Nodus « demo » project

Bart Jourquin
2017 - 2022
(Last revision: February 9, 2022)

Center for Operations Research and Econometrics (CORE)
Louvain-la-Neuve and Mons, Belgium
e-mail: bart.jourquin@uclouvain.be

This document presents the basic usage of Nodus (since version 7.0) and the structure of the different files and database tables that are used or created by the software. It is followed by a presentation of different scenarios that can be run using the provided sample Nodus project.

Nodus 7.1 is functionally identical to Nodus 7.0, but embeds the Groovy 2.5.x scripting language instead of Groovy 2.4.x. This can potentially (but very unlikely) break some scripts written by end users. The use of Groovy 2.5.x also introduces a change in the directory structure of Nodus, affecting the Java classpath. This change should, however, remain transparent for the end users.

Nodus 7.2 embeds Groovy 3.xx.

In addition to a lot of minor improvements, **Nodus 8.0** adds the possibility to define "transit time" functions along with "cost" functions. This version also improves the "exclusions" mechanism that allows defining the type of operations that can be performed or not at the node level. Nodus 8.0 also introduces a simplified API for "user defined" modal choice plugin's. This is a breaking change as old plugin's are not anymore compatible with this version of Nodus.

Nodus 8.1 introduces the possibility to develop Python and R scripts in addition to Groovy scripts. It also runs on Java 16.

Nodus 8.2 runs on Java 17, but now needs Java 11 or above to run. It embeds Groovy 4.0 and runs the embedded HSQLDB, H2 and Derby database engines in server mode, allowing for multiple connections. H2 is also upgraded to version 2. Existing H2 databases need a migration. See https://www.h2database.com/html/migration-to-v2.html.

Table of contents

1 Needed Inputs

1.1 The Project File

The information needed to open a Nodus project must be stored in a text file with the ".nodus" extension. The format of the file is similar to a standard Java properties file. Only two mandatory entries must be present: "network.nodes" and "network.links". Both are a space separated list of layers, each one being a standard Esri shapefile, i.e. a ".shp", a ".shx" and a ".dbf" file.

Note that .dbf files are standard dBase files. Numerous software tools can be used to browse or edit these files. Among the open source tools, you can use LibreOffice (http://www.libreoffice.org) to edit .dbf files or QGis (http://www.qgis.org) to manipulate the shapefiles, including the content of the associated ".dbf" files.

Example:

```
network.nodes= centroids road_points iww_points rail_points
network.links= road_polylines iww_polylines rail_polylines \
               road_con iww_con rail_con
```

A Nodus compatible network needs a set of nodes (vertices) and links (edges). In order to be useable by Nodus, the .dbf files of each layer must contain some mandatory fields:

For Nodes:

- NUM,N,10,0 : The unique ID of the node
- STYLE,N,2,0 : The rendering style of the node
- TRANSHIP,N,1,0 : The type of operations that can be performed at the node

For Links:

- NUM,N,10,0 : The unique ID of the link
- STYLE,N,2,0 : The rendering style of the link
- ENABLED,N,1,0 : If 0, the link will not be used in the model
- NODE1,N,10,0 : The ID of the node at the origin of the link
- NODE2,N,10,0 : The ID of the node at the end of the link
- MODE,N,2,0 : The transportation mode used in the link
- MEANS,N,2,0 : The # of means (types of vehicles) that can be used on the link
- CAPACITY,N,10,0 : The capacity of the link
- SPEED,N,7,2 : The (free flow) speed on the link

After these mandatory fields, the user can add additional numeric or character fields. It is wise to use UTF8 encoding for the DBF file. The (uppercase) name of all the numeric fields can be used as variables in the cost functions.

Before the project can be used, Nodus checks if the set of layers is suitable for a project. In particular, the following conditions must be met:

- The ID of a node must be unique across all node layers
- The ID of a link must be unique across all link layers
- NODE1 must be an existent ID in one of the node layers
- NODE1 must be an existent ID in one of the node layers
- NODE1 and NODE2 must be different

This ensures that the set of layers can be seen as a graph to which assignment algorithms can be applied.

Nodus imports all the ".dbf" files in an SQL database. By default, HSQLDB is used. Nodus is also shipped with Derby (the "official" Java DBMS) and H2 (by the original author of HSQL). The user can choose his preferred DB in the application preferences). Nodus can also be used with other popular DBMS's, such as MySQL or PostgreSQL. In this case, four additional entries must be added to the ".nodus" file: "jdbc.driver", "jdbc.user", "jdbc.password" and "jdbc.url".

Example:

```
jdbc.driver=org.mariadb.jdbc.Driver
jdbc.user=nodus
jdbc.password=nodus
jdbc.url=jdbc:mysql://localhost/demo
```

In this example, Nodus will connect to a MySQL database (schema) called "demo", using "nodus" as a user name and password. The MariaDB (an open source MySQL fork) JDBC connector is used. Nodus is shipped with a few JDBC drivers, located in the "jdbcDrivers" directory. The content of this directory is added to the Java CLASSPATH. You can try other DBMS/JDBC solutions, but only those provided with Nodus were tested.

Other, non-mandatory, entries can be found in the .nodus file:

- "xxx.prettyName" : The name used in Nodus to describe the xxx layer. If not given, the file name of the layer will be used.
- "import.tables": a space separated list of .dbf tables that will be imported in the database, along with the network layers. This can be the case for origin-destination tables for instance. Note that Nodus proposes some "SQL extensions" that can be used in the SQL console. It is for instance the case of the "importdbf tableName" command that imports a .dbf file.
- "openmap.layers" : the name of a ".openmap" file that contains a list of background layers. Please refer to the OpenMap documentation (http://openmap-java.org) for a description of the content of ".openmap" files.
- Since Nodus 7.3, colors can be associated to transport modes (used in the statistics pie charts). Use a "color.mode.x" ("x" being a mode ID) entry for each mode. The value can be a named color ("red" for instance") or an ARGB hex string.

If the files of an OpenMap layer (a shapefile for instance) is stored in the same directory than the node and link layers files, one can just prepend the name of the files with "./", representing the current directory. Nodus will automatically replace "./" with the full path to these files. This makes it easy to move a project directory to another place without having to modify the content of the ".openmap" file.

1.2 The Origin-Destination Matrices

The transport demand is stored in origin-destination (OD) tables. An OD table must at least contain the following fields:

- GRP,N,2,0 : The group of commodities (for instance an NST-R chapter)
- ORG,N,10,0 : The ID of the origin node (often a centroid)
- DST,N,10,0 : The ID of the destination node (also often a centroid)
- QTY,N,10,0 : The quantity to transport from ORG to DST

ORG and DST must be present in one of the node layers. Moreover, loading and/or unloading operations must be allowed at these nodes.

1.3 The Cost Files

Computing (a set of) routes between an origin and a destination needs weights being associated with each link of the networks. Nodus doesn't assign directly on the real networks but generates "virtual networks" that are further used for assignments. Please refer to the provided documentation/help for further details on this concept.

Five types of cost functions must be defined: loading ("ld"), unloading ("ul"), transit ("tr"), transshipment ("tp") and moving ("mv"). They must also be defined for each mode-means combination. The cost functions are written in text files with the ".costs" extension.

The provided ".costs" files are rather well documented and can be used to understand the basics of the Nodus cost functions. Much more is possible, but goes beyond the scope of the demo project.

It is important to note that if a type of cost function is not defined for a given mode/means, a path cannot be computed for it. It is therefore important to define all the needed cost functions.

Example:

```
# Moving cost function for mode 2, means 1
unitCost = 1.12
mv.2,1 = LENGTH*unitCost
```

Note that, since Nodus 8.0, transit time functions are defined in a similar way, using the '@' separator:

```
# Transit time function for mode 2, means 1
mv@2,1 = LENGTH/SPEED
```

2 The Basics of the Interface

Once a project loaded, the Nodus main window should show something similar to Figure 1.

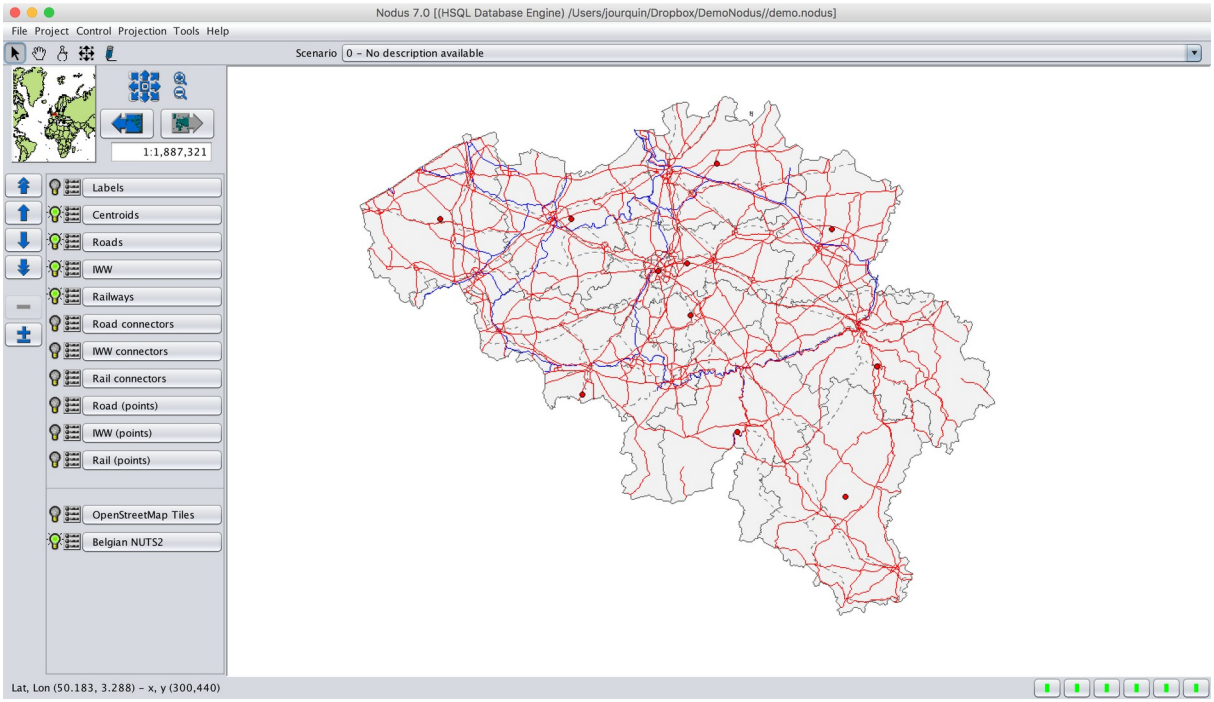

Figure 1: Main window

The largest main zone contains the networks and the background layers. The list of all the layers can be found on the left panel. The green bubble lights can be used to display/hide a layer. The background layers are listed at the bottom. All the layers are named by their "pretty name". The icon between the bubble light and the name of the layer gives access to the parameters of the layer. Finally, the blue arrows can be used to reorder the layers.

Just under the menu bar, four icons give access to specific modes used to interact with the map. The first (the arrow) is used to select/edit nodes and links. Just double-click on an object to open a window that allows editing the data relative to the object (**Figure 2**).

The fields in blue can be directly edited by the user. Those in black can only be modified using the provided controls (choice of a rendering style or allowed operations for instance) or when the network is edited (adding, removing a node or link, attaching a link between two other nodes…).

The second icon (the hand) is used to pan the whole network panel. The compass gives access to a tool that computes the distance between two points and the pen allows the editing of the map (add, move, delete nodes, add, move, delete and split links).

The upper-left corner of the main window displays an overview of the map. To its right, several controls are available to pan, zoom and navigate over the map. The scroll button of the mouse can also be used to zoom in/out.

Finally, just above the map, a combo-box displays the name of the current scenario.

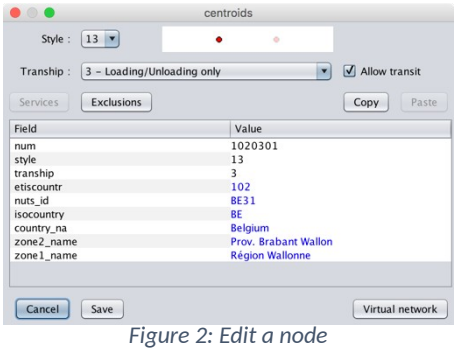

Figure 2: Edit a node

The main window has also a menu bar. The most important entries are:

2.1 Project Menu

This menu gives access to the core functionalities of the software.

2.1.1 Preferences

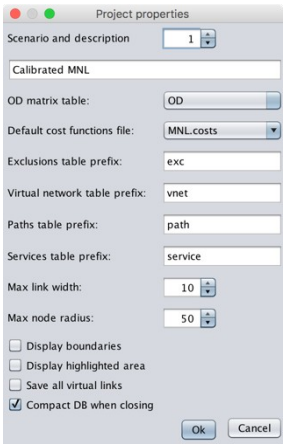

Figure 3: Project settings

This is the place (Figure 3) to choose a scenario, give it a description, associate a cost functions file and an OD matrix. These elements can also be set/modified with the assignment settings.

By default, the database tables created by Nodus have a name starting with the project name, followed by a suffix (such as “_vnet” for virtual network tables for instance). This can be changed here.

“Display boundaries” adds an embedded background layer that roughly represents the world countries.

“Display highlighted area” displays a user defined rectangle that limits the assignment to the network it contains. This is useful for large networks that are only partially used in a project. If the highlighted area is enabled, the generation of the virtual network will only take the nodes and link within the rectangle into account, which improves the performance (duration) of the assignment.
If the HSQLDB or H2 database is used, one can also choose to compact the database when closing the project. These DBMS’s data files can become rather large because empty space is not (always) reclaimed by these two DBMS’s for performance reasons. If this is checked, a “shutdown compact” is run on the database just before closing. See the documentation of these DBMS’s for more information.

2.1.2 Edit Cost Files

Opens a text editor with the current cost file.

2.1.3 Edit Services

Used to define lines and services. This feature is not covered by this document.

2.1.4 Assignment

Opens a window dedicated to the several available assignment algorithms. Please refer to the help/documentation for more info about these algorithms. Figure 4 shows the different controls the user has over multi-flow assignments.

Figure 4: Multiflow assignment

2.1.4.1 General Controls

These controls are present in all assignment tabs.

The user can choose the scenario ID to associate to the assignment. If the word “scenario” has a green background, the ID is still free. An orange background means that a scenario with this ID already exists and that a new assignment will overwrite the existing tables.

The number of iterations represents the number of alternative routes that will be computed for each mode of transport. For the equilibrium assignments, this value represents the maximum number of iterations to perform towards the equilibrium solution.

If the OD matrix contains more than one group of commodities, Nodus can assign them simultaneously (in parallel). The maximum parallel threads can be set here. Nodus limits the proposed values to the number of CPU cores the computer it runs on has. Parallel assignments drastically reduce assignment duration.

Beside the project preferences, this is also a place where the description of the scenario, the used OD table and the cost functions file can be set. The assignment can also be limited to a subset of the OD matrix defined by an SQL select statement.

If a highlighted area is defined and enabled, the user can choose to limit the assignment to this area by checking the relevant box.

If “log lost paths” is checked, the list of origin-destination pairs between which no route could be computed will be displayed in the console. This is useful for debugging: in most cases, “lost paths” are found if there is somewhere a missing link between O and D or if not all the necessary cost functions were defined.

Nodus offers the possibility to save the details of each computed path beside the general flows on the network. If “Save paths” and “Detailed paths” are checked, it will be possible to visualize the routes and flows computed for every OD pair. For large projects, this generates huge database tables. If only “Save paths” is checked, the list of individual links of each route is not saved, but only the basic information for each path (cost, length, …) is kept.

2.1.4.2 Specific Controls

These controls are specific for multi-flow assignments only.

Two shortest path algorithms are proposed. See the pro and cons of each in the help/documentation. In most cases, the “fast” method gives good results.

Three modal split methods are proposed:

- The proportional method spreads the demand over the different routes (modes) according to (the inverse) of their respective costs.

- The multinomial logit method is broadly used. In this version, the utility of each alternative route (mode) is defined as the (inverse) of the total cost of the alternative. The utility function is thus an univariate (the total cost) function without intercept.
- The Abraham method is less used, and mostly discussed in the French literature. In this version, it is based on the same utility function as the multinomial logit model.

The user can also develop its own modal split methods that will appear in the list. The way to develop such methods goes beyond the scope of this document although an example is provided in the "MLogit" subdirectory of this demo.

For models that include intermodal transport possibilities, the user can choose to retain intermodal solutions only if they are cheaper than any other solution.

If the assignment has more than one iteration, i.e. if more than one route must be computed for each mode, a "Cost Mark-Up" can be set. This is the value by which the cost of each link along the last computed route must be multiplied before the next iteration. More can be found on this in the help/documentation.

Finally, the value set for "Max detour" is used to limit the set of retained routes to all those that are shorter to the shortest route (all modes included) multiplied by this value. For example, if this value is set to 2, only the routes shorter than twice the length of the shortest route will we retained in the set of valid alternative routes. If the value is set to 0, all the alternatives are kept, regardless of their length.

2.1.5 Display Results

This opens a window in which the user can choose to display the OD matrix (node) or the assignment (link) of a scenario. If the "detailed paths" were kept during the assignment, one can also choose to display the route(s) computed for a given OD pair and group of commodities. For this, the user has to manually replace the "?" signs (see Figure 5) present in the proposed SQL statement by the IDs of the origin and destination nodes and the group of commodities.

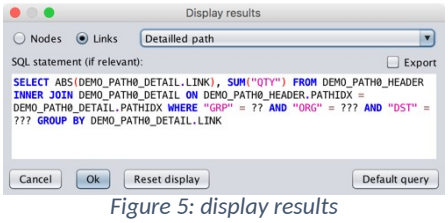

Figure 5: display results

Semi-dynamic assignments can also be displayed, but this is not discussed in this document.

2.1.6 Scenarios

From this window (Figure 6), it is possible to delete or rename (change the ID) of a scenario. This affects all the tables relative to the scenario. It is also possible to compare two scenarios.

In this case, a resulting scenario is created, and the assigned flows are compared, i.e., for each link, the flow assigned in “scenario 2” is subtracted from the flow assigned in “scenario 1”. The result can thus be positive or negative. When the resulting scenario is displayed via “Display results”, the negative flows are rendered with a “dimmed” color.

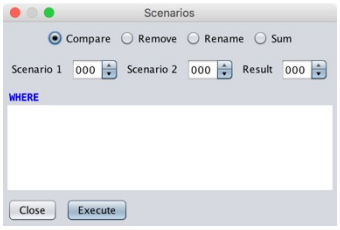

Figure 6: Scenarios

2.1.7 SQL Console

The SQL console (Figure 7) lists all the tables of the project. The user can write SQL statements in the upper right zone. The result of the query is displayed in the bottom-right zone.

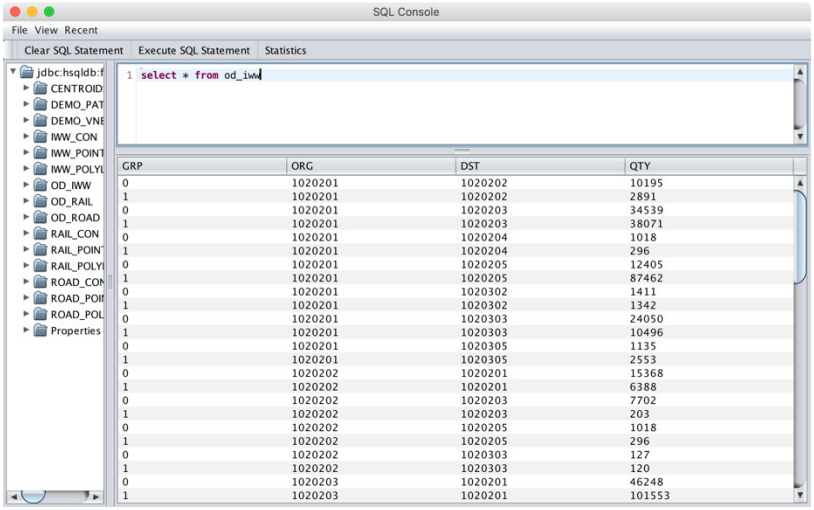

Figure 7: SQL console

The console is also able to handle SQL scripts, i.e. several SQL statements, each one being terminated by a semi-column.

Several Nodus specific extensions to standard SQL are proposed:

- **CLEARSCENARIO num**: Deletes all the tables related to the given scenario id.
- **CLRSCR** : Clears the output console.
- **DISABLEECHO** : If used, the SQL commands will not be echoed in the result area.
- **DISABLEHEADERS** : If used, the headers of SQL outputs will not be displayed.
- **DISPLAYGRID** : Forces the next SQL output to be displayed in grid format.
- **DISPLAYTEXT** : Forces the next SQL output to be displayed in text format.
- **ENABLEECHO** : If used, the SQL commands will be echoed in the result area (default).
- **ENABLEHEADERS** : If used, the headers of SQL outputs will be displayed (default).

- **EXPORTCSV tableName** : Exports a table in CSV format.
- **EXPORTCSVH tableName** : Exports a table in CSV format, the first line containing the field names (header line).
- **EXPORTDBF tableName** : Exports a table in DBF format.
- **EXPORTXLS tableName** : Exports a table in XLS format.
- **EXPORTXLSX tableName** : Exports a table in XLSX format.
- **EXTRACTSHP FROM shapefile1 TO shapefile2 WHERE sqlCondition**: Creates a new shapefile from the output of the SQL where statement performed on a shapefile.
- **IMPORTCSV tableName** : Imports a CSV file. The related empty table must exist in the database.
- **IMPORTCSVH tableName** : Imports a CSV file, ignoring the first (header) line. The related empty table must exist in the database.
- **IMPORTDBF tableName** : Imports a DBF file.
- **IMPORTXLS tableName** : Imports a XLS file. The related empty table must exist in the database unless the first row of the sheet contains the format of each column, following the DBF standard.
- **IMPORTXLSX tableName** : Imports a XLSX file. The related empty table must exist in the database unless the first row of the sheet contains the format of each column, following the DBF standard.
- **STOP** : Convenient command that stops a script. Useful for debugging.

These additional keywords are recognized by syntax highlighting.

It is also possible to use user defined variables, regardless the DBMS that is used. Such a variable must use the @@ prefix. Example:

```
@@od := od_iww;
select count(*) from @@od;
select sum(qty) as total from @@od;
```

If a “.dbf” file of a shapefile is modified by an SQL statement, it must be exported (“exportdbf”) to make the changes applicable to the loaded map. It must be clear that the SQL statements affect the table in the database, not in the “.dbf” files of the project. The changes must thus be saved (exported) to keep the content of the shapefiles synchronized with the content of the database.

The user must also be aware that direct manipulation of tables related to shapefiles can be dangerous and can potentially break the consistency of the networks. For instance, if the ID of a node is modified, the links that refer to this node must also be updated. If it is not the case, the integrity test that Nodus performs when a project is open will fail.

2.2 Control Menu

Offers several possibilities to show/hide some components and change the background color.

2.3 Projection Menu

By default, the Mercator projection is used, but other projections are available.

2.4 Tools Menu

2.4.1 Look & Feel

By default, Nodus uses the native Look&Feel of the system it runs on. Another can be chosen.

2.4.2 Language

The interface is available in English and French.

2.4.3 Console

This opens a “Console” window that intercepts all the warning and error messages that Nodus writes to the standard output. Useful for debugging.

2.4.4 Groovy Scripts

This opens an editor for script editing. Scripts must be written in the Groovy language (http://groovy-lang.org), which is close to Java. Scripts can use the Nodus API in order to add new or project specific functionalities to Nodus. According to what is selected in the main preferences, a Nodus editor is used or the native Groovy console is launched. Both allow running scripts.

2.4.5 Resources Monitor

Displays the memory and CPU usage of Nodus. For very large projects, more “heap” memory can be assigned to the software. Therefore, the “-Xmx” parameter found in “jvmargs.sh” or “jvmargs.bat” (created in the install directory after a first run of Nodus) can be set to a higher value. Please refer to the Java JVM documentation for further details.

It is highly recommended to use a 64 bits JVM, as the 32 bits version is limited to a heap of about 1.5 Go, which can be a limit for large projects.

3 The “demo” Project

The demo project is intended to understand the basics of Nodus. It is based on real, but partial, data fetched from the European Etisplus (http://www.etisplus.eu) project. The Belgian networks and data for the OD matrices are fetched from the project, and the values of the cost functions are a simplified version of functions that were used in the framework of ECCONET (https://www.ecconet.eu), another European research project. The OD matrices are defined at the NUTS2 level, which is much too coarse to set up a useable network model for Belgium. However, all these inputs make it possible to illustrate some basics of Nodus.

3.1 Load the project

The project is configured in the “demo.nodus” file. It has 5 node layers, 6 link layers and it imports three .dbf tables (origin destination matrices for road, inland waterways (IWW) and rail transport). The matrices contain data for two groups of commodities). OpenMap background layers are configured in the “demo.openmap” file.

Figure 8 illustrates the project when it is loaded for the first time. During this first loading, a HSQLDB database is created in the project’s directory and all the relevant ".dbf » files are imported.

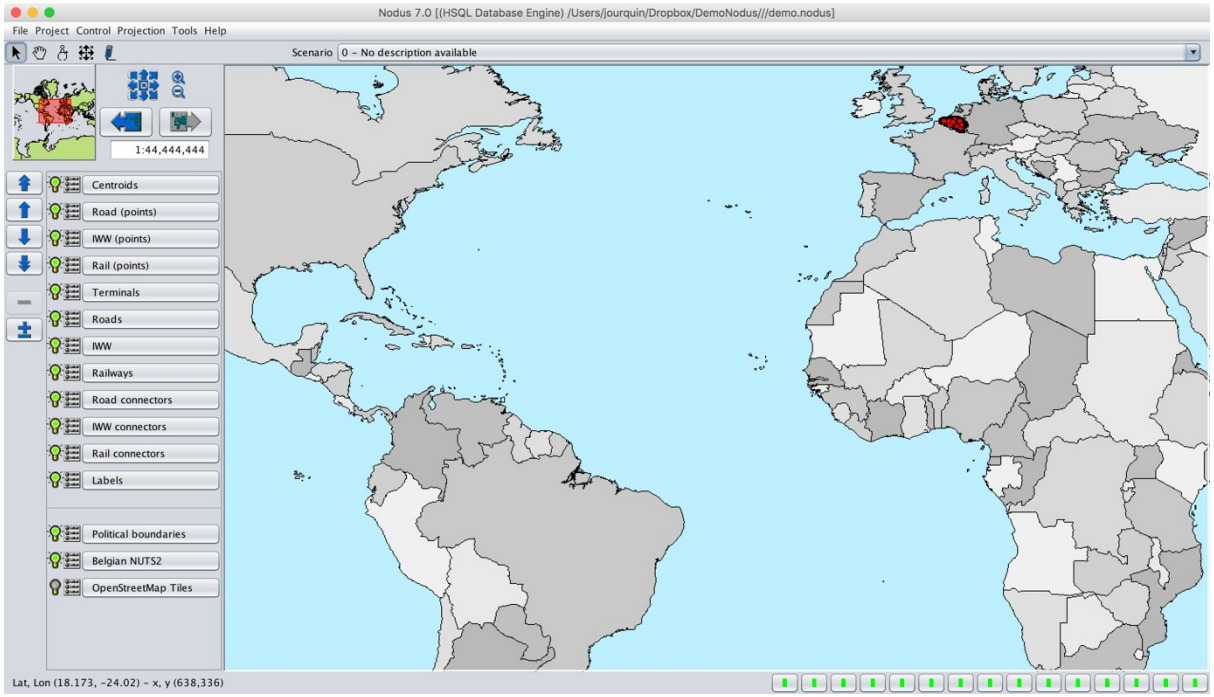

Figure 8: The "demo" project when first loaded

Before going further, it is useful to briefly describe each node and link layer of the project.

3.1.1 Node layers

- Centroids: these nodes represent the center of gravity of a region. They are the places where freight can be loaded and unloaded, and they are connected to the modal networks by connectors. Figure 9 shows the settings for a centroid: loading and unloading operations are allowed and transit operations are not permitted (to avoid traffic through centroids having more than one connector to the same modal network).
- Terminals: these nodes will be used in intermodal models. They are also connected to road and rail networks by connectors, and only transshipment operations are allowed. No transit is allowed.
- Road, IWW and Rail points: these are just nodes corresponding to the end points of all the links of the network. No operations beside simple transit is allowed.

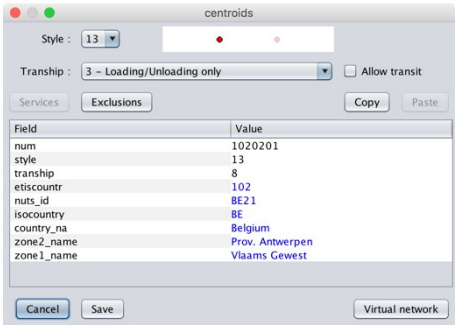

Figure 9: Data and settings for a centroid

3.1.2 Link layers

- Road, IWW, and Rail networks: represent the physical topology of the networks for these three modes.
- Road, IWW and Rail connectors: these links don't correspond to real network chunks, and are created to connect the centroids and the terminals to the modal networks.

The networks for this demo project are located in Belgium. In Figure 10, a zoom on the networks is illustrated, using a Mercator projection. The layers are also reordered: the "labels" layer comes first (but is not yet configured), followed by the centroids, the terminals and the three modal networks. The connectors and the point layers come afterwards and are not enabled (displayed). The background layer displays the Belgian NUTS 2 regions. In Figure 11, the NUTS 2 background layer is replaced by an OpenStreetMap image fetched from a web server. Finally, Figure 12 hides the railways and inland waterways (IWW) networks and displays the name of the centroids. In order to display labels, the icon just on the left of the "Labels" layer must be clicked, and "Query database for labels" for the "Centroids" layer must be chosen. Just select the field that must be displayed as a label.

Before creating models, it is a good idea to test the available options in the GUI. It is also interesting to have a look at the SQL Console (Figure 13), where the list of imported tables is shown: all the ".dbf" tables of the layers are present, along with the three OD matrices.

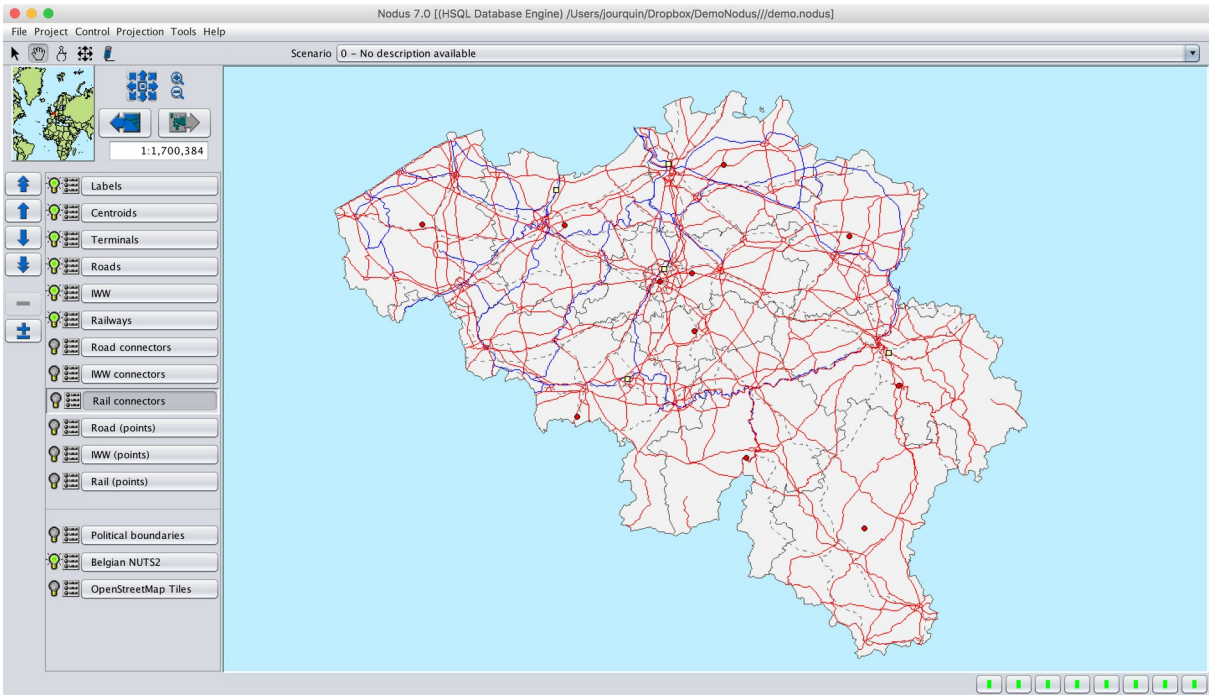

Figure 10: Zoom over Belgium with reordered layers

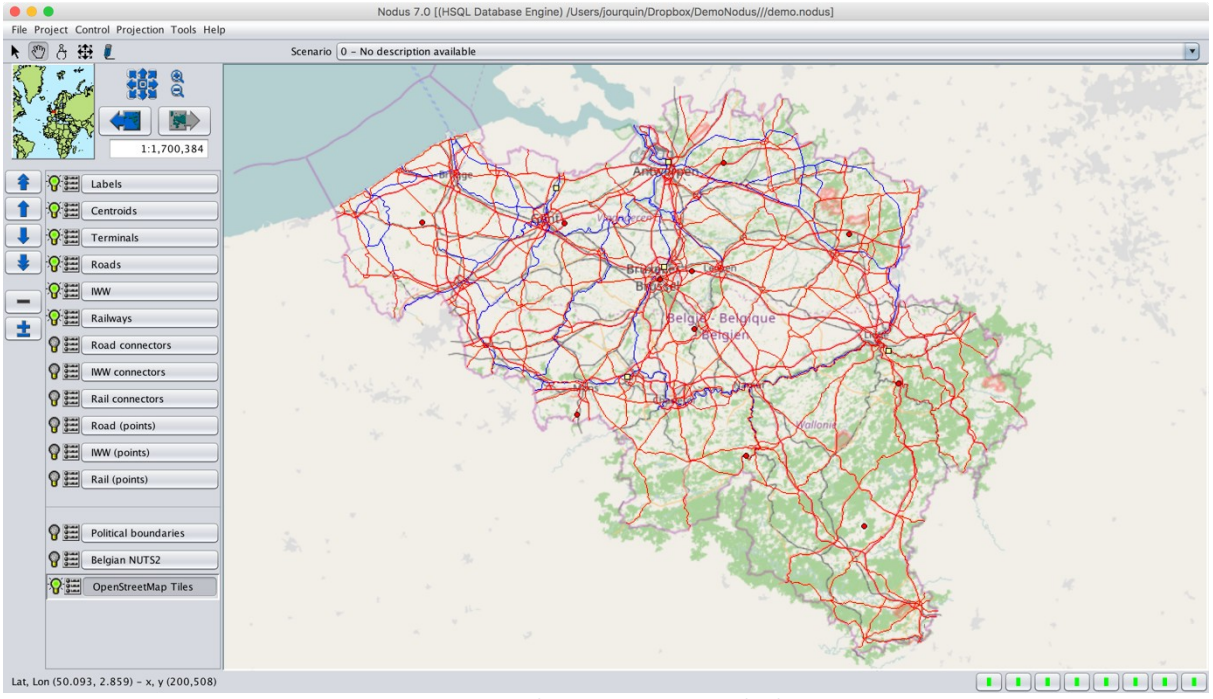

Figure 11: Display OpenStreetMap tiles layer

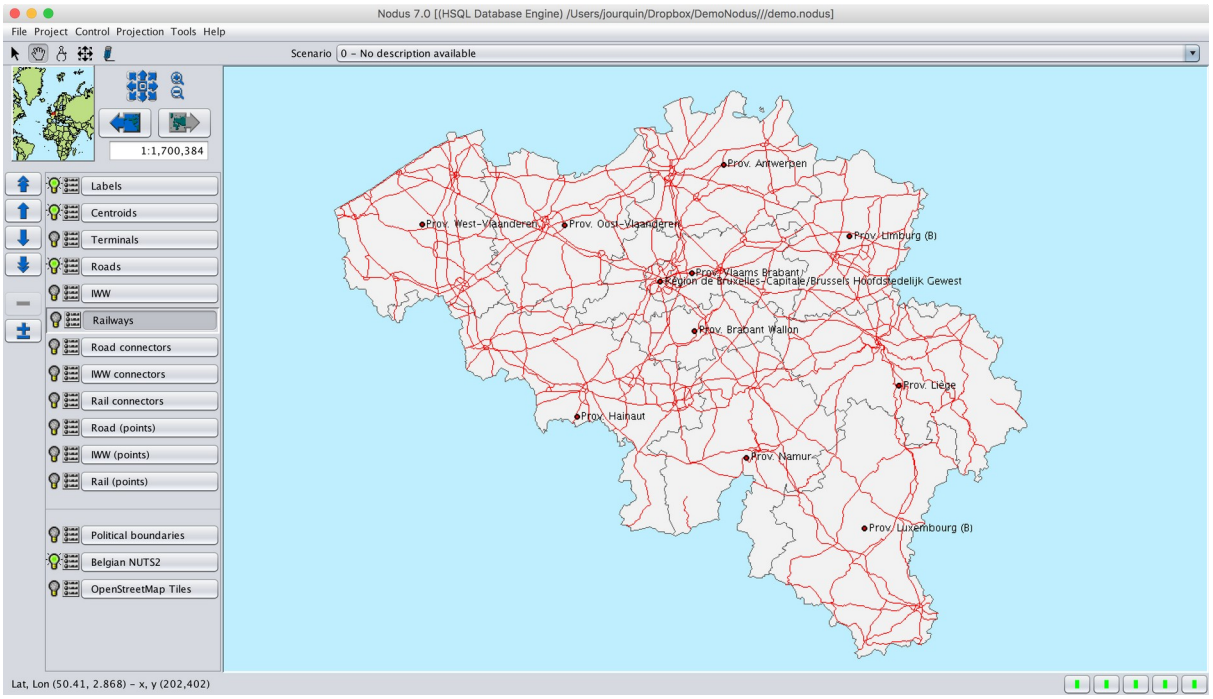

Figure 12: Display the names of the centroid

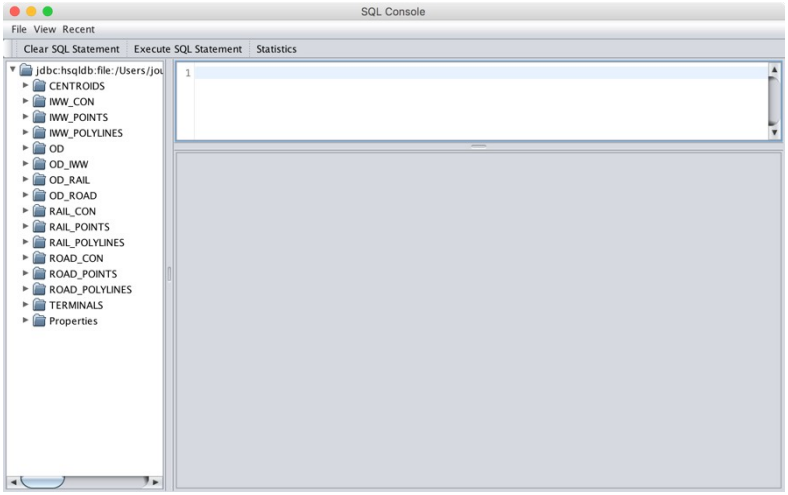

Figure 13: initial tables

3.2 Scenario 1: Road assignment

This first scenario shows how to run a simple assignment for a single mode. In Figure 14, the different settings for this assignment are illustrated:

- Set scenario ID to 1.
- Use of a fast multi-flow assignment. Note that, for this scenario, the result will be equivalent to a simple all-or-nothing assignment, because only one route for a single mode is computed for each OD pair.
- The “road.costs” cost functions are used. This is a very simple cost functions set: (un)loading and transit costs are set to 0 and the fastest routes are computed (moving costs = LENGTH/SPEED). Only one means (one type of vehicles) is defined.

- The OD matrix for road transport is used
- The details of each path (route) are stored in the database
- A short description is given to the scenario (“Road only”)

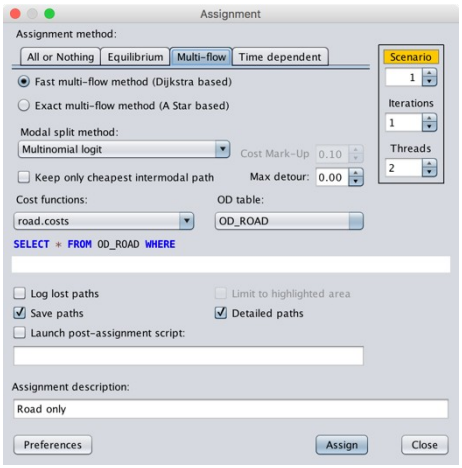

Figure 14: Road assignment

The assignment creates three additional tables in the database:

- demo_vnet1: The virtual network.
- demo_path1_header: The details of the computed paths (without the list of used links).
- demo_path1_detail: All the used links for each path.

The second table is generated because “Save paths” is checked, the last one because “Detailed paths” is also checked.

Figure 15 shows the resulting assignment. To obtain this, go to “Project|Display results” and select “Assigned quantities” in the list of available possibilities for links. Keep the proposed SQL query unchanged.

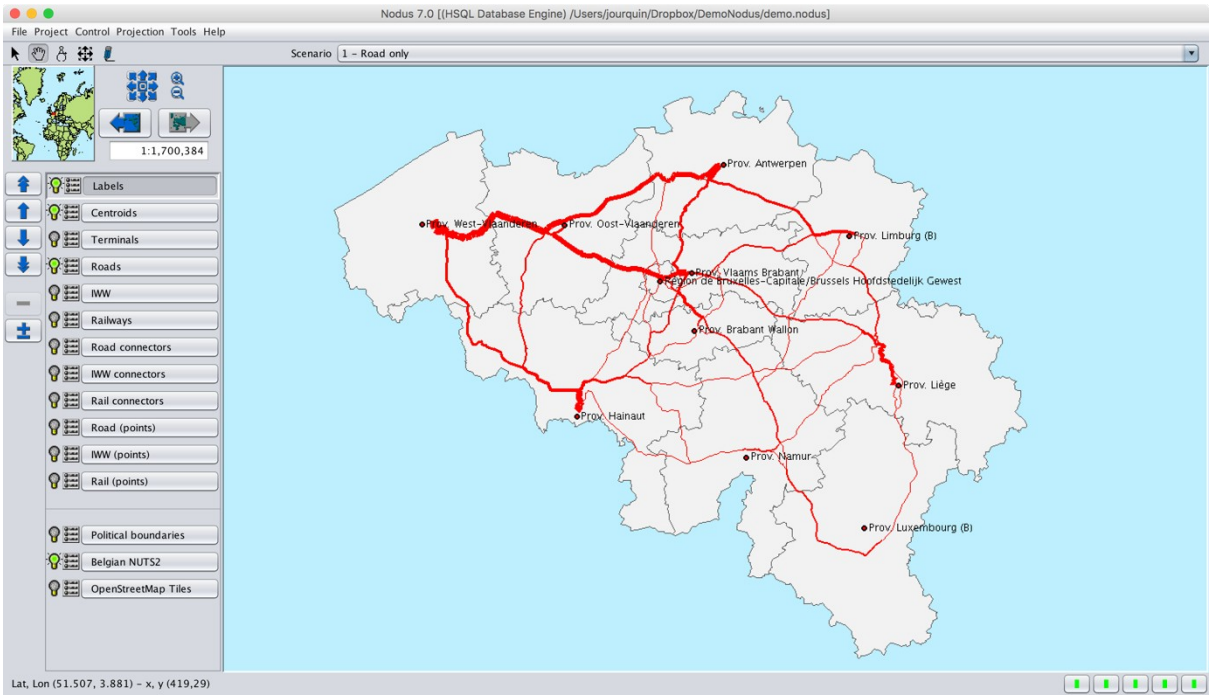

Figure 15: Assignment of scenario 1

Because the details of all the computed paths are saved in the database, it is possible to visualize the route computed for a given OD pair and group of commodities. Therefore, go to "Project | Display results" and select "Detailed path" among the proposed possibilities for links.

The IDs of the origin and destination nodes can be obtained by a double-click on the centroids that correspond to O and D. For instance, "Prov. Antwerpen" and "Prov. Hainaut": their "num" field contains respectively 1020201 and 1020302. In the proposed SQL query, replace the "??" signs: 0 for GRP, and the just retrieved IDs for ORG and DST, as in Figure 16. The result can be seen in Figure 17.

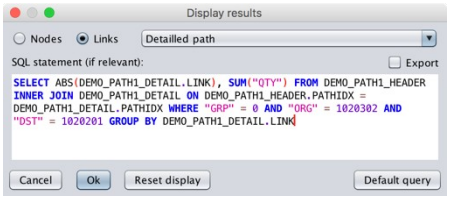

Figure 16: Display detailed path settings

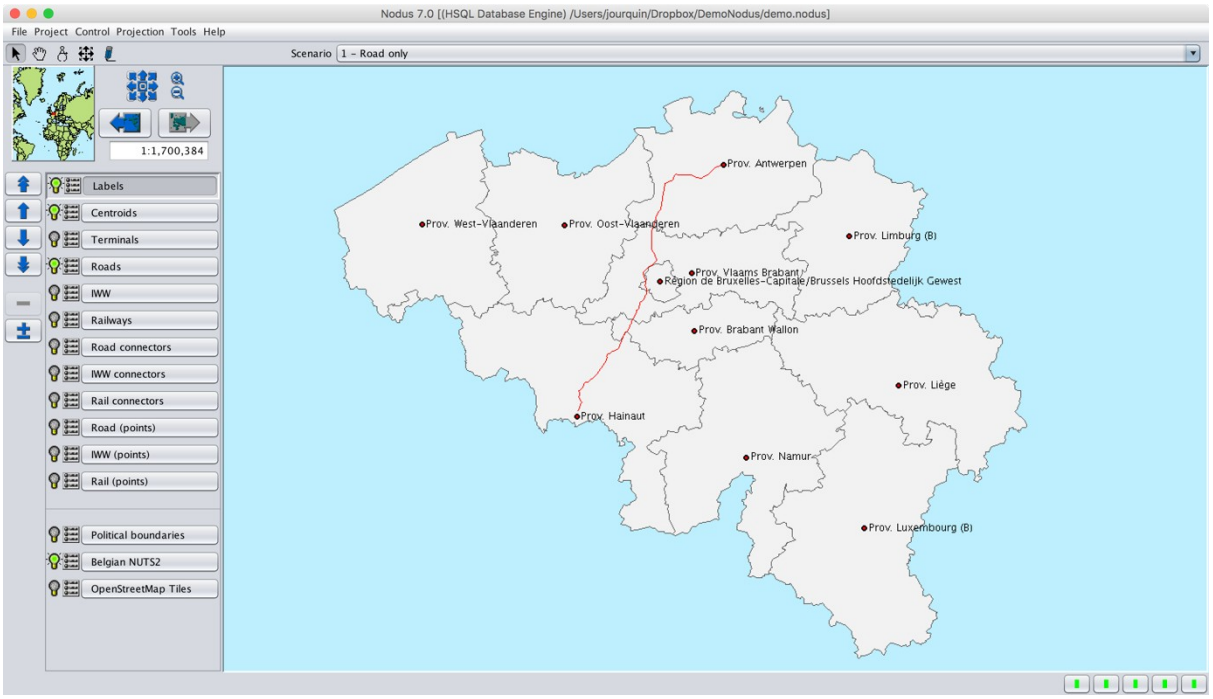

Figure 17: Route between "Antwerpen" and "Hainaut"

3.3 Scenario 2: Multi-route road assignment

Using the same basic data (cost functions and OD matrix), it is possible to obtain an assignment during which several routes are computed between every OD pair for the same transportation mode.

In Figure 18, three iterations (alternative routes) are performed. Between each iteration, a cost markup of 50% is applied to each already used link. Please refer to the help/documentation for more information on the way multi-flow assignments are implemented in Nodus. The resulting flows can be observed in Figure 19, in which the traffic is much more spread over the network than in Figure 15.

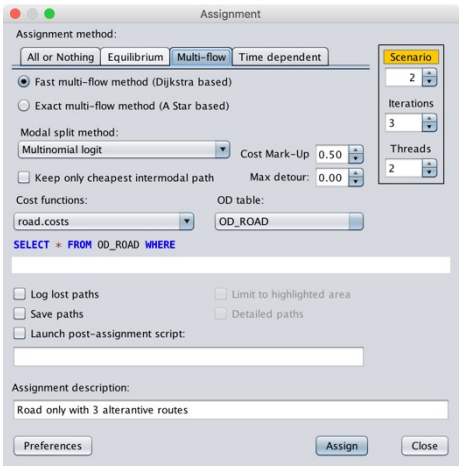

Figure 18: Multi-flow for road transport

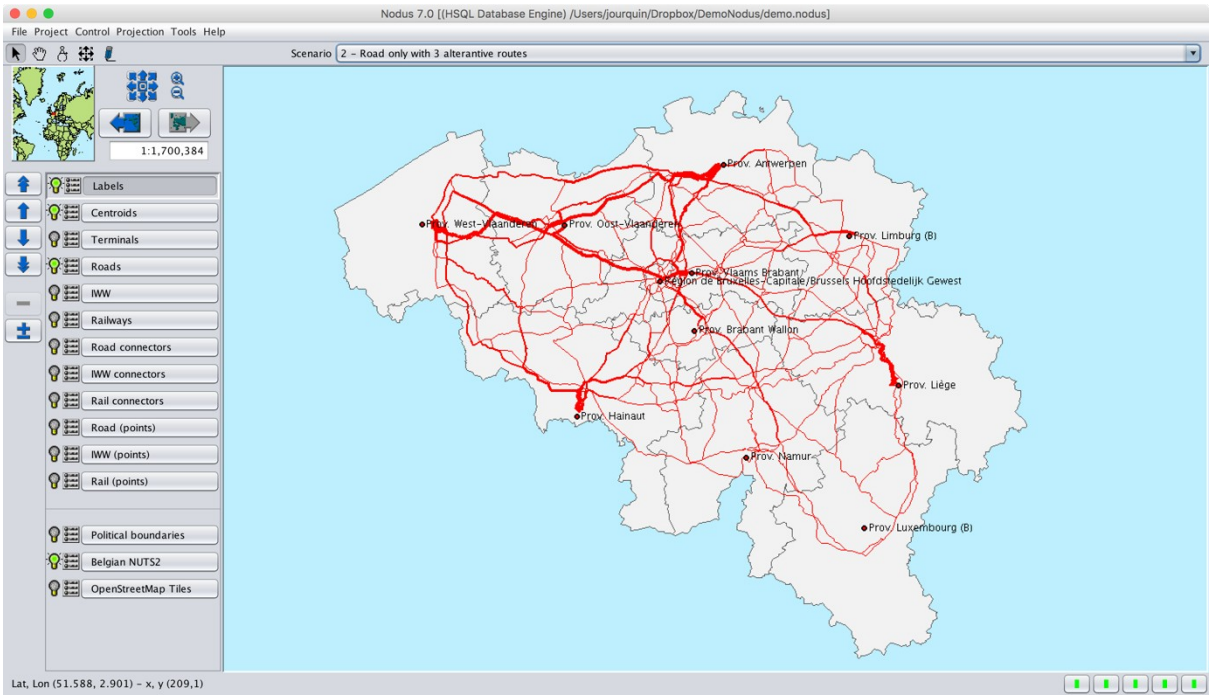

Figure 19: Multi-flow assignment for road transport

3.4 Scenario 3: Inland waterways, with two types of barges

In this scenario, only one mode (IWW) is assigned again, but two means (small and large barges) are defined. The split between the two means is performed by means of a "proportional" model. Beside a change in the modal-split method and the used cost functions, the settings (Figure 20) are similar to those used for scenario 2.

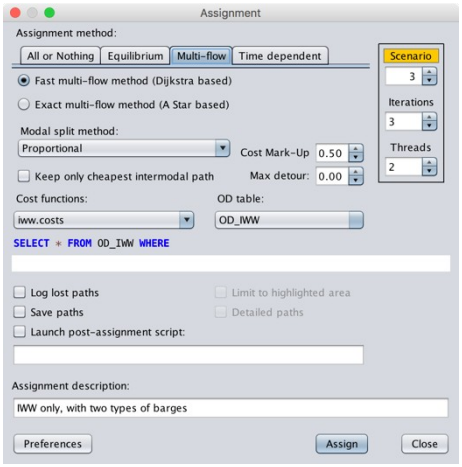

Figure 20: Settings for the IWW assignment

Note that the IWW network has a "CLASS" field which corresponds to the gage of the rivers/canals.

Knowing that the "MEANS" field specifies the "largest" vehicle that can be used on a link, the "MEANS" field is set to the value of "CLASS" divided 10. Thus, small barges (means = 2) can

use all the canals/rivers with of class 20+, and large barges (means= 4) can only use canals with class 40+.

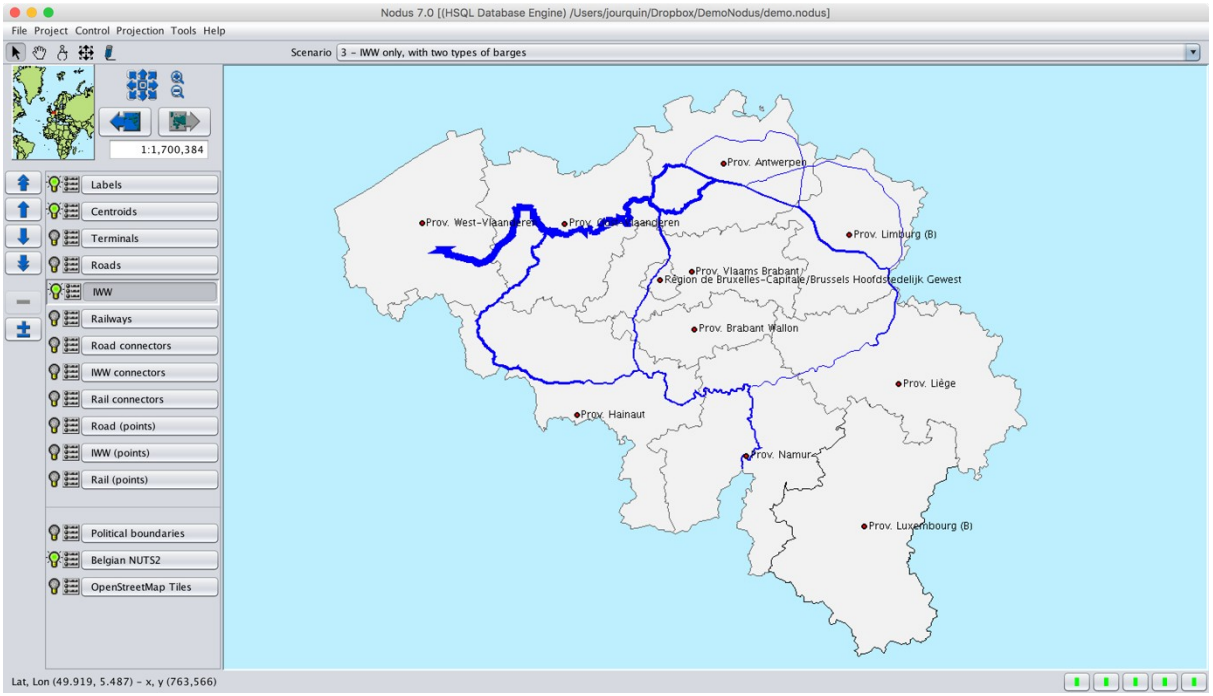

Figure 21: Assignment on the IWW network

From the SQL Console, it is also possible to obtain some statistics (Figure 22). For instance, the loaded tons per mode-means can be displayed (Figure 23). One could have expected the modal share for small barges being exactly 1/3, because a "proportional" modal-split method was used and that the use of small barges costs the double than large barges per transported ton. The share for small barges is, however, slightly higher. This is because, on some routes, large barges cannot be used because some rivers/canals have a class smaller than 40.

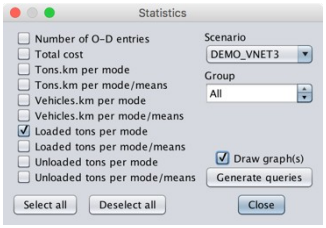

Figure 22: Available statistics

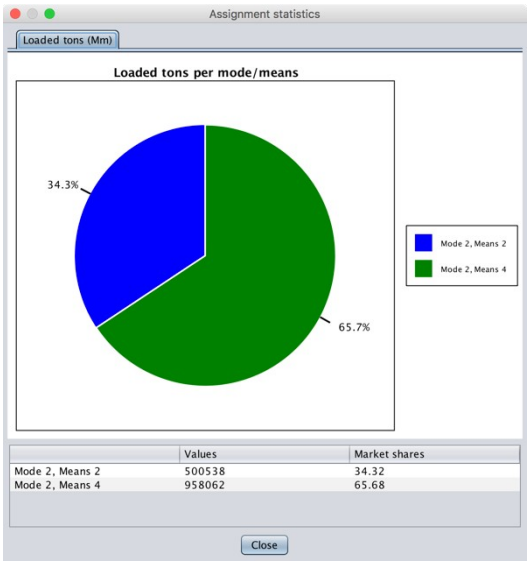

Figure 23: Statistics in a pie chart

3.5 Scenario 4: Rail transport

In this scenario, somewhat more complex cost functions are used, with figures taken from a real project. This illustrates how user defined variables can be used and how different values can be defined for each group of commodities.

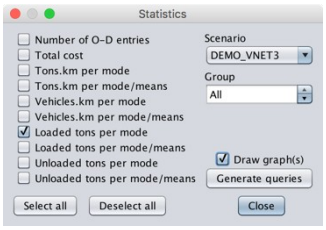

Figure 24: Setting for the rail assignment

The result is shown in Figure 26, which also displays the used origin-destination matrix. Therefore, once the flows displayed, go back to “Project|Display results”, and select “O-D matrix” from the choices provided when “Nodes” is checked (Figure 25), leaving the SQL query unchanged.

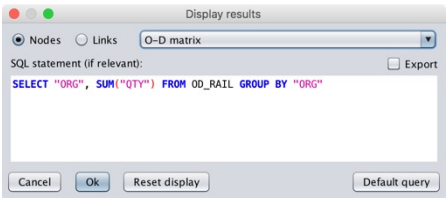

Figure 25: Display an O-D matrix

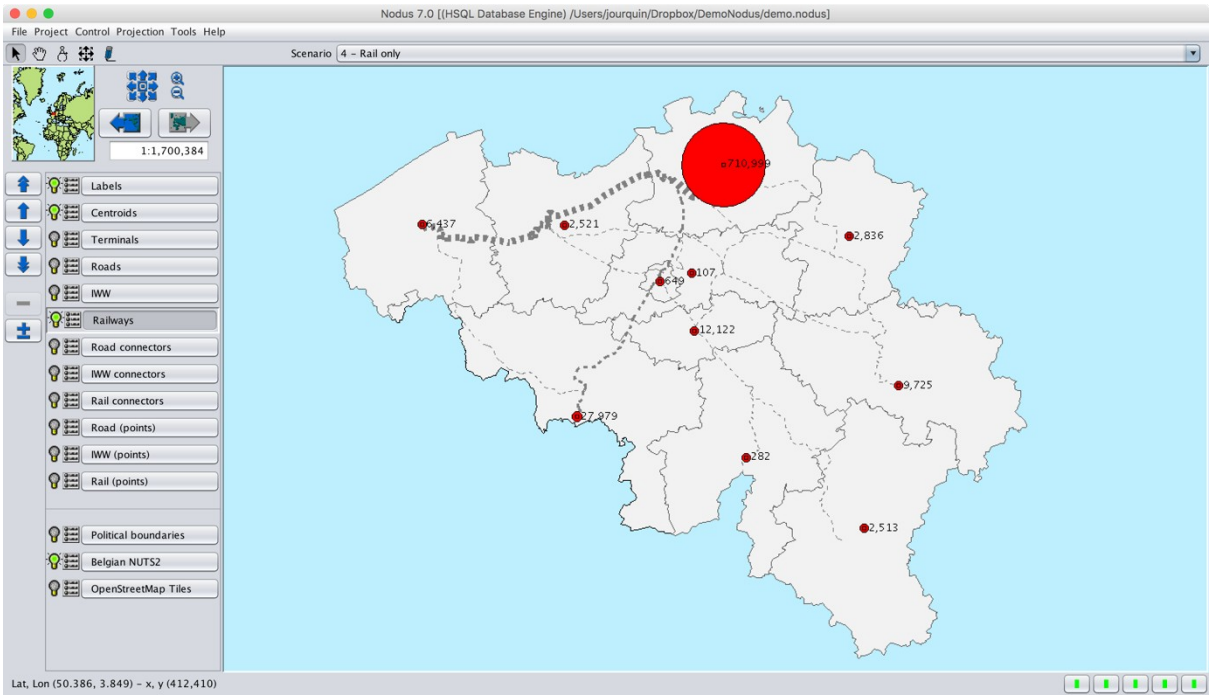

Figure 26: Rail assignment and O-D matrix

3.6 Scenario 5: Multimodal assignment

It is now time to go for multimodal assignments. In such an assignment, the demand between each origin and destination is spread over all the available modes, using a modal-split method.

The first thing to do is to merge the three modal matrixes into a single multimodal matrix. This can be achieved running the "MergeODs.sql" script from within the SQL Console (Figure 27). This creates the "OD" table, that can be used in multimodal assignment.

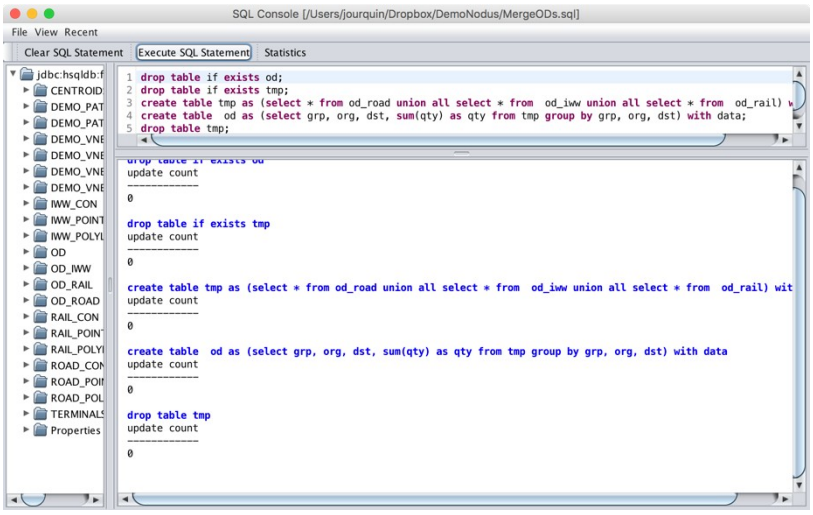

Figure 27: The "MergeODs.sq"l script in the SQL Console

The "uncalibrated.costs" file gives an example on how to define costs functions for several transport modes. It uses cost values grabbed from the same project as the one used for

scenario 4 (railways only). Figure 28 shows the settings used for this assignment, which are very similar to what was used in the previous scenarios: the most important differences are in the cost functions and the OD matrix. The resulting assignment (Figure 29) looks, however, very strange. Indeed, it seems that most of the freight is transported by barges. This is confirmed by the statistics presented in Figure 30, while the figures of Table 1 can be retrieved from the modal OD matrices. The model must thus be calibrated.

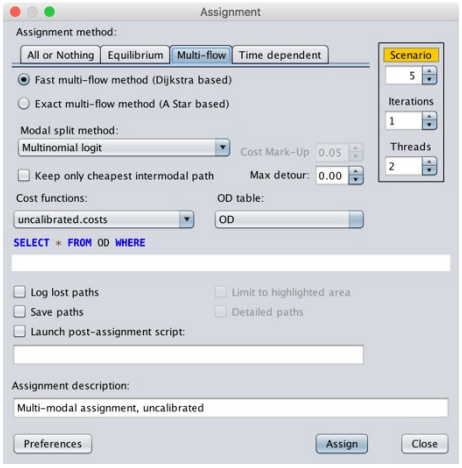

Figure 28: Settings for a multimodal assignment

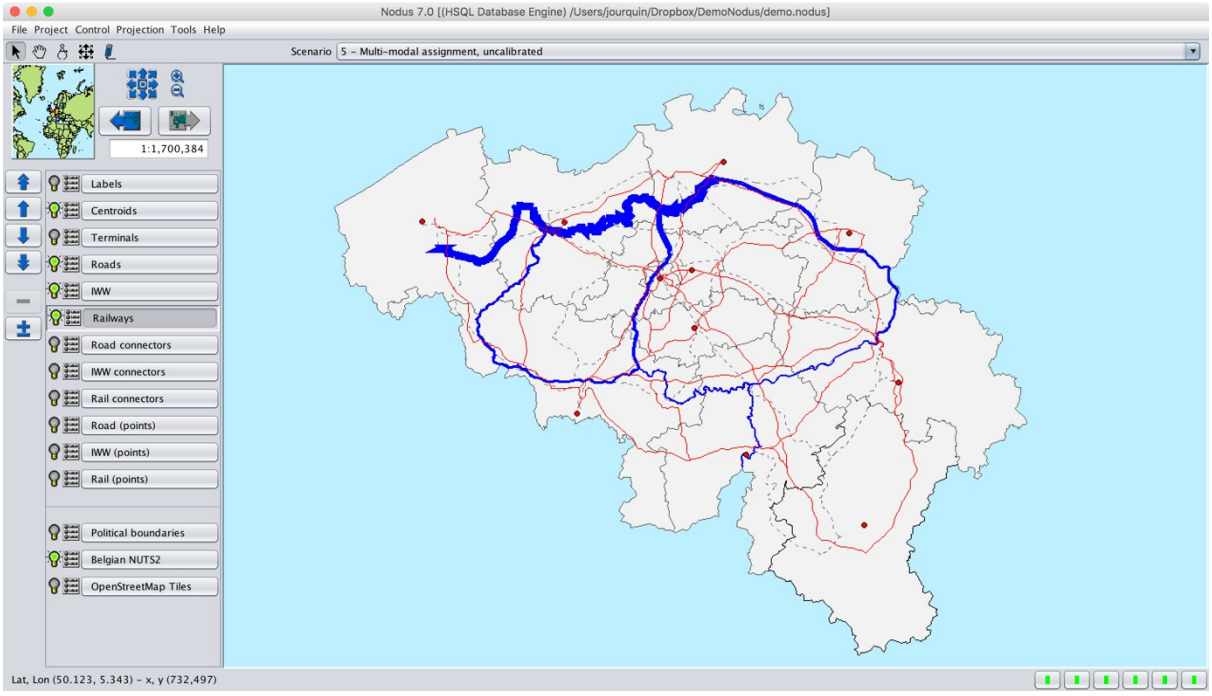

Figure 29: Uncalibrated multimodal assignment

		Tons per group		Share per group	
Mode		**0**	**1**	**0**	**1**
1	**Road**	7438402	17204671	90,1%	92,4%
2	**IWW**	728281	730319	8,8%	3,9%
3	**Rail**	91148	685022	1,1%	3,7%

Table 1: Reference modal shares

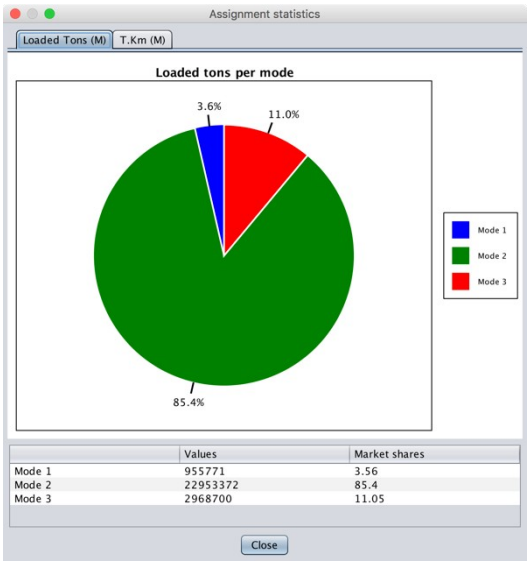

Figure 30: Modal shares

3.7 Scenario 6: Simple calibration of a multimodal assignment

Calibration involves estimating the values of various constants and parameters in the model structure. Even if certainly not the most solid way to achieve this, a simple technique is to find a series of multiplicative factors that can be applied to the cost functions in order to obtain a modal split close to the observed one. This is illustrated in ”MNL.costs”, in which a variable called “calibration” is defined for modes 2 and 3 (mode 1 is kept unchanged) and for the two groups of commodities.

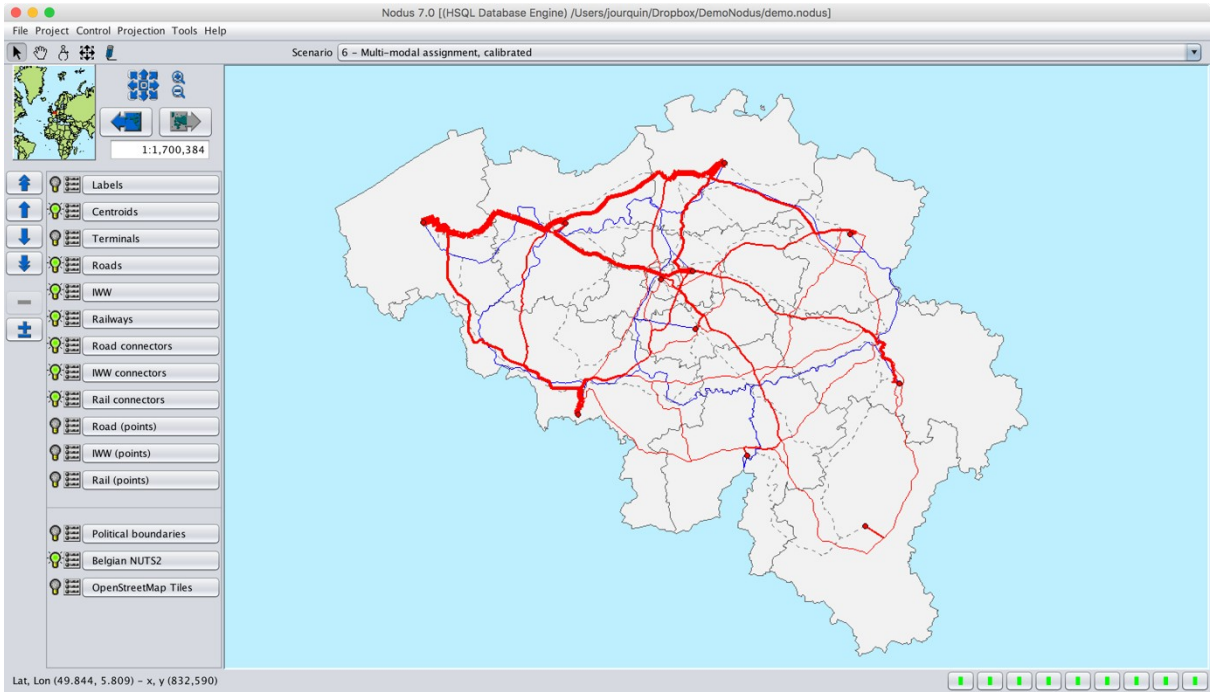

Figure 31: Multimodal assignment - simple calibration

With a manual iterative procedure, it is rather straightforward to find a combination of values that result in a correct modal split after assignment. Indeed, if a market share for a mode is

too large, one has to increase the value of the relevant “calibration” variable in order to increase the cost of the transport mode and to reduce its attractiveness. Decrease the value of the variable if the share must be larger. The values proposed in “MNL.costs” result in market shares of respectively 90.02%, 8.86% and 1.12% for group 0 and 92.41%, 2.93% and 3.66% for group 1, which is close to the reference figures found in Table 1. The resulting assignment is shown in Figure 31: road transport is clearly dominant.

3.8 Scenario 7: Calibration with “external” estimators

The calibration procedure presented in the previous scenario tries to obtain global market shares for the different modes that are as close as possible to the observed ones. More sophisticated methods can be used, based on econometric methods, such as the maximum likelihood. It goes beyond the scope of this document to explain how this can be done.

In this example, the utility functions used in the multinomial logit model has the following functional form:

$$-n_i \ln\left(TC_i\right)+\delta_i$$

where TC is the total cost of transport for the alternative (mode) and *i* the mode. We have seen that Nodus can provide this data (TC) for each mode, route and group of commodities if “Save path” is checked in the assignment parameters. This data, for an uncalibrated assignment, can further be used along with the reference OD matrices in some econometric software package in order to estimate the η_i and δ_i estimators. In this case, the “mlogit” R package was used, and the estimated values for a conditionnal logit model can be found in “MLogit-R.costs”. These values are used by a user-defined modal-split method that uses the Nodus API. This method is embedded in the provided “MLogit.jar” file, which is a Nodus plugin, loaded with the project. This method appears along with the “proportional”, “Abraham” and “Multinomial logit” methods in the list of available methods (Figure 32). The R script and the Java code corresponding to the plugin are provided in the “MLogit” subdirectory, along with an explanation on how to obtain the estimators.

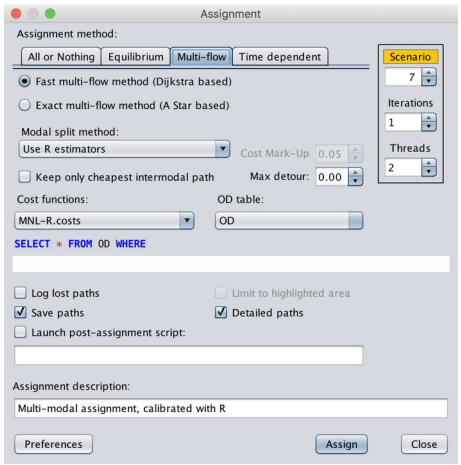

Figure 32: Use of a user defined modal split method

With this model, the estimated modal shares are respectively 90.07%, 8.82% and 1.10% for group 0 and 92.40%, 3.92% and 3.68% for group 1, and the flows are illustrated in Figure 33. At a first glance, these results are very similar to those obtained in scenario 6.

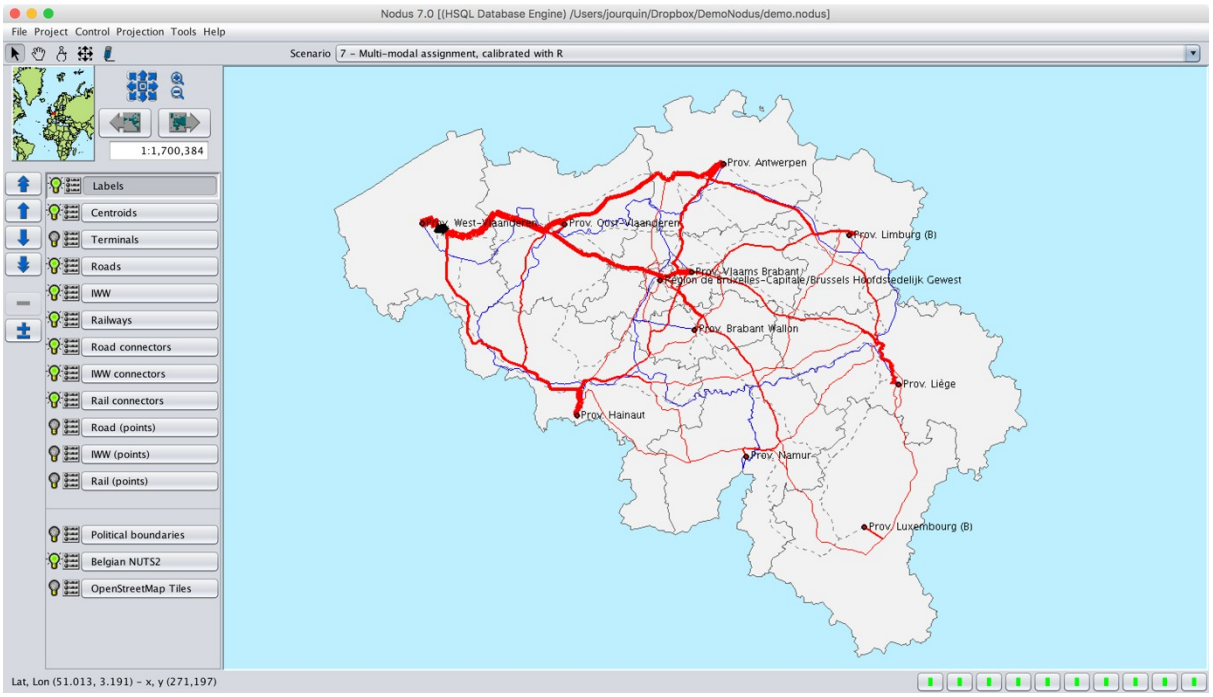

Figure 33: Assignment with a more complex modal choice model

3.9 Validation of a model

While calibration involves estimating the values of various constants and parameters in the model structure, validation is the application of the calibrated models and the comparison of the results against observed data. At the aggregated level, one can for instance compare the global observed and estimated market shares. As outlined for scenarios 6 and 7, those market shares are close or even identical to the observation, and both models perform equally well.

A correct validation should have a closer look at disaggregated data. At OD level for instance, one could compute the R squared (determination coefficient) between the quantities assigned to each mode and the observations. Again, the data at the path level can be used ("demo_path?_header", where "?" is the scenario ID). Here also, many software packages can be used to compute this indicator. It is also possible to write a Groovy or SQL script to do this in Nodus. The "ODCorrelation.sql" script provided with this demo is an example. It works perfectly well and is simple to understand, but it is not very efficient and could take a long time if used with large databases. The script contains two variables that must be set by the user before a run: the name of the path header table and the ID of the mode for which the R squared must be computed. The results are given in Table 2, and it becomes very clear that the calibration method used in scenario 7 gives much better results than the simple method applied for scenario 6. Obviously, more sophisticated utility functions can be imagined, but this goes beyond the scope of this tutorial.

Mode	Scenario 6	Scenario 7
1	0.948	0.955
2	0.432	0.527
3	0.009	0.220

Table 2: R^2 at the OD level

3.10 Scenario 8: Comparison between scenarios 6 and 7

The difference between the assignments of scenarios 6 and 7 can also be visualized using the "Project | Scenarios" tool. The settings proposed in Figure 34 will generate a new scenario 8 that compares the assignments of scenarios 6 and 7. This is illustrated by Figure 35. In all the previous maps, red was used for road transport, blue for IWW and dashed gray lines for railway transport. In this figure, some links are also "dimmed", corresponding to links for which the flow in scenario 7 is higher than in scenario 6. In such a case, the resulting flow is negative, and this is visualized by a "dimmed" color. This is very clear in the zoomed view provided by Figure 36. The figures displayed along the road network represent the difference between the flows of both scenarios. These figures can be displayed if the "Labels" layer is enabled and "Road polylines" is checked in the "Labels" settings.

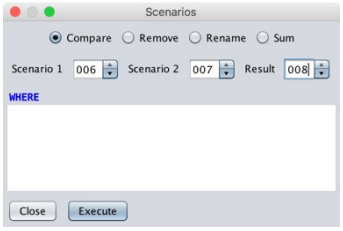

Figure 34: The "scenarios" tool

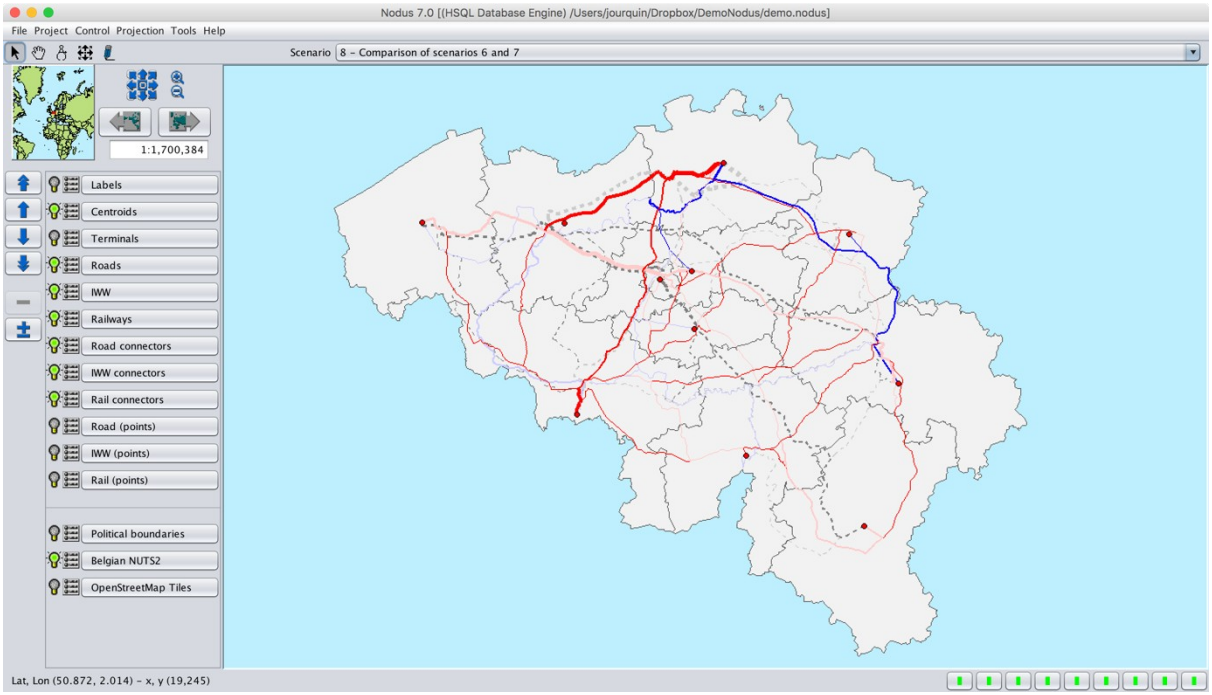

Figure 35: Visualization of the comparison

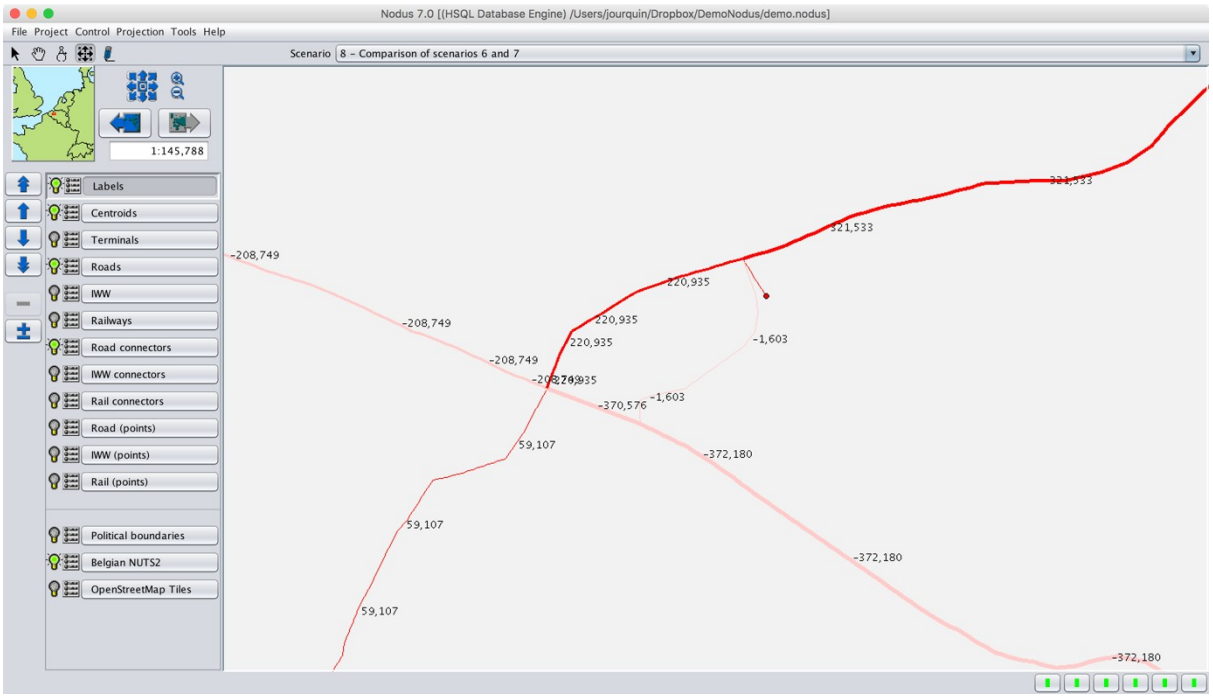

Figure 36: Zoom with displayed flows

3.11 Scenario 9: Intermodal model

Up to now, intermodal, i.e. a route along which at least two different transportation modes are used, was not modeled. Intermodal transport needs some transshipment points (terminals) being located in the network. Transshipment cost functions must also be defined. These cost functions must exist for each possible mode/means combination. In this demo, rail-road intermodal transport will be illustrated. Transshipment costs must be defined for the road to rail and the rail to road operations. If one considers that a transshipment costs 1.5€ per ton (this is not a real value), the cost functions can be defined as:

```
# From road to rail
tp.1,1-3,1 = 1.5
# From rail to road
tp.1,3-1,1 = 1.5
```

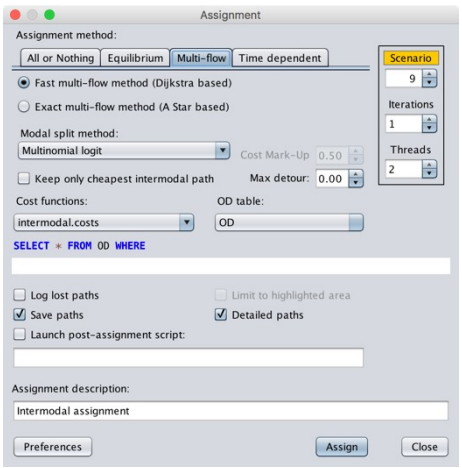

Figure 37: Settings for the intermodal assignment

The assignment settings are as illustrated in Figure 37. In the resulting tables, one can identify intermodal paths with the following SQL query:

```
select grp,org,dst,nbtrans from demo_path9_header where nbtrans > 0
```

The route between centroids 1020100 and 1020303 is an example, illustrated by Figure 38, in which the Brussels container terminal is used.

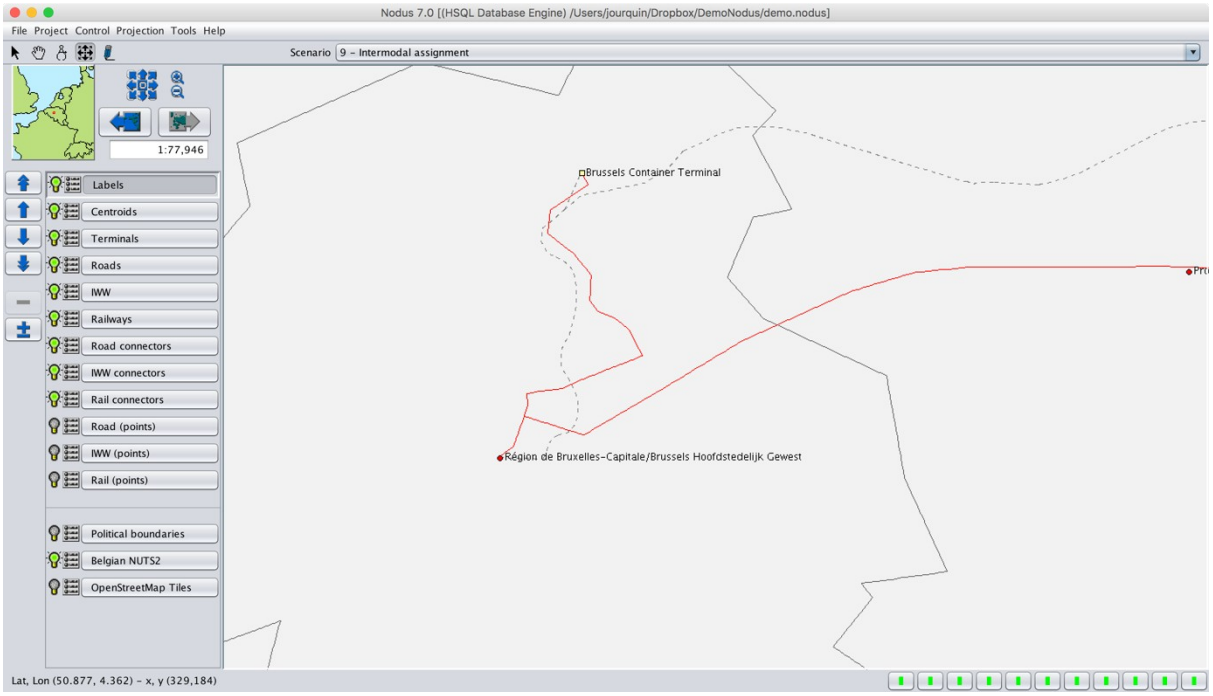

Figure 38: Intermodal route using the Brussels terminal

Note that, in this scenario, all the intermodal routes use only one terminal (the above presented SQL query returns only record with "nbtrans" = 1). Modeling more realistic intermodal transport (use a truck from a centroid to a terminal, then a train to another terminal, and finally a truck again from the second terminal to the final centroid) is somewhat more complex to model.

3.12 Scenario 10: More complex intermodal model

A good understanding of the "Virtual Networks" method is needed to correctly model intermodal transport. An elegant solution is to make use of additional "means":

- 3 "means" can be defined for road transport:
  - Means 1: regular trucking for an OD relation (already defined)
  - Means 2: pre-haulage to a terminal
  - Means 3: post-haulage from a terminal to the final destination
- 2 "means" can be defined for rail transport:
  - Means 1: regular trains for an OD relation (already defined)
  - Means 2: trains used for intermodal transport between two terminals

All the cost functions must not be defined for the different means:

- For regular truck and rail transport, loading, transit, moving and unloading cost functions must exist, similarly to what was used in all the previous scenarios.
- Only loading, transit and moving costs must be defined for pre-haulage by truck.
- Only transit, moving and unloading costs must be defined for post-haulage by truck.
- Only transit and moving costs must be defined for intermodal trains
- The transshipment costs must be defined from pre-haulage trucks to intermodal trains and from intermodal trains to post-haulage trucks.

Once all these costs are defined, the assignment algorithm from O to D can identify the following alternatives:

- A truck only route, using mode 1, means 1 from O to D
- A rail only route, using mode 3, means 1 from O to D
- An intermodal route starting with mode 1, means 2 (the only other loading possibility), followed by a transshipment to mode 3, means 2 at a first terminal, a transshipment to mode1, means 3 at the second terminal and a final unloading at D.

As no loading and unloading cost functions are defined for intermodal trains (mode 3, means 2), they cannot be chosen at O or D. They can only be loaded and unloaded at transshipment facilities. Two additional means are needed for trucking because, otherwise, the means could be used for loading and unloading, making no difference with regular trucking.

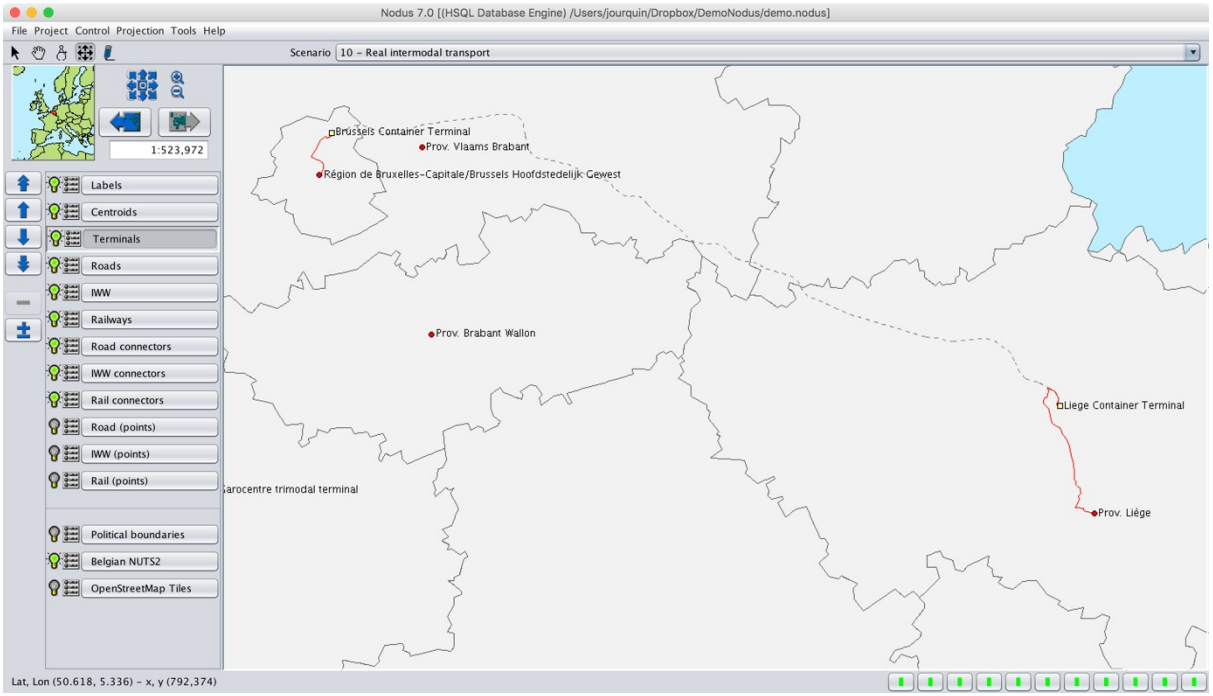

Figure 39: Intermodal route with two terminals

Once all these cost functions defined (see “intermodal2.costs”), one must also be sure that the all these additional means will exist in the virtual network. Indeed, the road and rail

networks that were used up to now have only one means. The following SQL statements can be run from the SQL console:

```
# Set the means on the main networks and the connectors
update road_polylines set means = 3;
update road_con set means = 3;
update rail_polylines set means = 2;
update rail_con set means = 2;
# Export the tables in the ESRI shapefiles
exportdbf road_polylines;
exportdbf road_con;
exportdbf rail_polylines;
exportdbf rail_con;
```

The assignment settings are similar to those of the previous scenario. The only difference is the cost functions file that is used. In the resulting scenario, the route from 1020100 to 1020303 makes use of two terminals, as illustrated in Figure 39.

4 Conclusion

This document is not a complete user manual of Nodus. The software offers much more, such as semi-dynamic assignments, the possibility to model lines and services[1] or the development of scripts written in the Groovy language and plugins. It presents, however, a series of 10 scenarios showing the basics of multi- and intermodal transport modeling.

1 This functionality is temporarily disabled in Nodus 8.0

5 Appendix – More on cost functions

Nodus offers many more possibilities with cost functions. This appendix gives more details about the available options.

5.1 Modes, means, groups, classes and scenarios

The core (and mandatory) cost functions are loading (“ld”), unloading (“ul”), transit (“tr”) and moving (“mv”) costs. For intermodal transport, transshipment costs (“tp”) must also be defined. The basic usage of these functions is described in the different scenarios illustrated in the demo.

The mandatory cost functions must be defined for each mode-means combination. The user has to decide which numerical ID he assigns to each mode (from 0 to 99). The same is true for the means (type of vehicle) available for each mode. For example, if only one mode is defined (road) and only one means (general truck), the cost functions file must at least define values for these functions:

```
ld.1,1 =
ul.1,1 =
tr.1,1 =
mv.1,1 =
```

A transshipment cost function, if used, must specify the “from” mode-means and the “to” mode-means:

```
tp.1,1-2,1 =
```

Variables (and cost functions) can be defined for several “groups” (categories of commodities for instance). The group must be defined in the OD matrix table in the “grp” field. For instance, the user-defined variable “truckLoadingCost” can be defined for groups 1 and 2 with different values:

```
truckLoadingCost = 0
truckLoadingCost.1 = 10
truckLoadingCost.2 = 12
```

~~Note that, if group specific values are defined, the “generic” variable (i.e. without the appended group ID must also be defined (even with a value = 0).~~ **This is not anymore needed since Nodus 8.0.**

If the following cost function is defined

```
ld.1,1 =  truckLoadingCost
```

The value of ld.1,1 will be equal to 10 when “group” 1 is assigned, and 12 when “group” 2 is assigned.

In addition to the mandatory “grp” field in the OD matrixes, an optional “class” field can also be defined in the OD tables. This can be useful if one needs to use different cost values for different distance classes (short, medium and long hauls for instance). Specific class related values can then be defined for variables or functions using the following syntax:

```
truckLoadingCost.1 = 10       # Group specific value
truckLoadingCost-1 = 12       # Class specific value
truckLoadingCost.1-1 = 13     # Group and class specific value
```

Finally, one can also define scenario specific variables and cost functions instead of having a separate cost functions file for each scenario. The scenario ID must be used as a prefix:

```
1.truckLoadingCost.1 = 10     # Scenario and group specific value
1.truckLoadingCost-1 = 12     # Scenario and class specific value
1.truckLoadingCost.1-1 = 13   # Scenario, group and class specific value
```

Since Nodus 8.0, transit time (duration) functions can also be defined in the cost function file. These functions follow exactly the same structure as the regular cost functions. A transit time function uses a ‘@’ instead of a dot after the type of movement. Example: “mv.2.1=” is a moving cost function while “mv@2.1” is a moving transit time function. The durations are stored in the “path_header” tables when an assignment is performed with “save paths” enabled (saving the detailed paths is not needed). The stored values can further be used in modal choice model estimations.

Note that, unlike what is needed for cost function, duration function must not be defined for all the types of movements.

5.2 Embedded variables

Nodus also has some “reserved words” for some variables (variables are case sensitive). The most important are:

- AVGLOAD : The average payload for a type of vehicle. It is used to compute the number of vehicles needed to transport a given quantity. Can be defined for each mode-means, group and scenario. “AVGLOAG.1,1.0 = 20” means that the average load of a vehicle of mode 1, means 1 is 20 tons when it transports goods of group 0.
- PCU : The Personal Cars Units ratio. If a truck represents 1.5 PCU, that means that it takes the place of 1.5 cars. This is used in equilibrium assignments when the capacity of the links is expressed (as usual) in PCU’s/hour. Can be defined for each mode-means, group and scenario. “PCU.1,1 = 1.5” means that a vehicle of mode 1, means 1 represents 1.5 cars on a road.

- ~~*LD_DURATION, UL_DURATION, TP_DURATION : Loading, unloading and transhipment durations, expressed in hours. Can be defined for each mode-means combination, group and scenario. "LD_DURATION.1,2.0 = 4" means that one needs 4 hours to load commodities of group 0 on a vehicle of mode 1, means 2. These values, converted in seconds, are added to the travel duration field stored in the path header tables. In this ways, the duration can represent the total transit time (and not only the travel time).*~~
  This is deprecated since Nodus 8.0, as transit time functions are now defined following the same structure as cost functions (see sections 1.3 and 5.1).

Some variables can be directly used in the cost functions:

- LENGTH : the length of a link (in km). Can be used in "mv" cost functions.
- VOLUME (or FLOW for Nodus versions prior to 8.0) : This variable is available for "mv" costs during equilibrium assignments. It represents the flow assigned to the link. If the AVGLOAD and PCU variables are correctly defined, the flow is expressed in Personal Car Units. An example of cost functions suitable for equilibrium assignments is given in a next section.

The numerical fields of the ".dbf" files can also be used as variables. For example, the mandatory "SPEED" or "CAPACITY" fields that must be present in the "link" layers can be used as variables.

5.3 Cost functions for equilibrium assignments

Equilibrium assignments take capacity constraints into account and spread the traffic over several routes, trying to minimize the total cost of transport. Several algorithms can be found in the literature, among which the most cited are the "incremental" method, the "Method of Successive Averages" (MSA) or the "Frank-Wolfe" algorithm. They are all implemented in Nodus.

Note, however, that the equilibrium assignment algorithms were developed in the context of urban areas (and primarily for passenger transport). Indeed, these algorithms work well if one have to examine the traffic at moments where there is a lack of capacity. This is typically the case during peak hours. To be efficient, one needs OD matrixes containing transport during the peak hours.

The same algorithms are much less useful if they are applied to large areas and/or when the demand (OD matrixes) contains yearly figures, which is often the case for strategic models.

A more in depth discussion on these algorithms can be found in Jourquin B. and Limbourg S., *Equilibrium Traffic Assignment on Large Virtual Networks, Implementation issues and limits for Multi-Modal Freight Transport*, European Journal of Transport and Infrastructure Research, Vol 6, n°3, pp. 205-228, 2006 (Available in the Nodus help system).

However, here is an example cost function file that can be used with an equilibrium algorithm (MSA is the most efficient), in which the moving cost function is based on the well-known Bureau of Public Roads speed-flow relation:

```
# Example of a MSA equilibrium assignment
# Using the Bureau of Public Roads speed flow relation

# T : travel time
# T0 : free flow travel time
# alpha & beta : calibration factors
# X : volume
# C : capacity
#
# T = T0 * (1 + beta * (X/C)^alpha)

# Calibration factors
alpha = 4
beta = 1

# The VOLUME variable in Nodus can be expressed in vehicles if
# the AVGLOAD (average load) variable is defined.

# Average load per truck
AVGLOAD.1,1 = 20

# Nodus also can handle the capacity usage of a vehicle
# with the PCU (Passenger Car Units) variable.
# One truck = 1.5 PCU
PCU.1,1 = 1.5

# Load.Mode,Means
ld.1,1 = 0

# Unload.Mode,Means
ul.1,1 = 0

# Transit.Mode,Means
tr.1,1 = 0

# Move.Mode,Means (cost function that depends on the volume)
mv.1,1 = (LENGTH/SPEED) * (1 + beta *(VOLUME/CAPACITY)^alpha)
```

In order to run an assignment, chose the “Equilibrium” tab in the “assignment” dialog, the select one of the algorithms (MSA for instance). You also have to select the maximum number of iterations to perform before stopping the process. Nodus will stop before reaching this maximum if the equilibrium solution is found before.

Since Nodus 8.0, two well-known predefined volume-delay functions are recognized : Bureau of Public Roads (BPR) and the conical function proposed by Spiess (CONICAL). They can be written as :

- BPR(VOLUME, CAPACITY, alpha, beta)
- CONICAL(VOLUME, CAPACITY, alpha)

The last lines of the previous example can thus be written as:

```
# Move.Mode,Means (cost function that depends on the volume)
mv.1,1 = (LENGTH/SPEED) * BPR(VOLUME, CAPACITY, alpha, beta)
```

The “equilibrium.costs” and “od_equilibrium.dbf” (OD matrix) are provided in the “demo” directory for testing purposes.

5.4 Other capacity constraints

Other forms of capacity constraints that are not immediately related to congestion but are linked to the characteristics of the infrastructure are often encountered and must be properly handled in transport models. It is for instance typically the case of the locks that can be found on inland waterways segments. There are several ways to handle this.

If the segments of your network don’t identify each lock as a separate node, you can add a field (“NBLOCKS” for instance) to the links layer(s) “.dbf” file that contain the number of locks in the segment. You can then edit the segments with locks and update the number of locks it contains. A simple cost function file could then be something like this:

```
# Average time lost per lock (in seconds)
AvgLockTime = 1800

# Load.Mode,Means
ld.2,1 = 0

# Unload.Mode,Means
ul.2,1 = 0

# Transit.Mode,Means
tr.2,1 = 0

# Move.Mode,Means
mv.2,1 = SECONDS(LENGTH, SPEED) + NBLOCKS * AvgLockTime
```

Since Nodus 8.0, the “HOURS”, “MINUTES” and “SECONDS” functions are defined, which compute the travel time along a network link with a given LENGTH and SPEED.

This can obviously be refined. For instance:

- Adding a second field in the “.dbf” file with the average time lost per lock in the segment (instead of a global value).
- Replacing the “NBLOCKS” field with another one, representing the total average time lost in the segment.
- ...

If each lock is represented by a node, one could for instance add a “LOCKTIME” field to the nodes layers(s), and use the content of this field in the transit cost functions (instead of the moving cost function as in the previous example):

```
# Load.Mode,Means
ld.2,1 = 0

# Unload.Mode,Means
ul.2,1 = 0

# Transit.Mode,Means
tr.2,1 = LOCKTIME

# Move.Mode,Means
mv.2,1 = SECONDS(LENGTH,SPEED)
```