Delft Electrical Engineering, Mathematics and Computer Science; TU Delft DC systems, Energy conversion & Storage, 2019)

<http://resolver.tudelft.nl/uuid:a0ba73b5-8e05-48c1-9b5e-ebb55af4ae14>