**PLANitIO Unit Tests**

Contents

[1. Unit Tests and Integration Tests 3](#_Toc39852193)

[2. Integration Tests 4](#_Toc39852194)

[2.1 Overview 4](#_Toc39852195)

[2.2 General Structure of Integration Tests 4](#_Toc39852196)

[2.3 Input File Formats 5](#_Toc39852197)

[2.4 Standard Results Files – Naming Convention 5](#_Toc39852198)

[2.5 Storing Expected Results in Memory – Link Output 5](#_Toc39852199)

[2.5 Storing Expected Results in Memory – Path Output 6](#_Toc39852200)

[2.6 Storing Expected Results in Memory – Origin-Destination Output 6](#_Toc39852201)

[3 Test Cases 7](#_Toc39852202)

[3.1 Explanatory Test 7](#_Toc39852203)

[3.1.1 test\_explanatory 7](#_Toc39852204)

[3.1.2 test\_explanatory\_attempt\_to\_change\_locked\_formatter 7](#_Toc39852205)

[3.2 Tests for Duplicate External Ids 8](#_Toc39852206)

[3.3 Tests for Reading Initial Cost Values 8](#_Toc39852207)

[3.3.1 test\_reading\_initial\_cost\_values 8](#_Toc39852208)

[3.3.2 test\_reading\_initial\_cost\_values\_with\_missing\_rows 8](#_Toc39852209)

[3.3.3 test\_checking\_for\_missing\_columns\_in\_initial\_costs 8](#_Toc39852210)

[3.4 Tests using the Basic Network 9](#_Toc39852211)

[3.4.1 test\_basic\_shortest\_path\_algorithm\_a\_to\_b\_one\_initial\_cost\_file 9](#_Toc39852212)

[3.4.2 test\_basic\_shortest\_path\_algorithm\_a\_to\_b\_two\_initial\_cost\_files 9](#_Toc39852213)

[3.4.3 test\_basic\_shortest\_path\_algorithm\_a\_to\_c 9](#_Toc39852214)

[3.4.4 test\_basic\_shortest\_path\_algorithm\_a\_to\_d 9](#_Toc39852215)

[3.4.5 test\_basic\_shortest\_path\_algorithm\_a\_to\_b\_three\_time\_periods 9](#_Toc39852216)

[3.5 Route Choice Tests 10](#_Toc39852217)

[3.5.1 test\_route\_choice\_case\_1 10](#_Toc39852218)

[3.5.2 test\_route\_choice\_case\_2 12](#_Toc39852219)

[3.5.3 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_one\_iteration 13](#_Toc39852220)

[3.5.4 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_one\_iteration\_and\_three\_time\_periods 13](#_Toc39852221)

[3.5.5 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_one\_iteration\_using\_link\_segment\_external\_ids 14](#_Toc39852222)

[3.5.6 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_500\_iterations 14](#_Toc39852223)

[3.5.7 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_500\_iterations\_and\_three\_time\_periods 14](#_Toc39852224)

[3.5.8 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_500\_iterations\_using\_link\_segment\_extrnal\_ids 14](#_Toc39852225)

[3.5.9 test\_route\_choice\_case\_3 15](#_Toc39852226)

[3.5.10 test\_route\_choice\_case\_4 17](#_Toc39852227)

[3.5.11 test\_route\_choice\_case\_4\_with\_two\_time\_periods 19](#_Toc39852228)

[3.5.12 test\_route\_choice\_case\_4\_using\_odrawmatrix 19](#_Toc39852229)

[3.5.13 test\_route\_choice\_case\_4\_using\_odrawmatrix\_with\_plus\_sign\_separator 19](#_Toc39852230)

[3.5.14 test\_route\_choice\_case\_5 20](#_Toc39852231)

[3.5.15 test\_route\_choice\_case\_5\_identify\_links\_by\_id 22](#_Toc39852232)

# Unit Tests and Integration Tests

*Unit tests* confirm that individual methods generate the correct result from a known set of input values. They are used to confirm that methods or classes are working correctly, rather than the whole application. If code changes cause a unit test to fail, the developer can immediately review the changes to see what caused the failure.

*Integration tests* test the whole application. They run the whole application using standard input files and confirm that the contents of the output files generated by the program match the expected results.

Unit testing is particularly useful when a team of several developers is working on the same piece of software. In that case each developer must write tests for his own code before integrating it into the larger codebase of the whole project.

We have not written unit tests at the method or class level for the PlanIt application. In the case of PLANit, the development team was small (2 people) and it was sufficient to use integration tests to find issues. Test failures could still be traced back to bugs which caused them.

All tests described in the rest of this document are integration tests.

# Integration Tests

## 2.1 Overview

This section begins with a general description of how the integration tests work, which is applicable to all test cases. It then goes into more detail about the contents of the input and output files which are used in the tests.

## 2.2 General Structure of Integration Tests

Integration tests must use standard input files which define the network, demands and zoning for the test case. In the case of PLANitIO all of these are defined in one XML input file (usually named macroscopicinput.xml).

The expected results should have been generated by another method. These can include:

* hand calculation (if the test case is simple);
* running the same test case in another application. All the tests in the “Route Choice” section were run on OmniTRANS, whose outputs were recorded for comparison;
* expected error messages, where a test has deliberately invalid inputs in order to test PLANit’s input validation methods.

The expected results generated from these alternative calculation methods are stored in two ways:

* CSV and XML output files;
* Java data storage objects, usually Java Maps.

PLANitIO stores results from its runs using objects called *output formatters*. It uses two output formatters:

* PLANitOutputFormatter, which writes results to CSV and XML files;
* MemoryOutputFormatter, which stores results in Java memory.

The contents of the CSV results files generated by PLANitOutputFormatter can be compared to the standard results CSV files, and the values stored in Java memory objects are compared to those stored in MemoryOutputFormatter.

The XML results files generated by PLANitIO contain details of the output configuration used in the test, the output CSV files storing the results and the columns included in the output files. But they also contain a timestamp which gives the date and time of the PLANitIO run. This timestamp should never be the same for any two different runs. The tests must check that all the contents of the created XML file *except the timestamp* are the same as those in the standard results file, but the value of the timestamp in the generated file must always differ from that in the standard results file.

Each integration test must include:

* an XML input file to define the inputs for the test case;
* Java code to run PLANit to read the input file, run the traffic assignment and save the results;
* CSV files to store the expected results (except for tests with invalid inputs to test PLANit’s input validation);
* XML output files;
* Java code to store the expected results in Java;
* Java code to compare the results from the test run with the expected results.

Java contains its own libraries which can read CSV files (Apache Commons CSV) and check that results from a test run match the expected values (JUnit).

The class PlanItIOTestHelper.java contains common code which is used to set up and run PLANit for these tests. This class defines:-

* default configuration methods which are used in many of the tests but not all of them;
* utility methods which are used for common file actions (e.g. comparing the contents of one file with another, deleting CSV results files after the test has finished with them).

The class PlanItIOIntegrationTest contains the individual test cases described in Section 3.

## 2.3 Input File Formats

Each test case has an XML input file called macroscopicinput.xml which contains its input data in the standard XML format (using the <macroscopicdemand>, <macroscopicnetwork> and <macroscopiczoning> elements.

## Standard Results Files – Naming Convention

CSV standard results files use the naming convention:

<OutputType>\_Time Period <Time period number>\_<Number of Iterations>.csv

Where:

<OutputType> is one of “Link”, “Path” or “Origin-Destination”

<Time period number> is the appropriate time period number for the assignment execution

<Number of iterations> is whatever was defined as the maximum number of iterations in the test (for most tests this is either 1 or 500).

So typical CSV output file names include “Link\_Time Period 1\_500.csv”, “Origin-Destination\_Time Period 2\_1.csv” etc.

XML output files follow a similar naming convention but do not include the number of iterations in their title (number of iterations is included in their content anyway). So the equivalent names to the above examples would be “Link\_Time Period 1.xml” and “Origin-Destination\_Time Period 2.xml”

It can be seen that if a test case has more than one time period it will produce more than one set of output files. Several tests have three time periods; for these tests the generated results for all time periods are checked.

## 2.5 Storing Expected Results in Memory – Link Output

Tests of the contents of the MemoryOutputFormatter use data transfer objects (DTOs). These objects are populated with expected result values in the Java code, and then stored in Java Maps. After the traffic assignment run has finished, the values stored in the MemoryOutputFormatter can be compared to these standard results in the code.

Expected results for link output are stored in ResultDto objects. A ResultDto object is populated by its constructor call, which has the following arguments:

* startNodeId external id of start node (used to define the link segment);
* endNodeId external id of end node (used to define the link segment);
* linkFlow flow through link (output);
* linkCost cost (travel time) of link (output);
* totalCostToEndNode cumulative travel time from start of output path to the end of the current link (output);
* capacity capacity of the link (input);
* length length of the link (input);
* speed travel speed of the link (input).

ResultDto objects are stored in a Java Map whose keys are run id, time period and mode. Test cases may have more than one run, time period or mode so this allows all the results to be stored for each.

The fifth argument in the ResultDto constructor, total cost to end node, is not currently used. This was originally included to provide a sort order for the ResultDto objects in the Map which stores them. This is quite helpful for human inspection of output files (such as CSV output files) but it not required for tests against the contents of a MemoryOutputFormatter.

## Storing Expected Results in Memory – Path Output

## Storing Expected Results in Memory – Origin-Destination Output

# 3 Test Cases

## 3.1 Explanatory Test

This test is in the “explanatory/xml” sub-directory.

This test corresponds to the “Basic PLANitIO Input File” test case described in the ReadMe.md file. It is the simplest test case possible, with one link, one user class, one time period and a demand of one unit from one node to the other.

This test has turned out to be very useful *because* it is so trivial. If you make a code change which causes this test to fail, you know you have made a mistake or mistype in your code changes to cause the failure. And it is usually easy to trace back to the coding error which causes a wrong result in this case.

### 3.1.1 test\_explanatory

This runs the explanatory test in its basic form.

### 3.1.2 test\_explanatory\_attempt\_to\_change\_locked\_formatter

This test attempts to change a locked formatter, by trying to run the traffic assignment, changing the output properties in the link output type configuration and running the assignment again. This test is expected to throw a PlanItException when it attempts to change the locked formatter file.

This test uses the same input files as test\_explanatory.

## 3.2 Tests for Duplicate External Ids

All these tests are in the “duplicate\_tests/xml” sub-directory.

* test\_for\_duplicate\_link\_segment\_external\_id
* test\_for\_duplicate\_link\_segment\_type\_external\_id
* test\_for\_duplicate\_mode\_external\_id
* test\_for\_duplicate\_node\_external\_id
* test\_for\_duplicate\_time\_period\_external\_id
* test\_for\_duplicate\_user\_class\_external\_id
* test\_for\_duplicate\_zone\_external\_id

All these tests have a duplicate external Id in their input. The unit tests pass if the code throws an appropriate exception. No standard results files are used in these tests, each test is expected to throw an exception before getting to the point of writing any output.

## 3.3 Tests for Reading Initial Cost Values

These tests verify the that initial cost files are read in correctly and their values are correctly stored in link segment objects. They do not run the traffic assignment.

These tests use the same input network as the “Basic Shortest Path” test cases, but they do not run the traffic assignment anyway. The network files are only included to allow the CustomPlanItProject object to be instantiated.

### 3.3.1 test\_reading\_initial\_cost\_values

Tests that the values of an initial costs file are read in by start and end node and registered by PlanItProject, and that the stored values match the expected ones. Links are identified by link external Id.

### 3.3.2 test\_reading\_initial\_cost\_values\_with\_missing\_rows

Tests that the read in initial costs values match the expected ones when there are some rows missing in the standard results files

### 3.3.3 test\_checking\_for\_missing\_columns\_in\_initial\_costs

Tests that PlanItProject throws an exception when the initial costs file references are link segment which has not been defined

## 3.4 Tests using the Basic Network

These tests use the basic network used in the Strategic Transport Planning course (page 122 of the 2019 course presentation slides).

All these test cases use a demand of 1 unit, with Node A in the diagram being the origin. The destination is Node B, C and D for test cases 1, 2 and 3 respectively. The optimal paths for each of these test cases can easily be calculated by hand. The tests check that the results from the traffic assignment match these hand-calculated results.

Some of these test cases use initial cost files, to test that the methods for reading initial costs work with one or two initial cost files. The contents of the initial cost files are the same as the network link costs, so they do not affect the results.

### 3.4.1 test\_basic\_shortest\_path\_algorithm\_a\_to\_b\_one\_initial\_cost\_file

Time Period 1 uses route A to B in the example, which has a total route cost of 85

### 3.4.2 test\_basic\_shortest\_path\_algorithm\_a\_to\_b\_two\_initial\_cost\_files

Time Period 1 uses route A to B in the example, which has a total route cost of 85

### 3.4.3 test\_basic\_shortest\_path\_algorithm\_a\_to\_c

This test case uses route A to C in the example, which has a total route cost of 77.

### 3.4.4 test\_basic\_shortest\_path\_algorithm\_a\_to\_d

This test case uses route A to D in the example, which has a total route cost of 108.

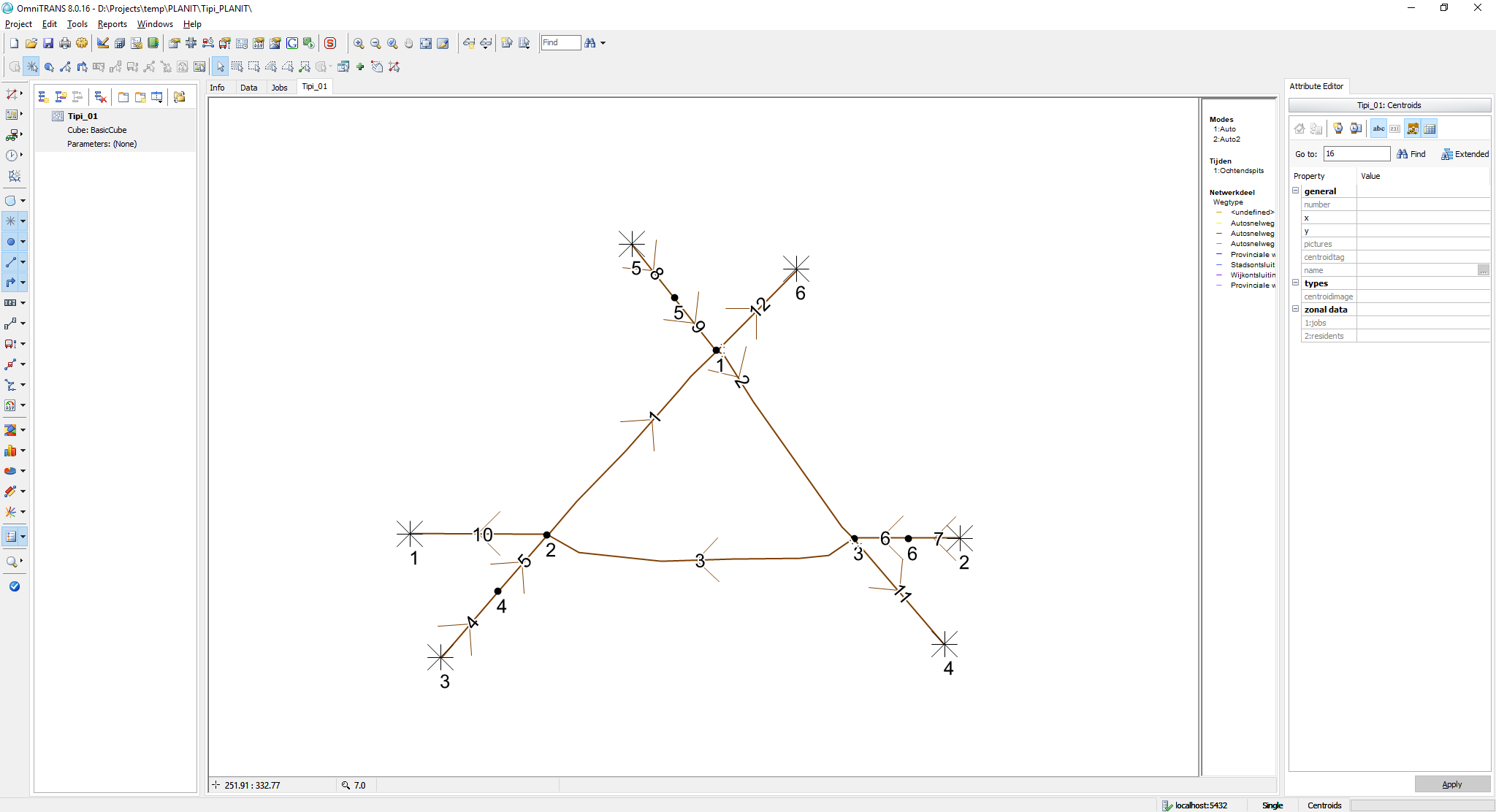
### 3.4.5 test\_basic\_shortest\_path\_algorithm\_a\_to\_b\_three\_time\_periods

Time Period 1 uses route A to B in the example, which has a total route cost of 85. Time Period 2 use the route A to C, which has a total route cost of 77. Time Period 3 uses route A to D in the example, which has a total route cost of 108.

## 3.5 Route Choice Tests

The Route Choice Tests are more complicated test cases whose results must match corresponding tests run in OmniTRANS with the same networks.

### 3.5.1 test\_route\_choice\_case\_1



**Length:**

* All links have a length of ***1 km***

**BPR**:

* All links use a BPR cost function with ***alpha: 0.5*** and ***beta: 4.0***

**Link properties:**

* All links have a maximum speed of ***100 km/h*** and a capacity of ***2000 veh/h/lane***, except for link 2 which has a capacity of only ***1000/veh/h/lane***
* All links have ***1 lane,*** *except links 4,7 and 8*who have ***10 lanes***

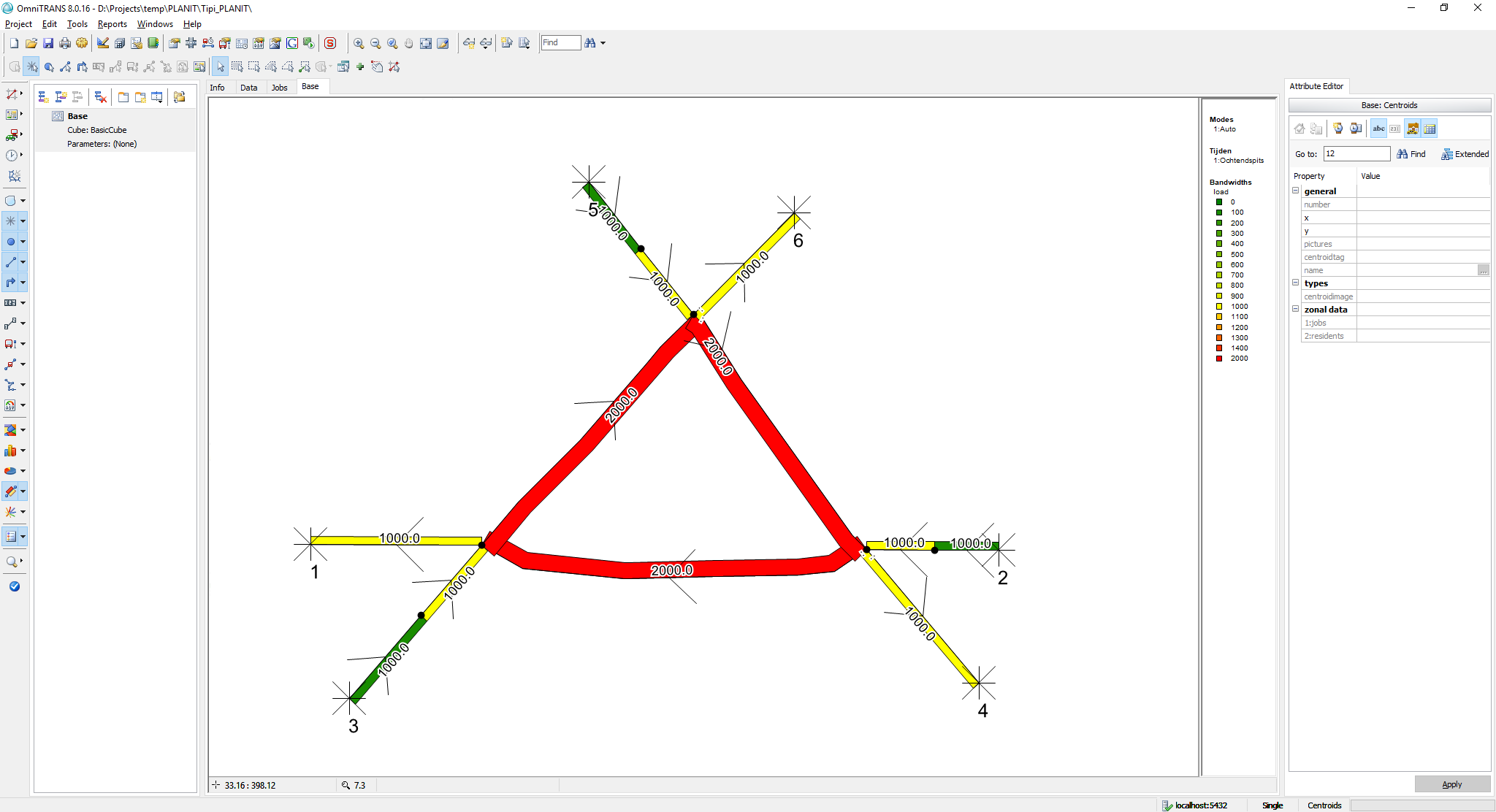
**Demand:**

* Travel demand is only non-zero on ODs 3,4 and 5,1 and 2,6. All three ODs have a travel demand of ***1000*** ***veh/h***

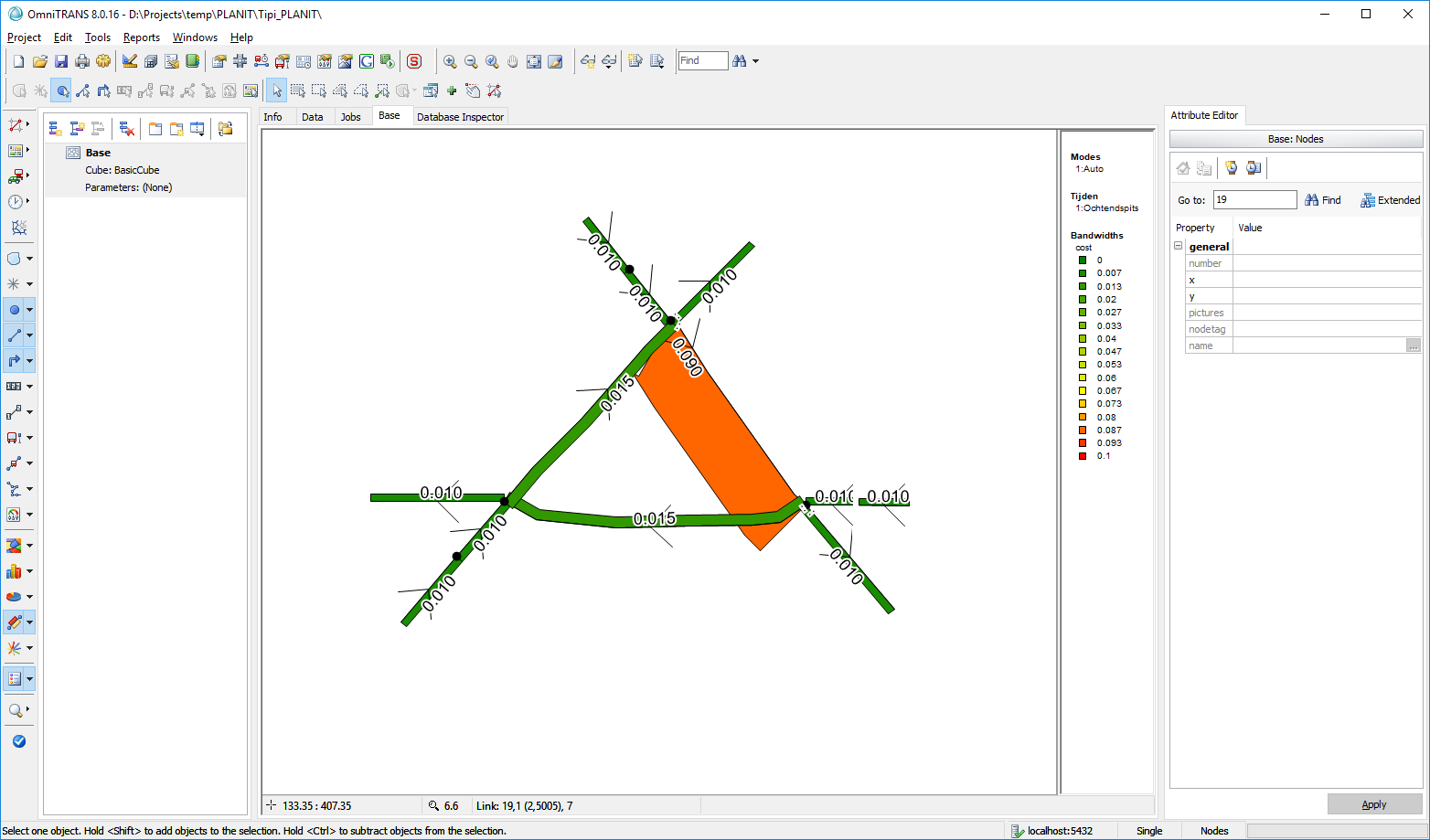
**Simulation:**

* Simulation time, i.e. period is ***1 h***.
* Single mode only

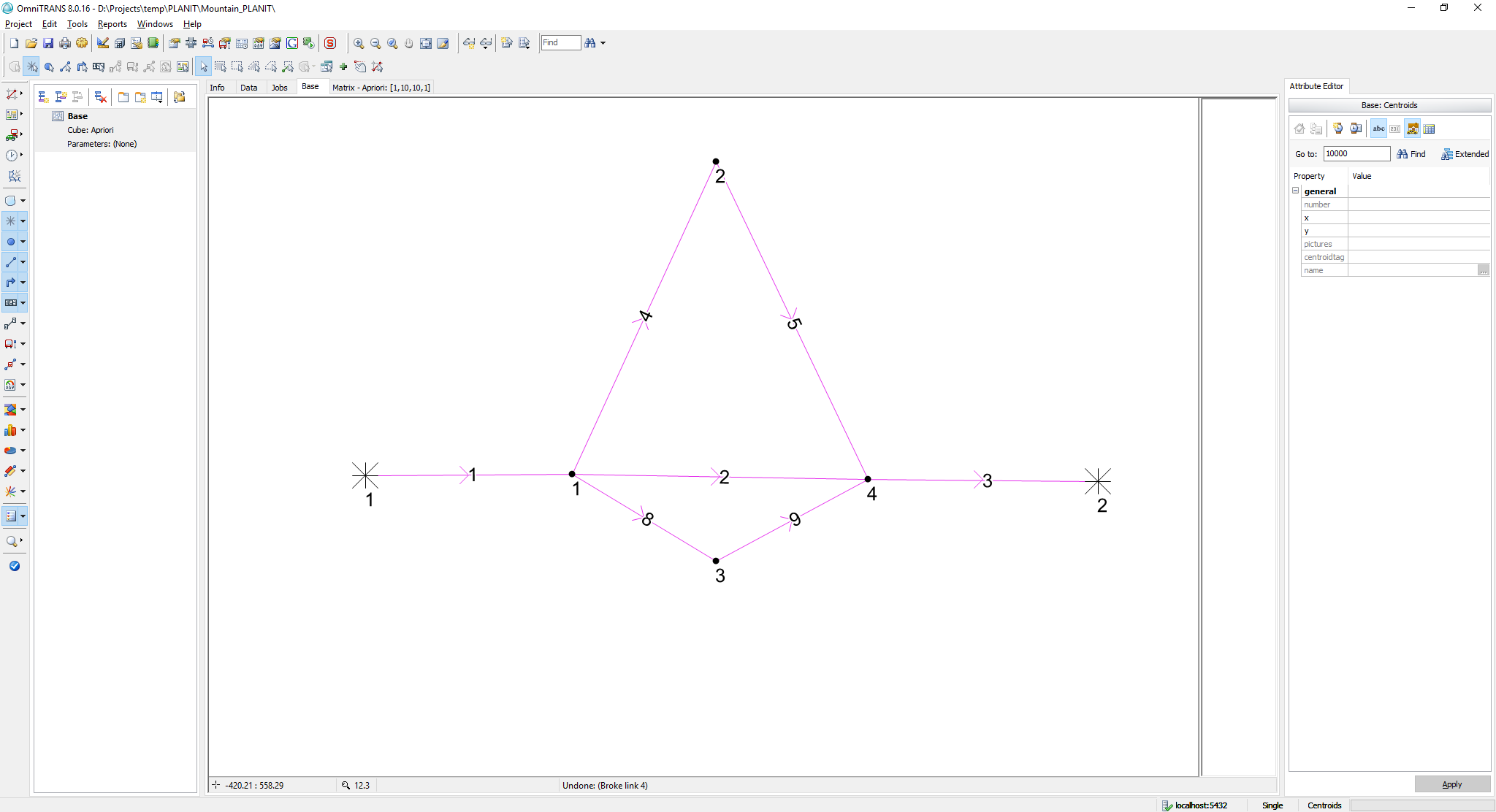
The resulting flows should be the following



Travel times (h) on the links are the following:



### 3.5.2 test\_route\_choice\_case\_2



**Length:**

* All links have a length of ***1 km***, except for links 4 and 5 which are ***2 km*** in length

**BPR**:

* All links use a BPR cost function with ***alpha: 0.5*** and ***beta: 4.0***

**Link properties:**

* All links have a maximum speed of ***60 km/h*** and a capacity of ***1200 veh/h/lane***.
* All links have ***1 lane***, except for links 1 and 3 who have ***3 lanes***

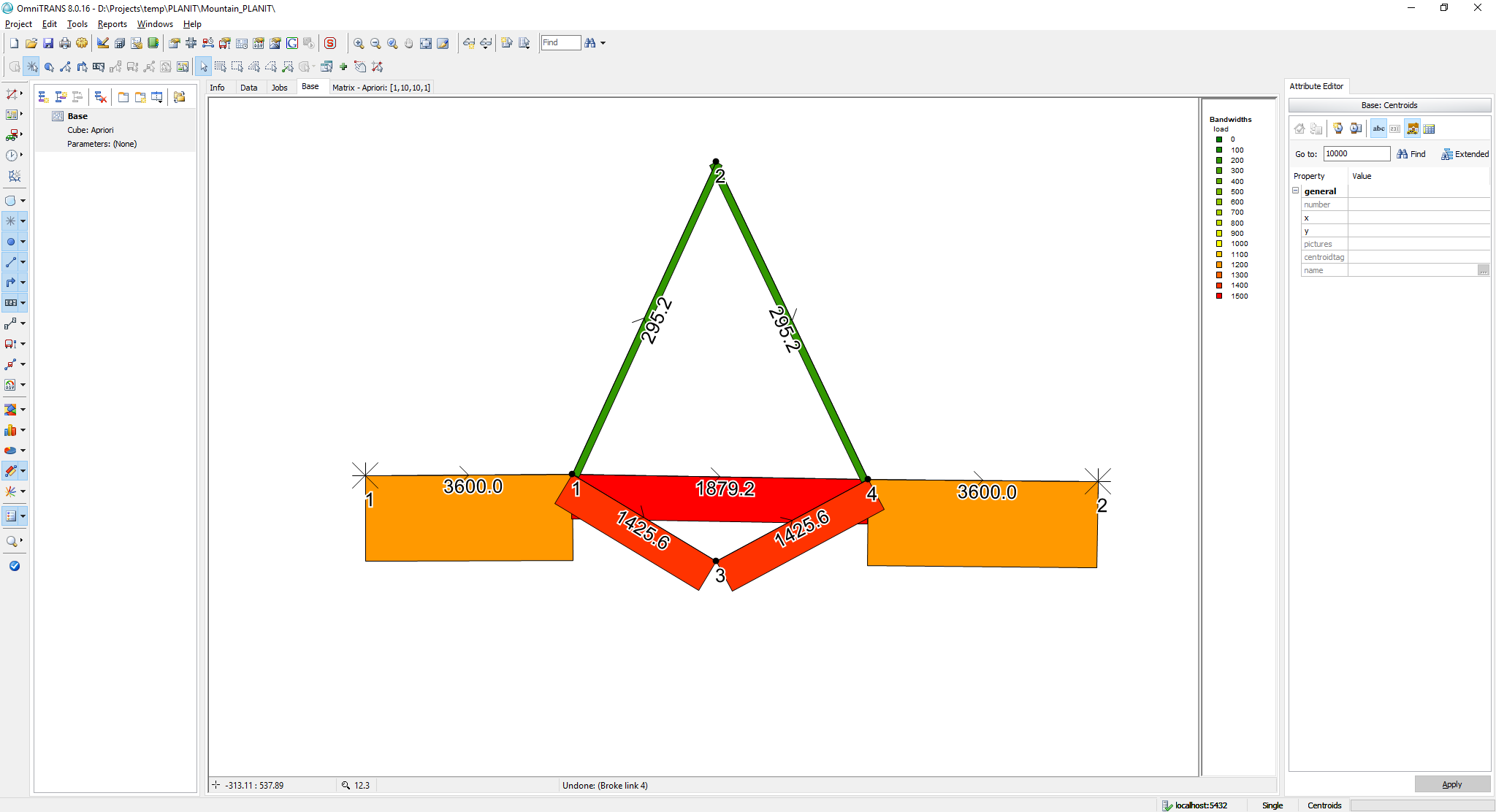
**Demand:**

* The travel demand from 1 🡪 2 is set to ***3600 veh/h***

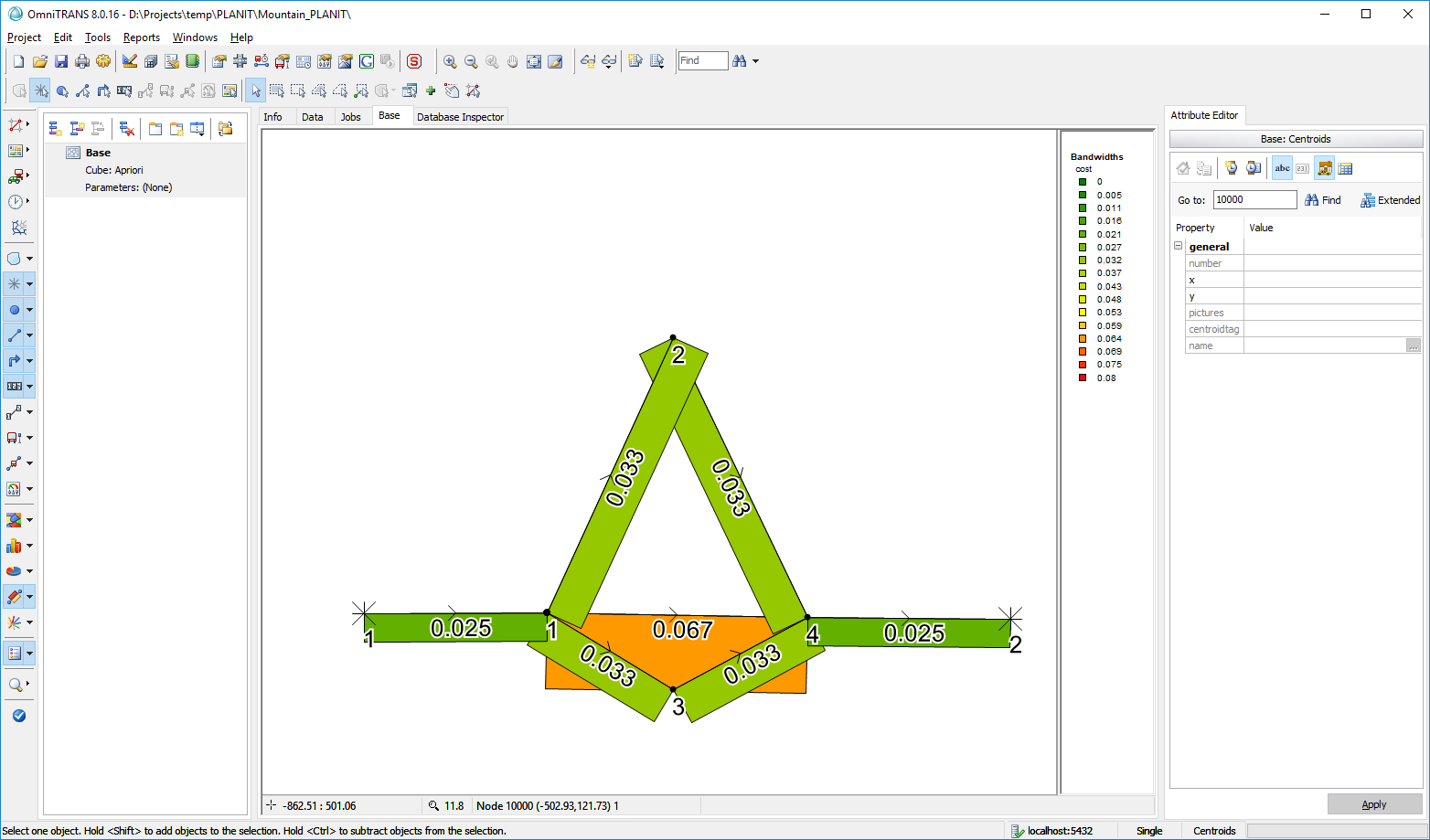
**Simulation:**

* Simulation time, i.e. period is ***1 h***.

The equilibrium result after 500 iterations using regular MSA smoothing should yield the following link flow rates (Using OmniTRANS 8.0.16 and OTTraffic):



The generalised link costs in terms of link travel times (h) are:



### 3.5.3 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_one\_iteration

This test checks that PlanItProject reads the initial costs from a file correctly, and outputs them after the first iteration. The test initial costs file uses Link Segment Id to identify link segments.

### 3.5.4 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_one\_iteration\_and\_three\_time\_periods

This test runs the network using one iteration with different initial costs for each time, checking that the results are different for each time period.

### 3.5.5 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_one\_iteration\_using\_link\_segment\_external\_ids

This test checks that PlanItProject reads the initial costs from a file correctly, and outputs them after the first iteration.

### 3.5.6 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_500\_iterations

This test checks that PlanItProject reads the initial costs from a file correctly, and outputs them after 500 iterations. The test input initial costs file uses link segment Id to identify link segments.

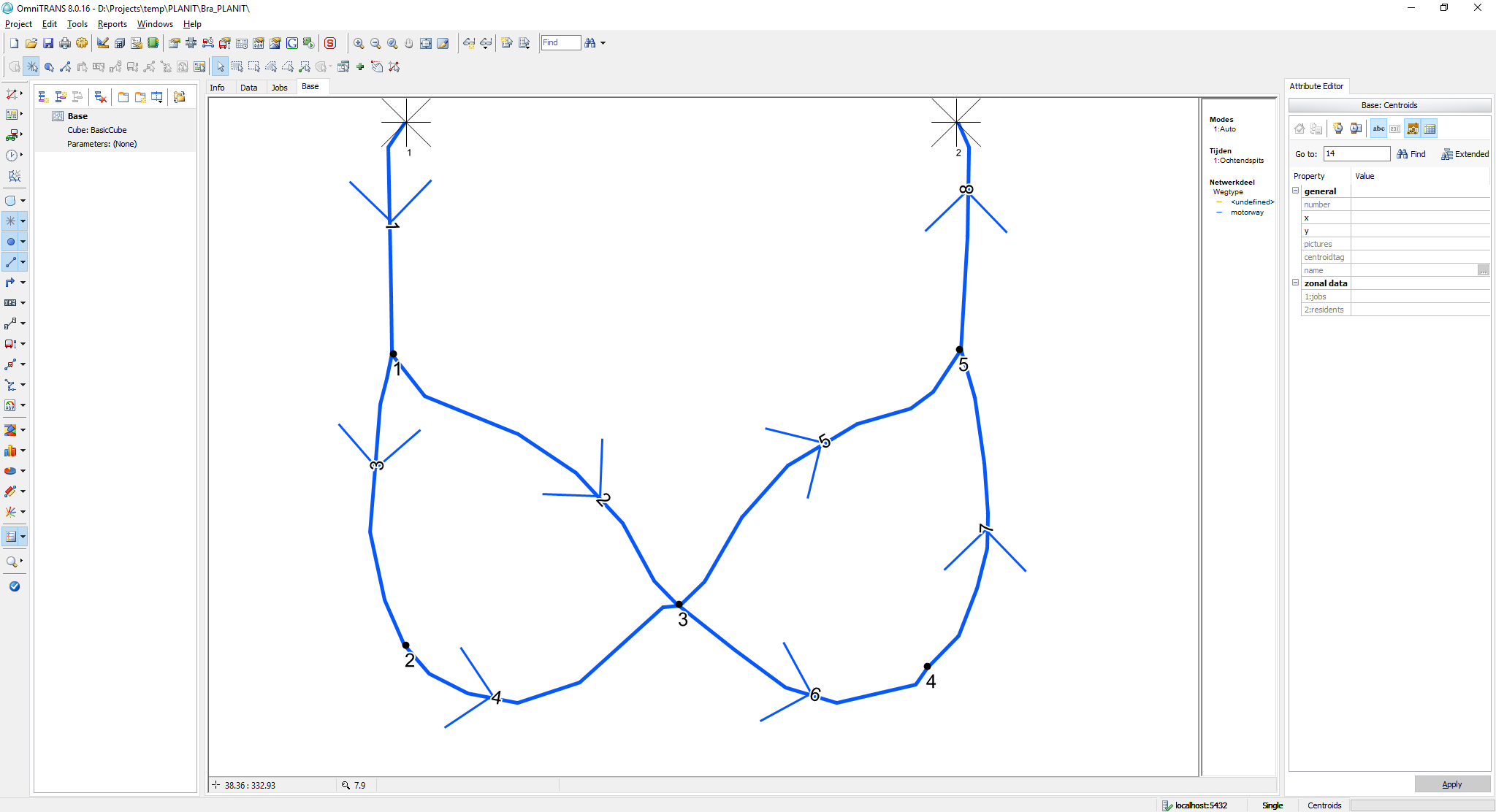
### 3.5.7 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_500\_iterations\_and\_three\_time\_periods

This test runs the network with three time periods with different initial costs for each, running the test for 500 iterations.

### 3.5.8 test\_route\_choice\_case\_2\_with\_initial\_costs\_and\_500\_iterations\_using\_link\_segment\_extrnal\_ids

This test checks that the PlanItProject reads the initial costs from a file correctly, and outputs them after 500 iterations. The test initial costs file uses Link Segment External Id to identify link segments.

### 3.5.9 test\_route\_choice\_case\_3



**Length:**

* All links have a length of ***2 km***

**BPR**:

* All links use a BPR cost function with ***alpha: 0.5*** and ***beta: 4.0***

**Link properties:**

* All links have a maximum speed of ***100 km/h***
* Capacity per lane:
  + **Link 1:** 4 lanes, 2000 veh/h/lane (8000)
  + **Link 2:** 2 lanes, 1500 veh/lane (3000)
  + **Link 3:** 2 lanes, 2500 veh/lane (5000)
  + **Link 4:** 2 lanes, 2000 veh/lane (4000)
  + **Link 5:** 1 lane, 2000 veh/lane (2000)
  + **Link 6:** 2 lanes, 1500 veh/lane (3000)
  + **Link 7:** 1 lane, 2000 veh/lane (2000)
  + **Link 8:** 1 lane, 2000 veh/lane (2000)

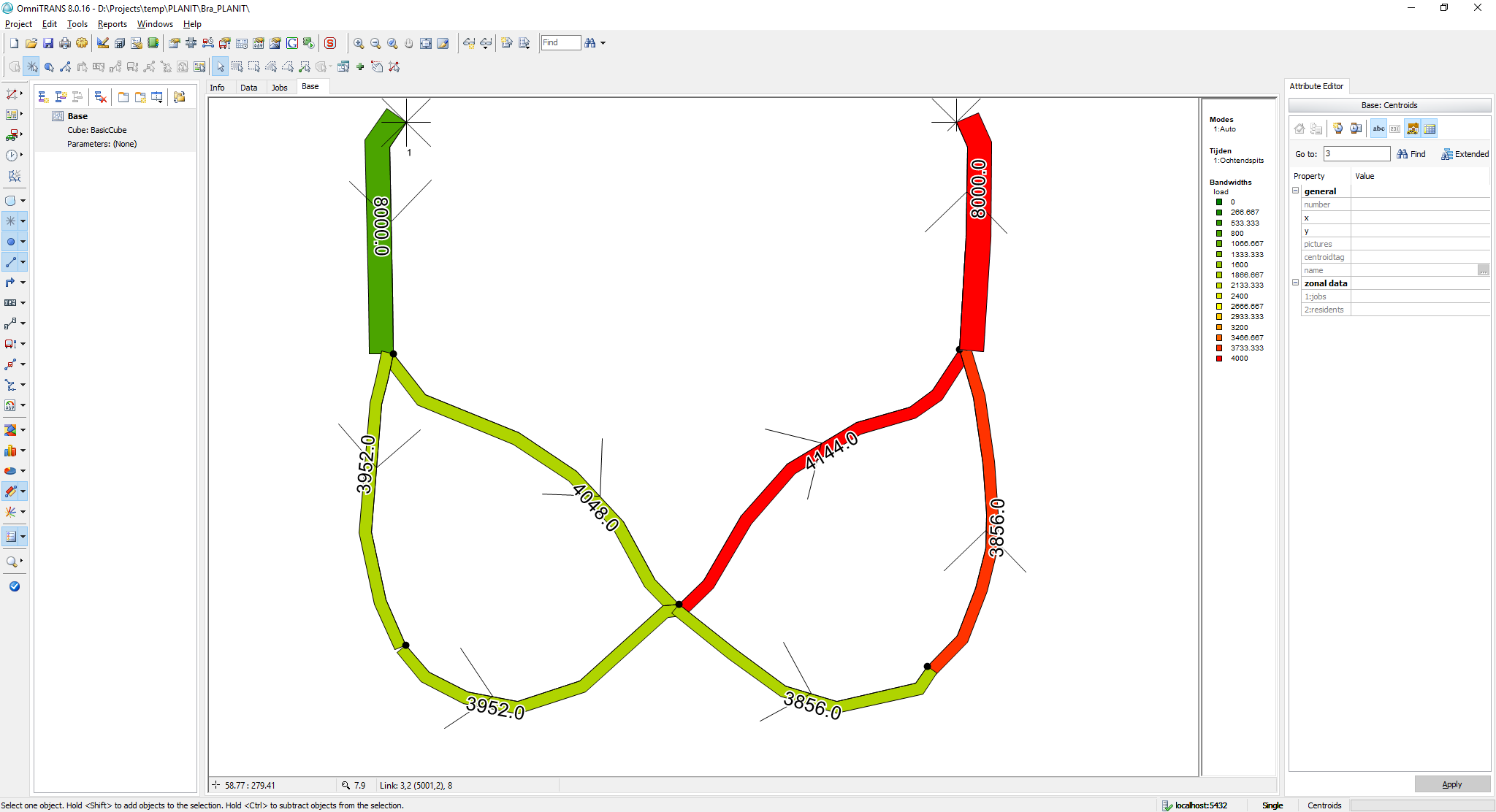
**Demand:**

* The travel demand from 1 🡪 2 is set to **80*00 veh/h***

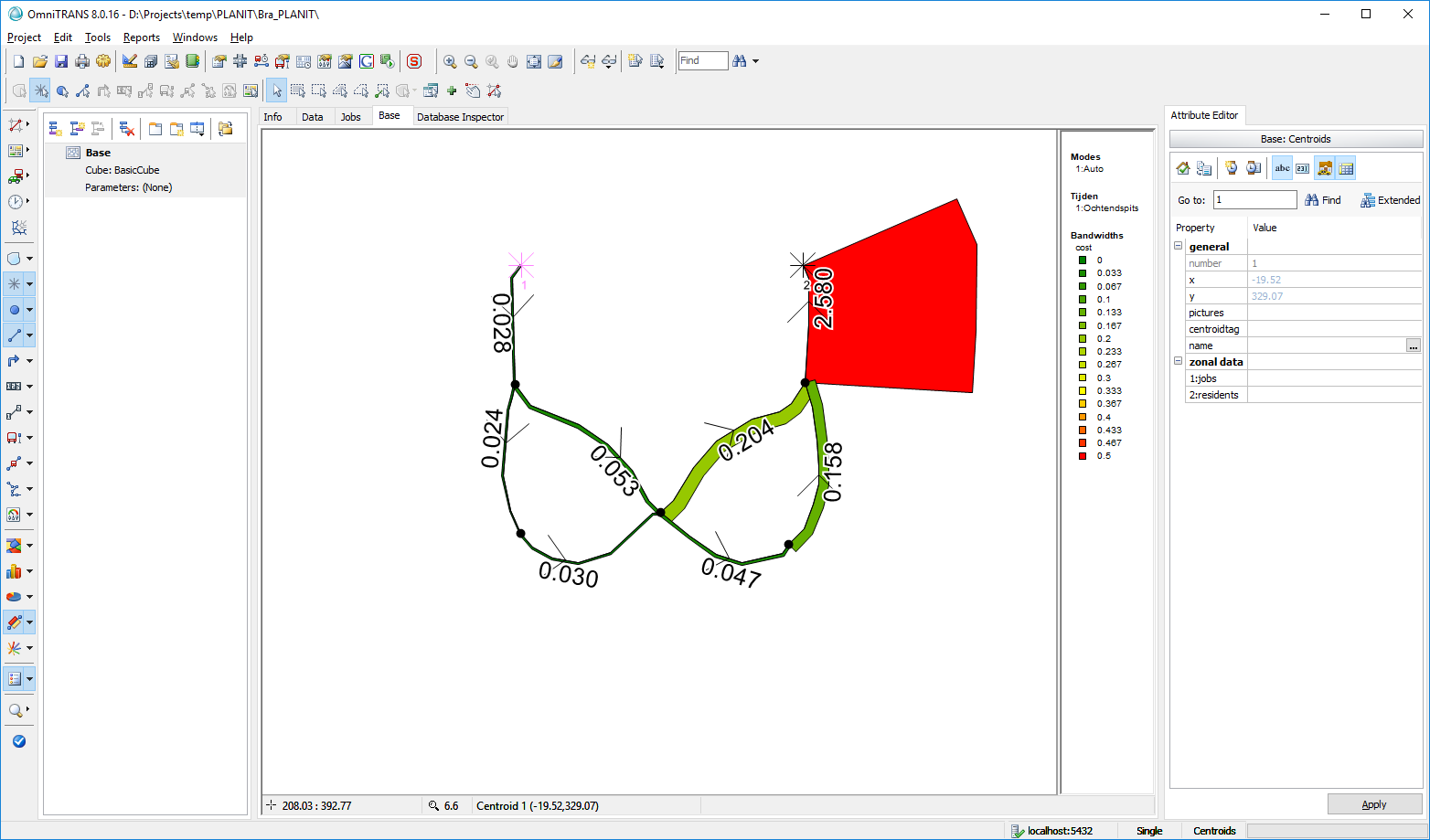
**Simulation:**

* Simulation time, i.e. period is ***1 h***

The equilibrium result after 500 iterations using regular MSA smoothing should yield the following link flow rates (Using OmniTRANS 8.0.16 and OTTraffic):

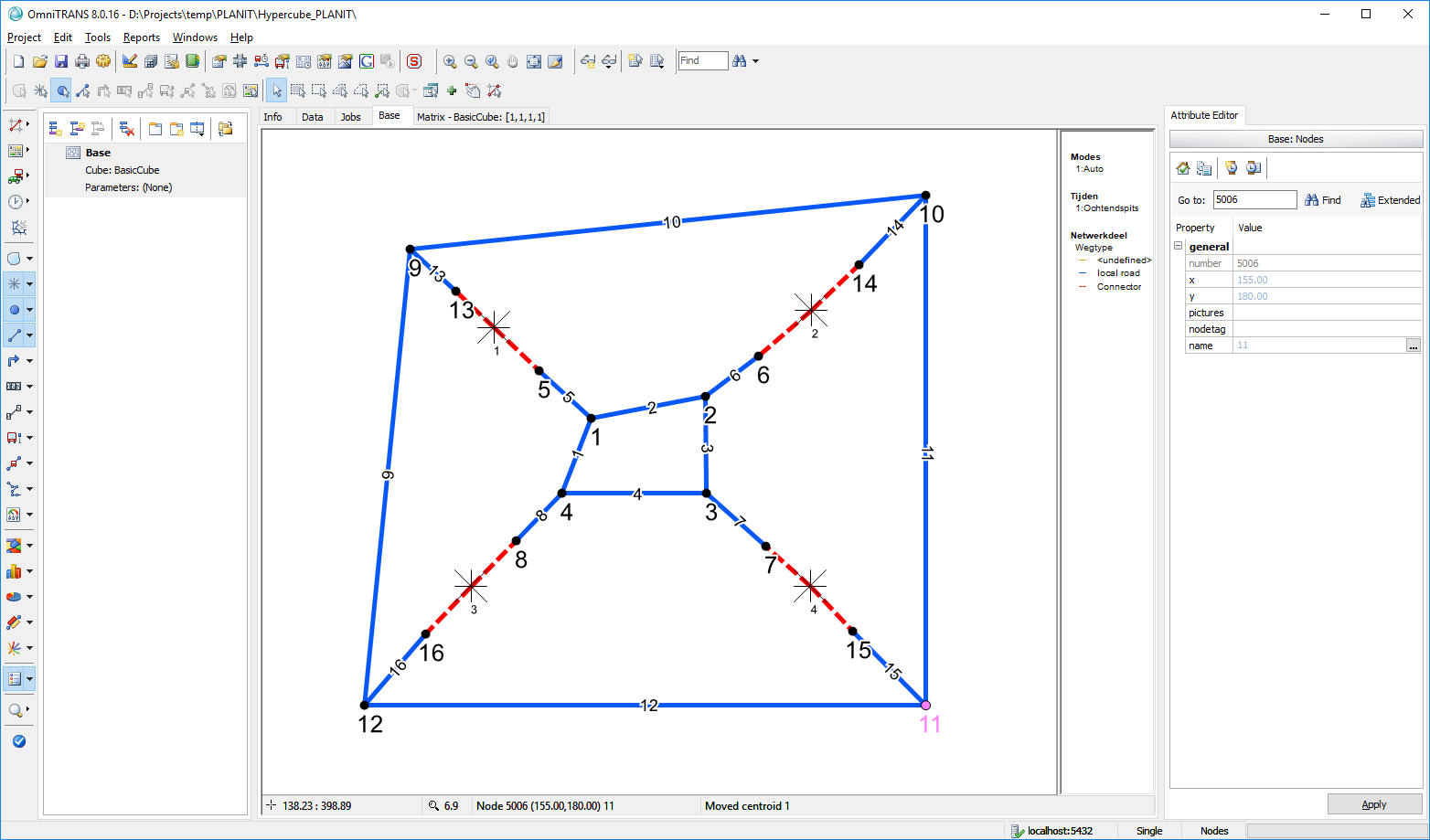


Below are the generalised link costs which consist of travel time (h) only:



### 3.5.10 test\_route\_choice\_case\_4

This test case uses the <odrowmatrix> method in the macroscopicinput.xml file to define the OD demands input matrix.



**Length:**

* All links have a length of **1 *km***except for
  + Links 1 and 2 have a length of **0.9 km**
  + Links 11 and 12 have a length of **3 km**
  + Link 9 and 10 have a length of **2.9 km**

**BPR**:

* All links use a BPR cost function with ***alpha: 0.5*** and ***beta: 4.0***

**Link properties:**

* All links are bi-directional
* Regular (blue) links:
  + have a maximum speed of ***100 km/h***
  + Capacity per lane: **1500 veh/h/lane**
  + Have **1** lane
* Connector links (dashed red)
  + Have a maximum speed of **50 km/h**
  + Capacity per lane **1000 veh/h/lane**
  + Have **10** lanes

**Demand:**

* The origin – destination travel demands are given in the table below

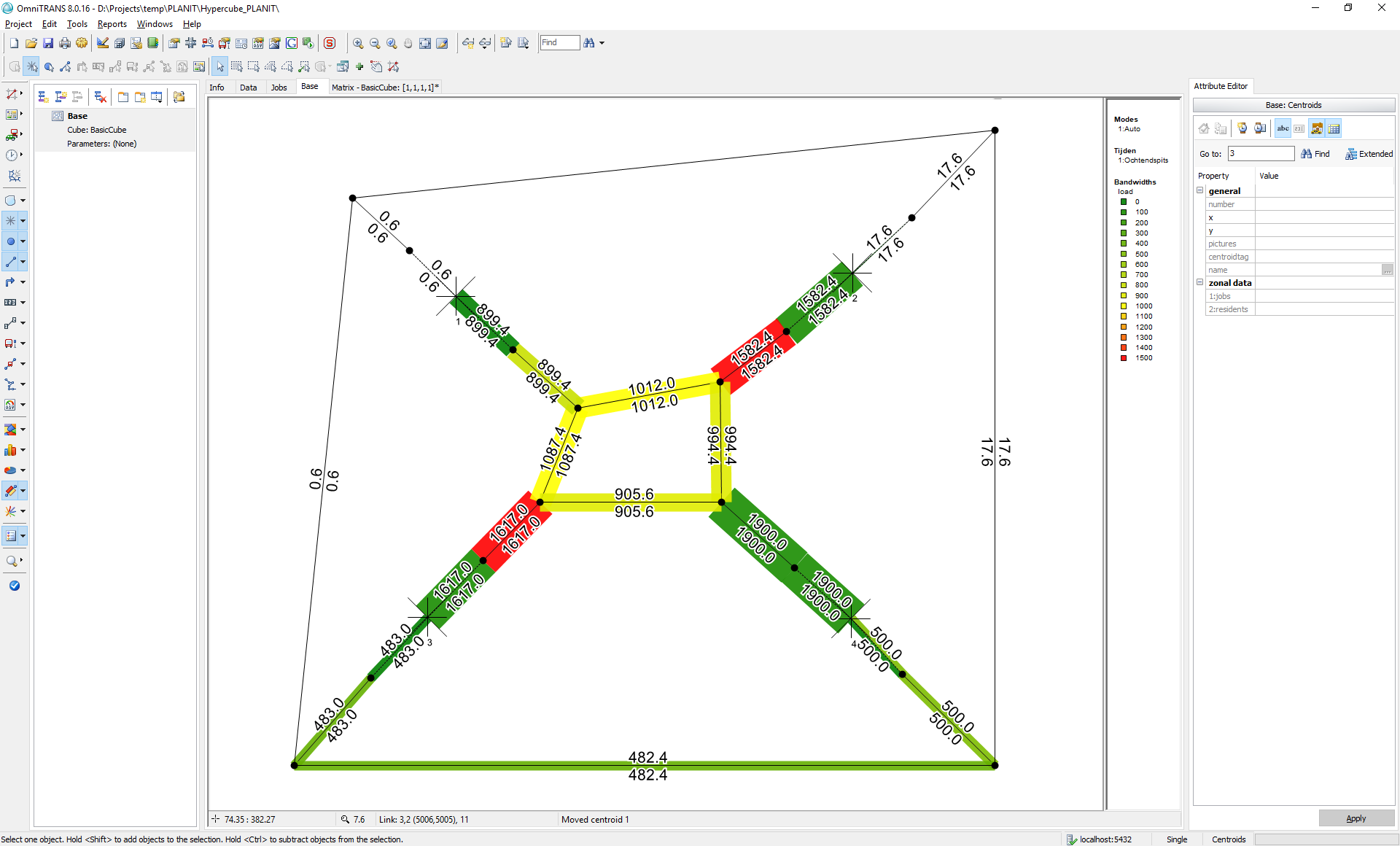
Table : OD travel demands for the simulation duration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **From/to** | **1** | **2** | **3** | **4** |
| **1** | 100.000 | 200.000 | 300.000 | 400.000 |
| **2** | 200.000 | 400.000 | 600.000 | 800.000 |
| **3** | 300.000 | 600.000 | 900.000 | 1200.000 |
| **4** | 400.000 | 800.000 | 1200.000 | 1600.000 |

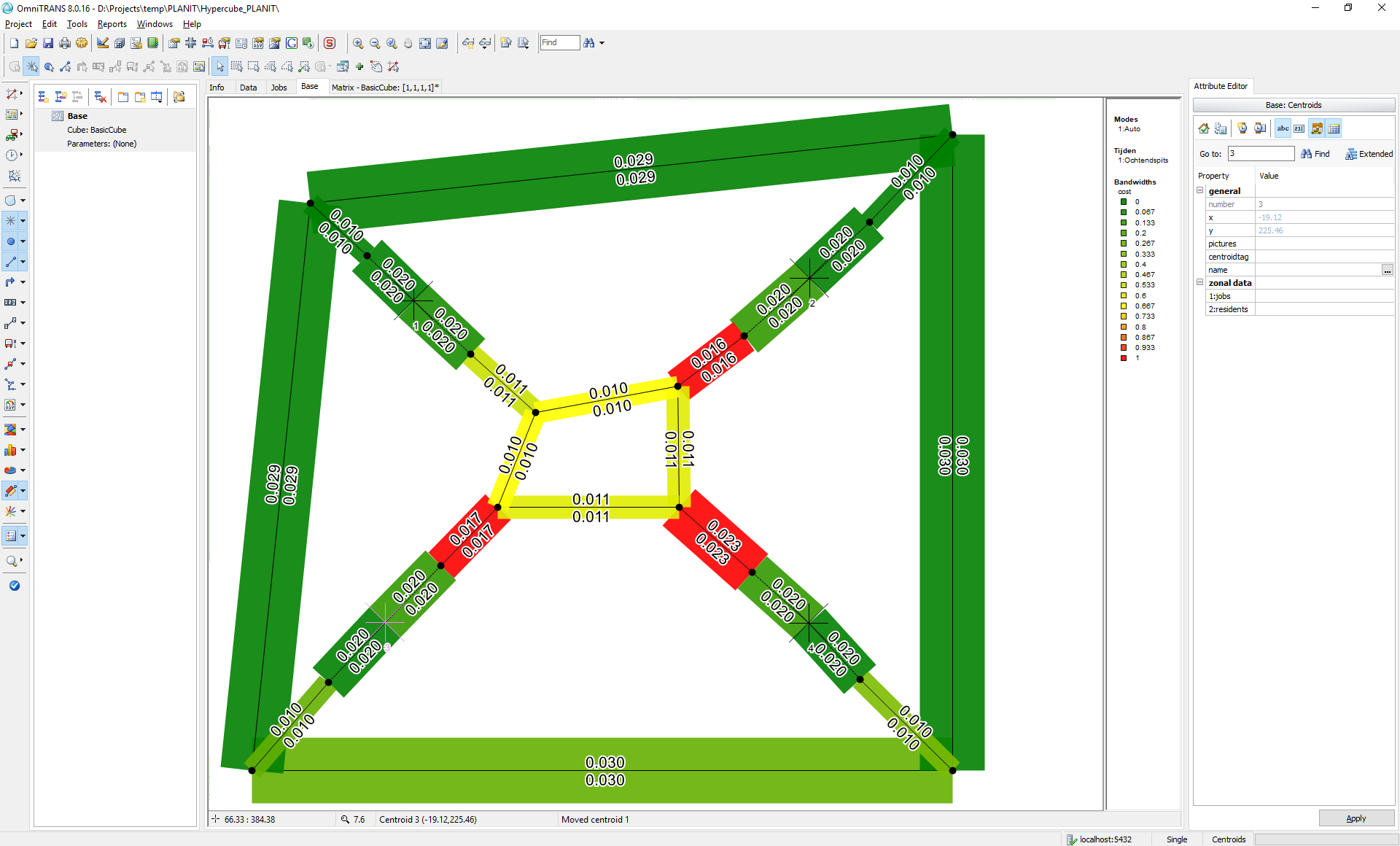
**Simulation:**

* Simulation time, i.e. period is ***1 h***
* 500 iterations
* Epsilon gap set to 0 to ensure we run the full 500 iterations

Link flow results are shown below per link direction:-



The generalised link costs associated with each link (direction) are shown below in travel time (h):



### 3.5.11 test\_route\_choice\_case\_4\_with\_two\_time\_periods

This test has two time periods. It uses the <odrowmatrix> method in the macroscopicinput.xml file to define the OD demands input matrix.

### 3.5.12 test\_route\_choice\_case\_4\_using\_odrawmatrix

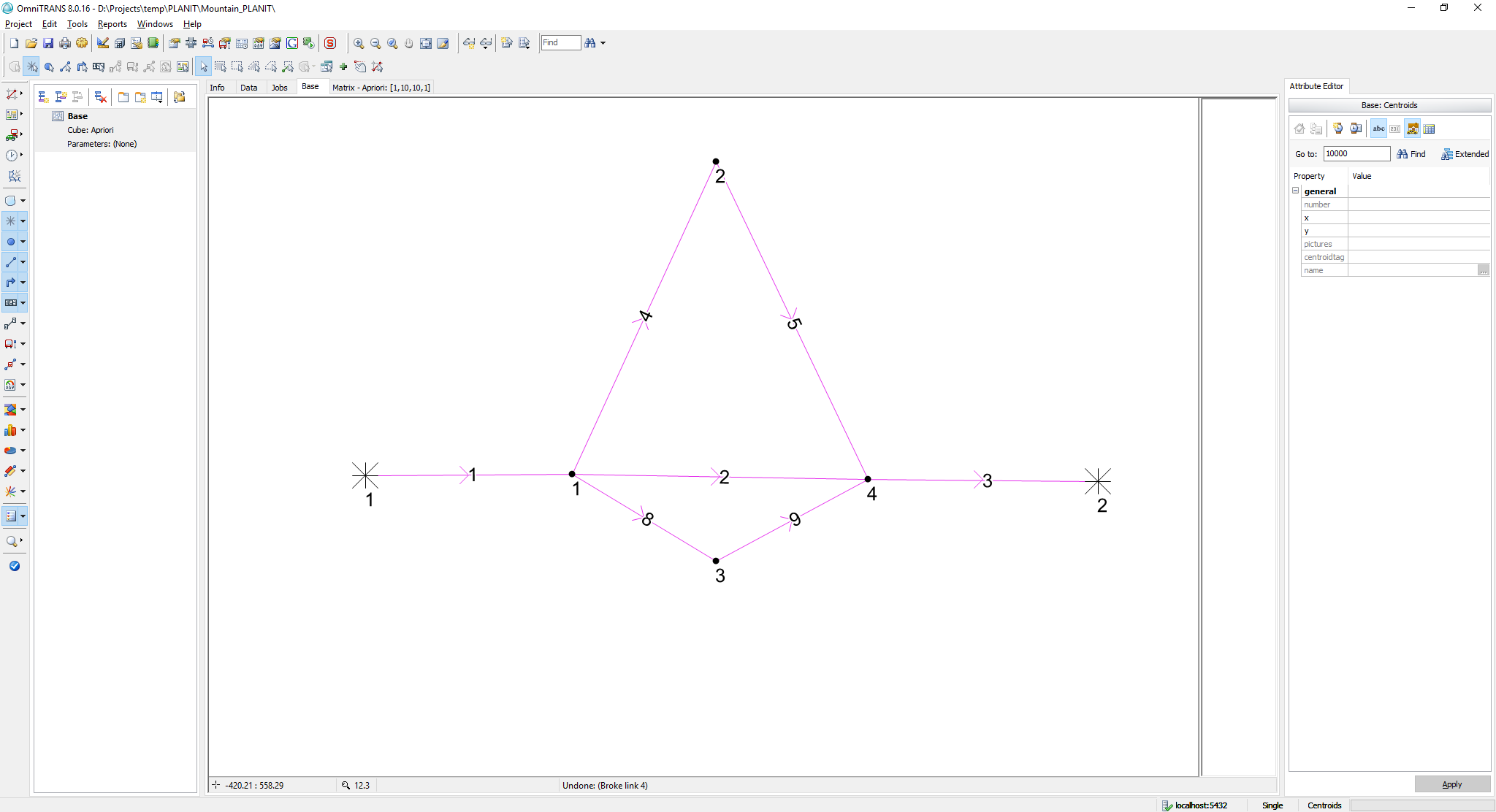
This test uses the <odrawmatrix> method in the macroscopicinput.xml file to define the OD demands input matrix.

### 3.5.13 test\_route\_choice\_case\_4\_using\_odrawmatrix\_with\_plus\_sign\_separator

This test case uses the <odrawmatrix> method with the plus sign separator in the macroscopicinput.xml file to define the OD demands input matrix.

### 3.5.14 test\_route\_choice\_case\_5

This is a testcase for multi-modal assignment using two modes (cars, trucks). Trucks are allowed on all but the middle link (which represents an inner city of some sort. Hence, the network looks like the following:



**Length:**

* All links have a length of ***1 km***, except for links 4 and 5 which are ***2 km*** in length

Modes:

* Mode 1: cars have a pcu=1
* Mode 2: trucks have a pcu of 2.5

**BPR**:

* All links use a BPR cost function with
  + Cars (mode 1): ***alpha: 0.5*** and ***beta: 4.0 for all links***
  + Trucks (mode 2): ***alpha: 0.8*** and ***beta: 4.5 for all links, except link 2 where trucks are not allowed***

**Link properties:**

* All links have a capacity of ***1200 veh/h/lane***.
  + Cars (mode 1): maximum speed of ***60 km/h*** and a
  + Trucks (mode 1): maximum speed of **5*0 km/h***
* All links have ***1 lane***, except for links 1 and 3 who have ***3 lanes***

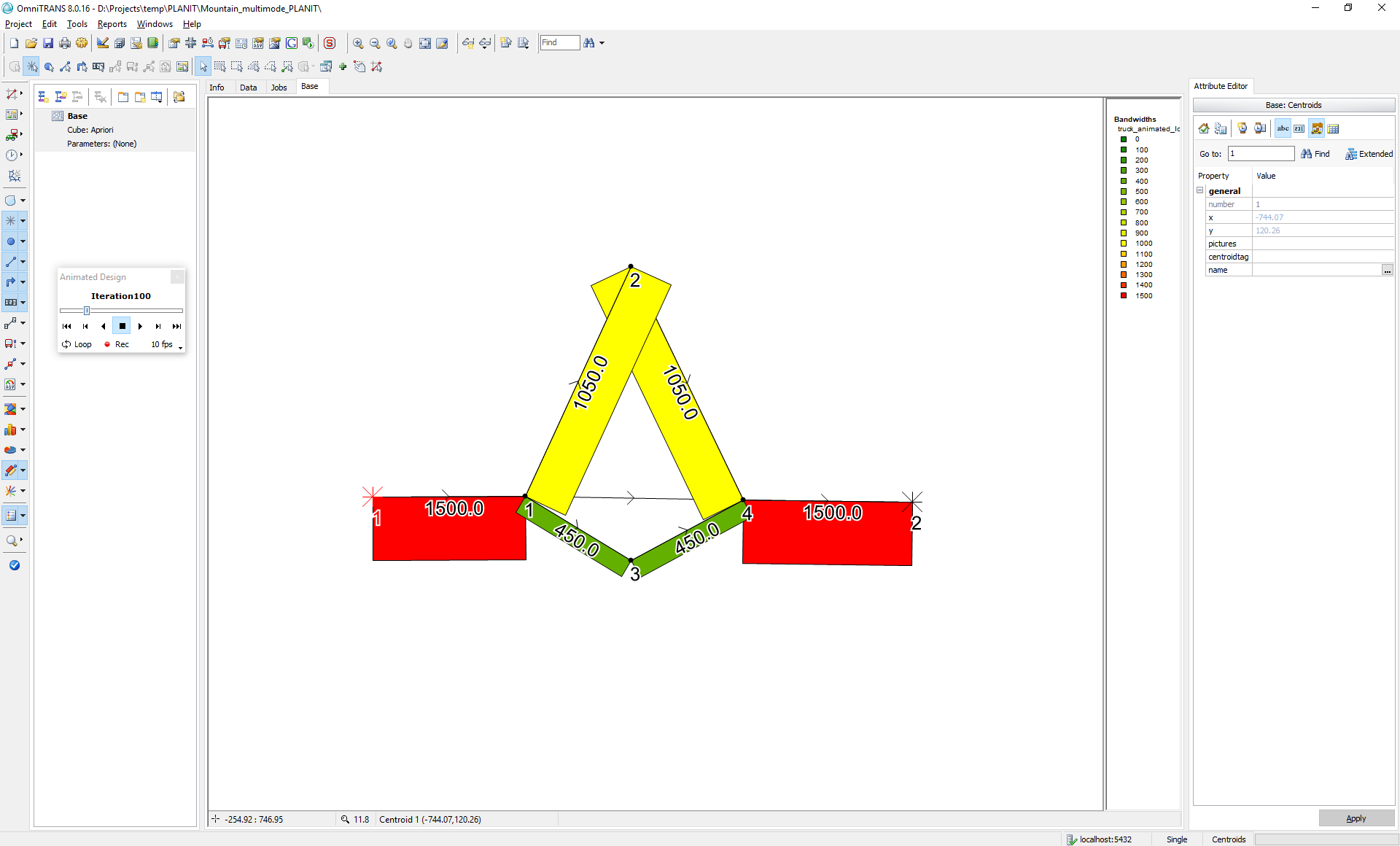
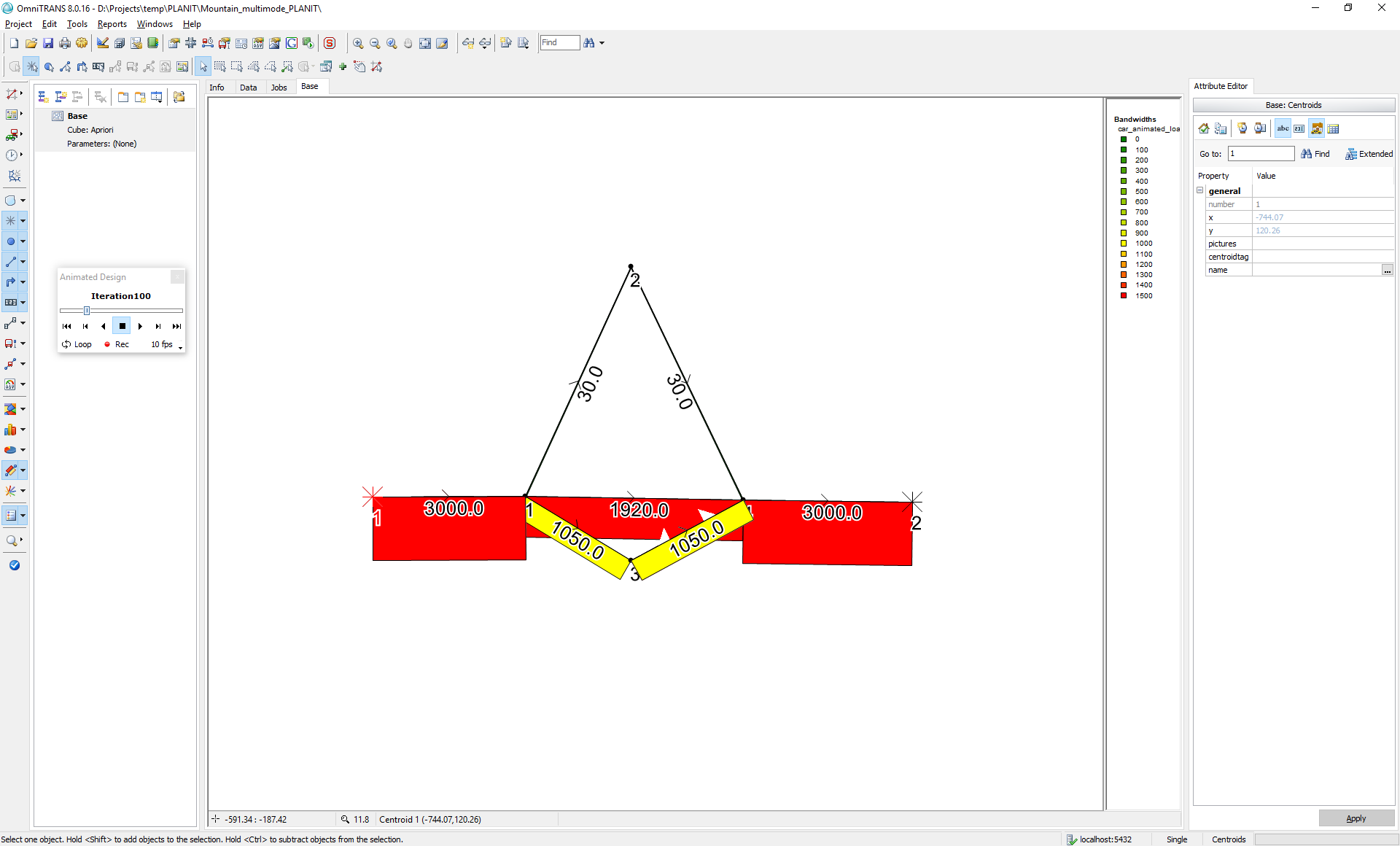
**Demand:**

* The travel demand from 1 🡪 2 is set to
* Mode 1: **3000 veh*/h***
* Mode 2: **600trucks/h**

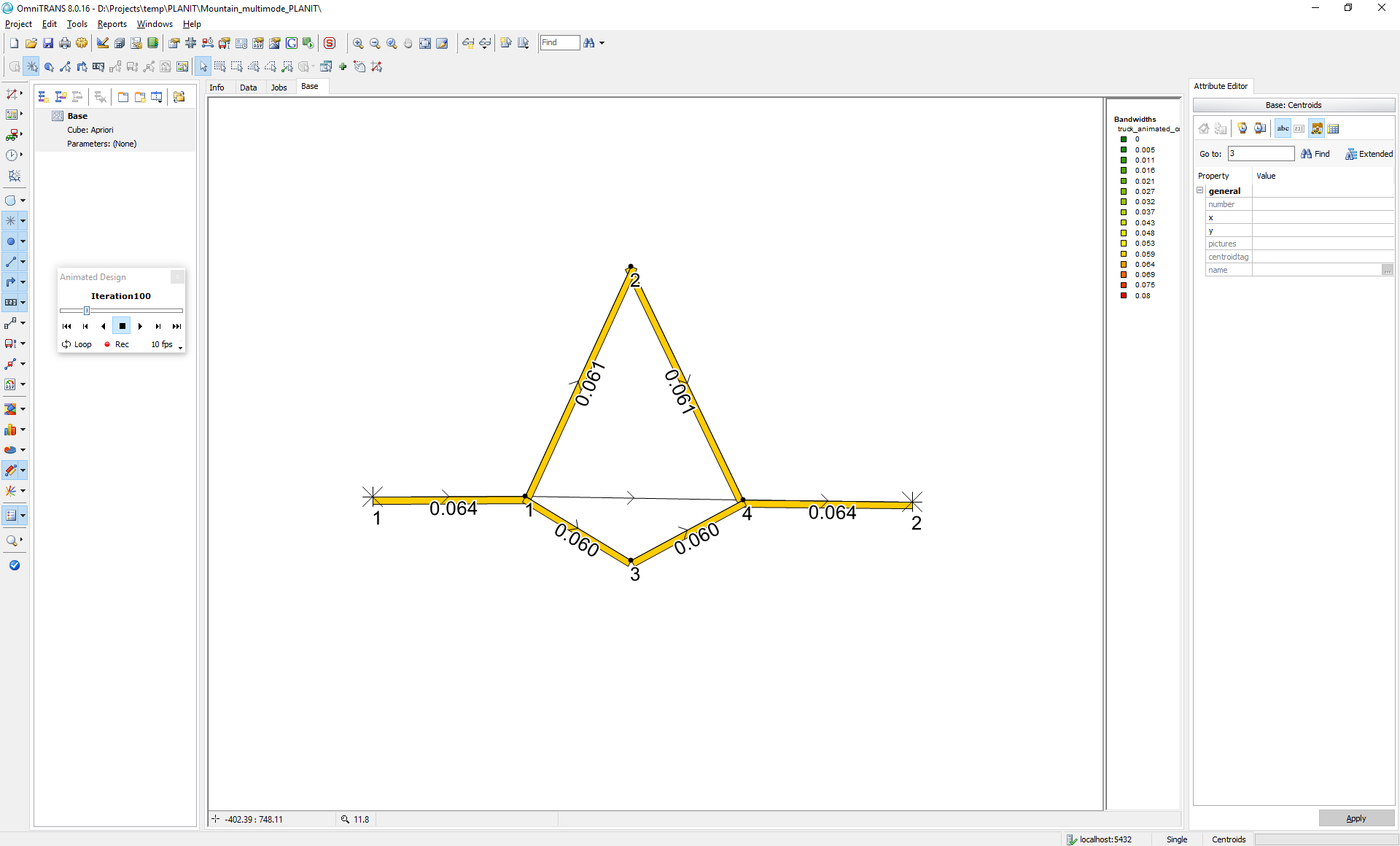
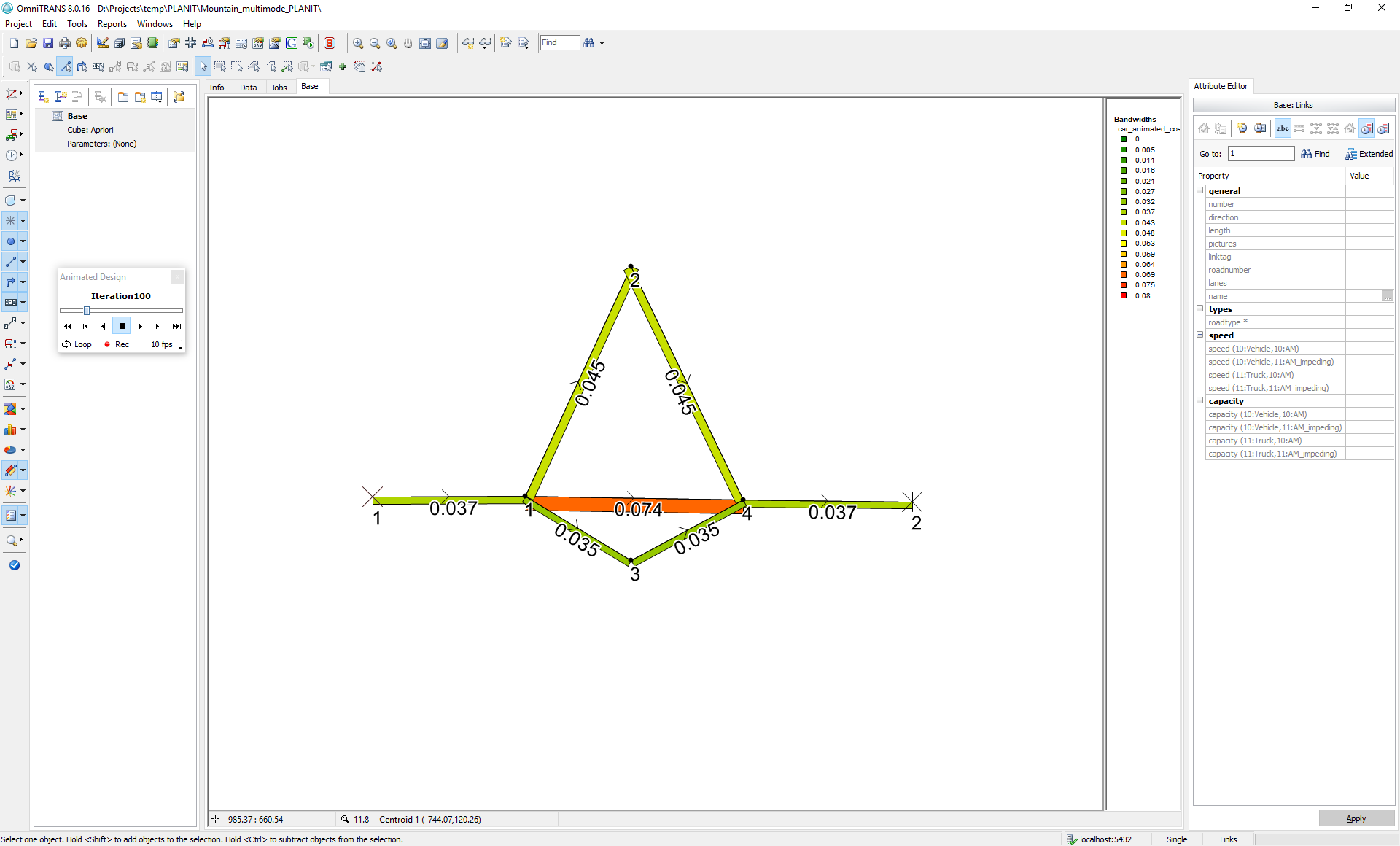
**Simulation:**

* Simulation time, i.e. period is ***1 h***.

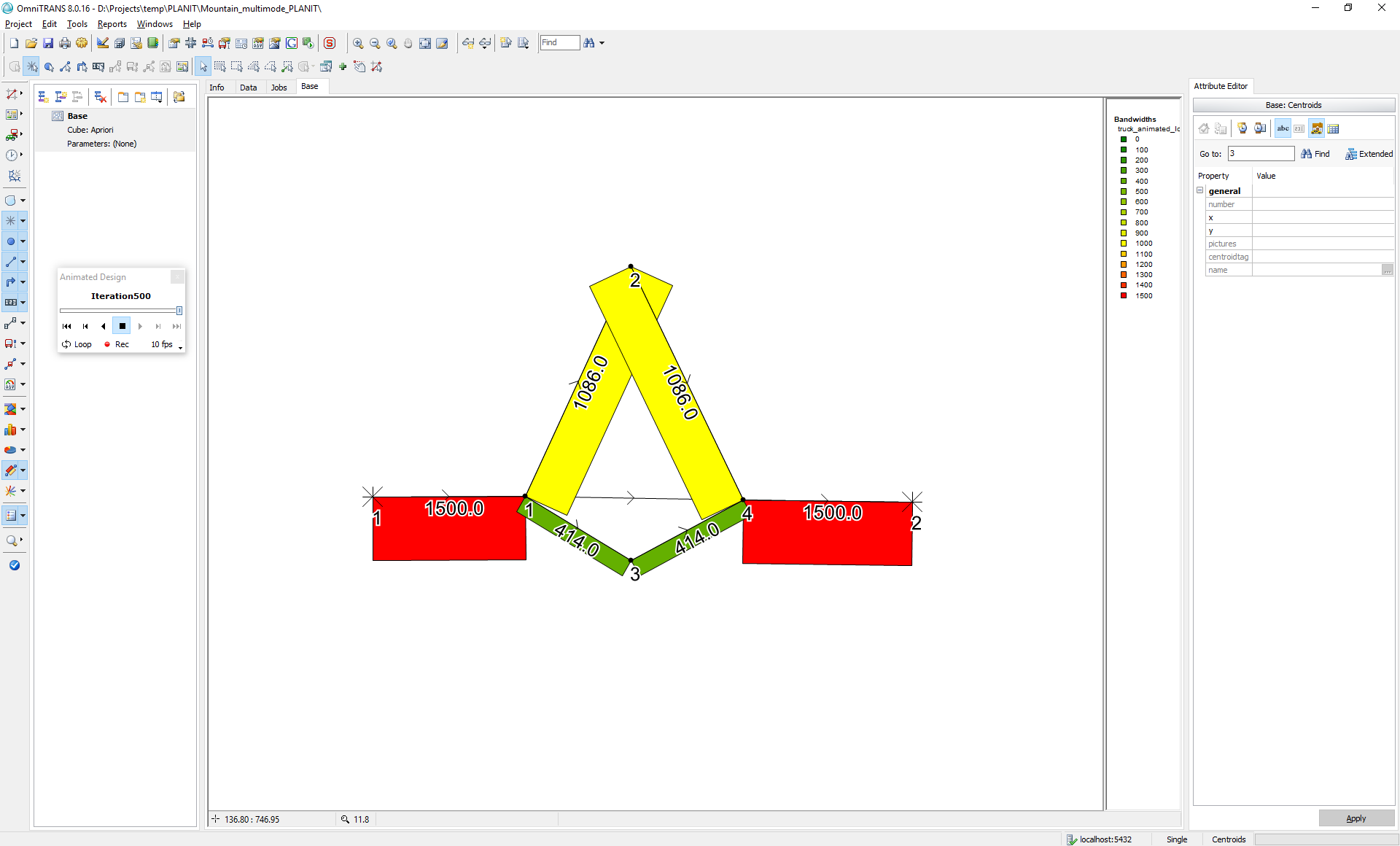
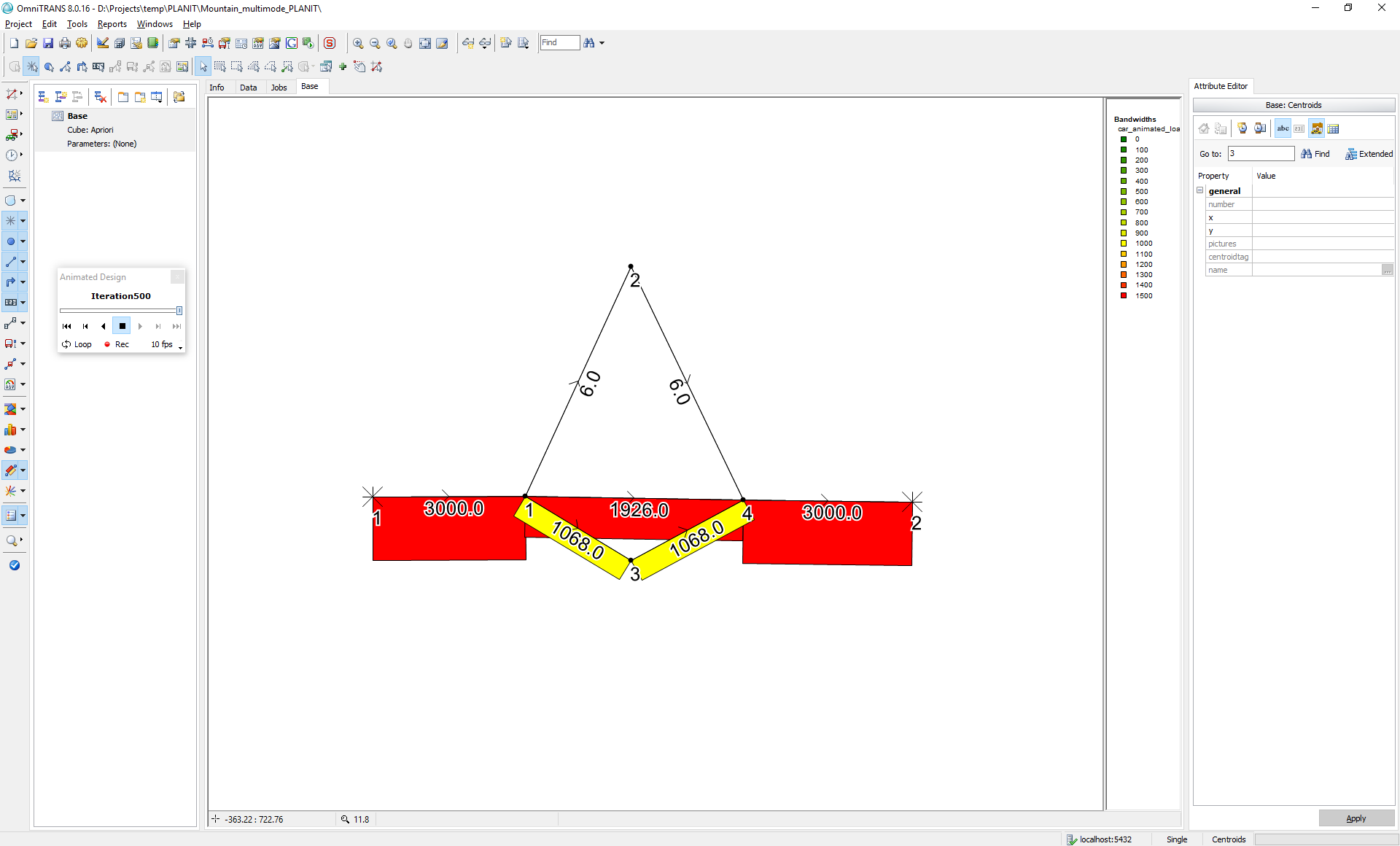
The result after 100 iterations using regular MSA smoothing should yield the following (mode specific) link flow rates (Using OmniTRANS 8.0.16 and OTTraffic):



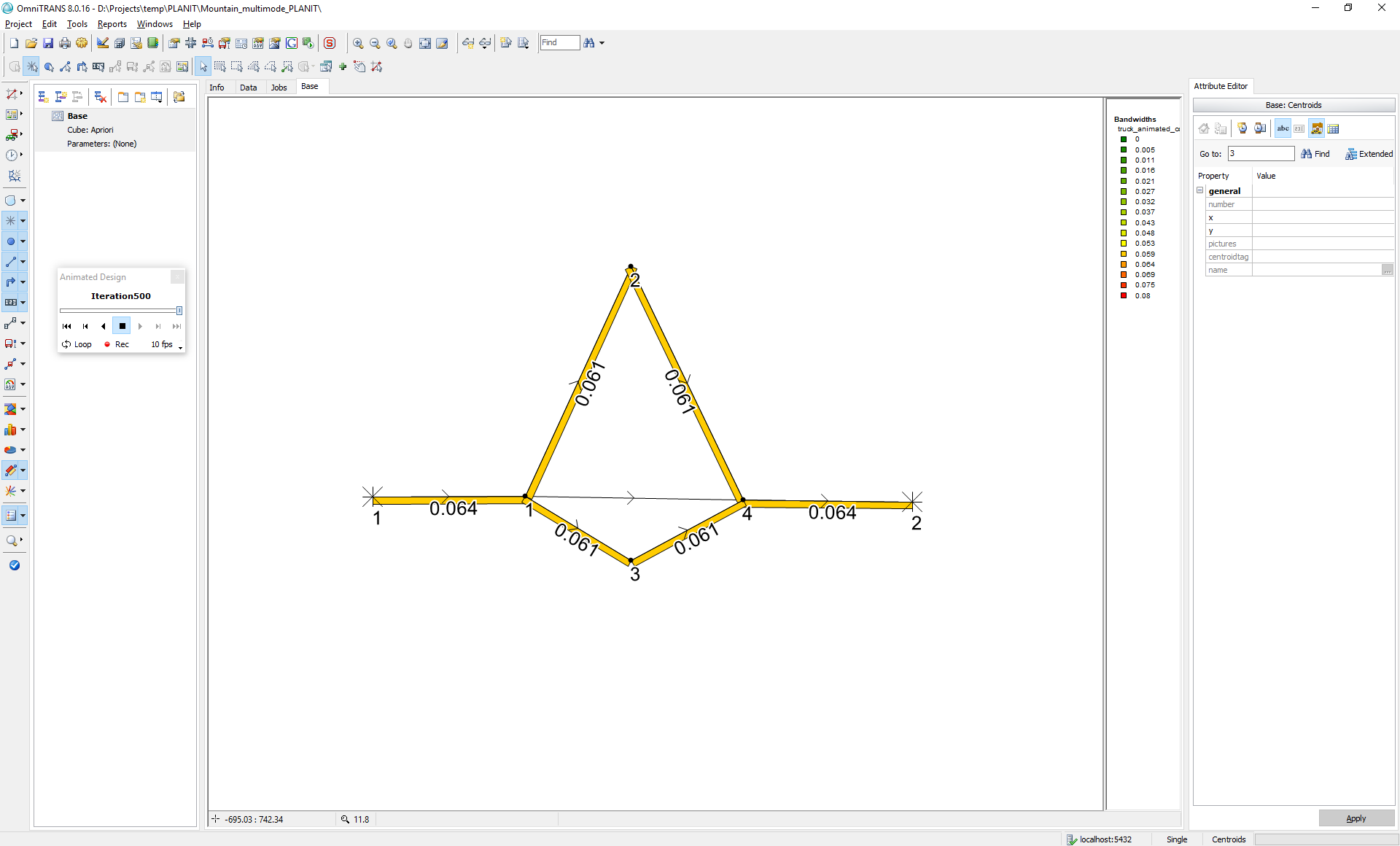
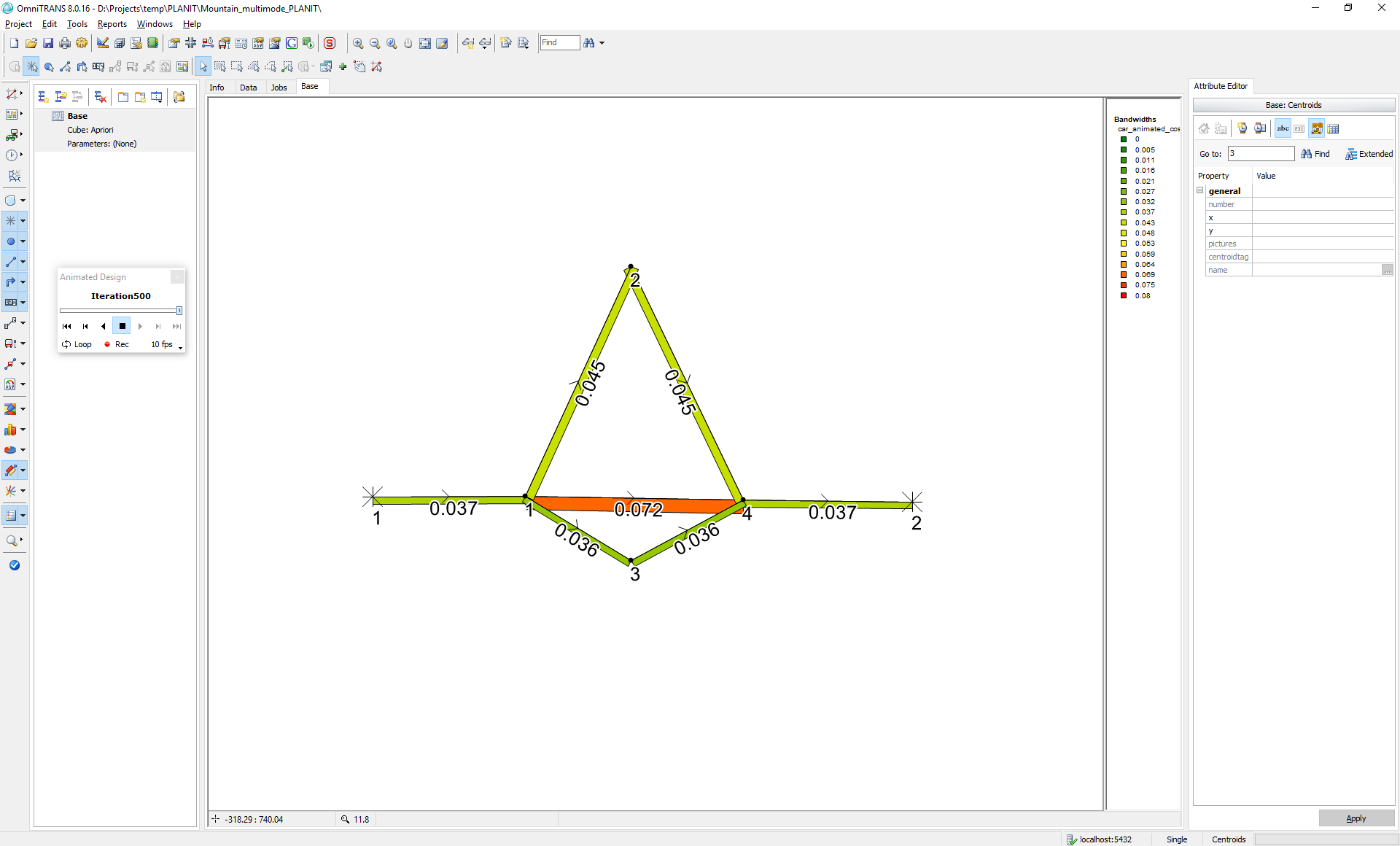
and link costs:



The result after 500 iterations using regular MSA smoothing should yield the following link flow rates (Using OmniTRANS 8.0.16 and OTTraffic):



and link costs:



### 3.5.15 test\_route\_choice\_case\_5\_identify\_links\_by\_id

This test identifies links using link id (all other tests use link external Id).