GridTool - User Manual

R. Gaugl[a,*], S. Wogrin[a], U. Bachhiesl[a], L. Frauenlob[a]

[a] *Institute of Electricity Economics and Energy Innovation (IEE), Graz University of Technology, Graz, Austria*

Abstract

This is the user manual of the GridTool available on GitHub[1]. GridTool is an easy-to-use tool to convert and simplify electricity grid data from the OpenStreetMap-project for the use in power simulation models. The main advantages of the tool are its flexibility, customizability and its heuristic approach for identifying grid nodes. Instead of the lengthy and predominantly manual process of converting and implementing electricity grid data from transmission and distribution system operators, the GridTool shortens that task to seconds. During this process the OpenStreetMap data gets reduced to the essential things that are needed in energy simulation models for load flow purposes, with optional plots that help to tweak the most important settings. To our knowledge there is no other open-source tool, that works out-of-the-box that does this task and is as easy-to-use, only needs one program to run (MATLAB) and uses a heuristic approach.

This manual includes an overview of the program, a guide for downloading and installing the GridTool, a detailed description of the modules and functions of the GridTool, a guide for how to run the GridTool and how to download the necessary OpenStreetMap data with OverpassTurbo, an overview of the options and warnings, an explanation of the default behaviour of the GridTool for missing tag data, a guide for how to use the output of the GridTool in power system models and a step-by-step illustrative example.

*Corresponding author:
URL: `robert.gaugl@tugraz.at` (R. Gaugl)

[1] `https://github.com/IEE-TUGraz/GridTool`

Contents

1. Overview

The GridTool presented in this paper does not rely on relations that are often missing in OSM data, but instead uses heuristics to identify power stations and lines. The main point of the GridTool is that it also condenses grid data for optimal usage in PSMs, where unnecessary detailed information prolongs simulation times by increasing the number of variables without adding any value in the results. The GridTool is programmed in only one language (MATLAB), which makes it really simple to understand and to enhance the code.

The GridTool is open-source, so you can download and use this tool freely, but if you do, please cite [1].

2. System Requirements and Licenses

In order to run the GridTool, the following software tools are required:

1. **MATLAB**. The GridTool is coded in MATLAB and so in order to use the GridTool MATLAB is required. The GridTool is tested with MATLAB version R2021a, but is programmed in a way that it should be compatible with older and newer versions.
2. **Microsoft Excel**. The results of the GridTool are stored in Microsoft Excel files (.xlsx).

We denote GridTool as an *open-source* tool because its source code is freely available. The user must be aware that, in order to use the GridTool, licenses for the software products mentioned above (MATLAB and Microsoft Excel) are needed.

3. Download and installation

1. Follow this link:`https://github.com/IEE-TUGraz/GridTool`
2. Download the ZIP file by clicking on the green 'Code' button and selecting 'Download ZIP'
3. Unzip and keep all files in the same folder

4. Software Architecture

The GridTool is split into 6 modules that itself are divided into 24 functions as shown in Figure 5 to ensure clarity, maintainability and easy expandability. The flowchart in Figure 5 shows the 6 modules and 24 functions. A detailed description of the functionality of these functions can be found in the following chapters 4.1-4.6, as well as in the official GridTool paper [1] and in the comments of the MATLAB code itself.

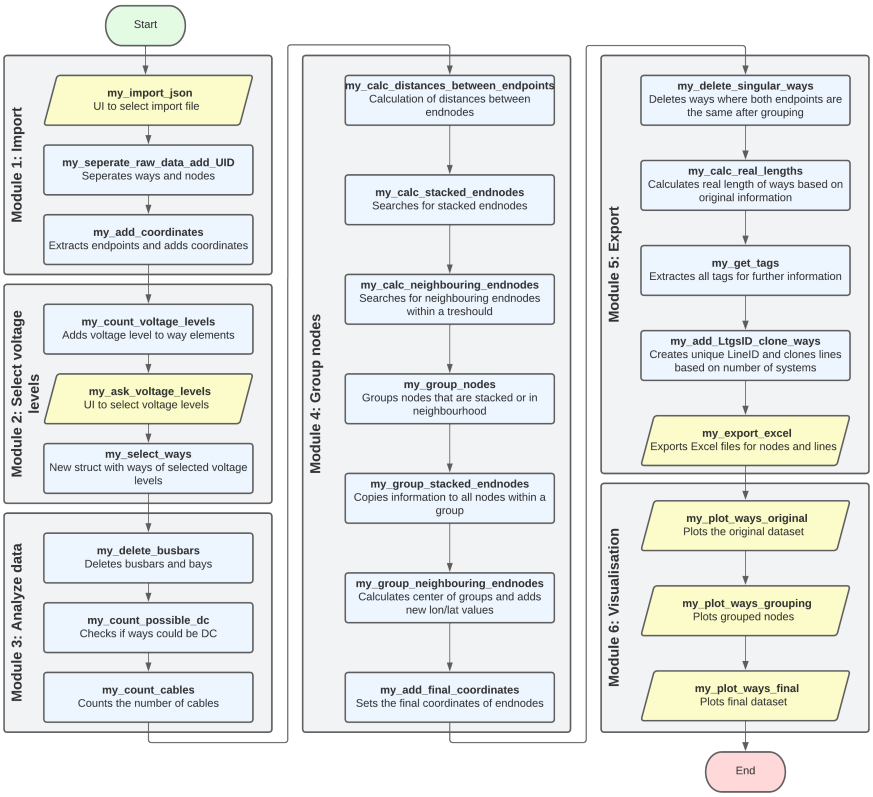


Figure 1: Flowchart showing the 6 modules and 24 functions of the GridTool.

4.1. Module 1: Import

The first module ensures that the data set from OSM in the JSON format (a file that stores simple data in a standard data interchange format) is imported

into MATLAB and separated into nodes and ways. (In the context of OSM lines are called ways or way-elements. For the sake of clarity, we also use this designation in this manual.) Also, the end nodes of the lines are extracted.

4.1.1. my_import_json

The function *my_import_json* imports the JSON data from OSM. A user interface for selecting the JSON file is opened. With the MATLAB function 'jsondecode' the file is imported into the MATLAB workspace.

4.1.2. my_seperate_raw_data_add_UID

In the next step the function *my_seperate_raw_data_add_UID* separates the imported data into nodes and way elements and stores them in separate variables. Every way element gets a unique identifier (UID) assigned for easier differentiation in later processes.

4.1.3. my_add_coordinates

In the first step, the function *my_add_coordinates* extracts the start and end nodes of all way elements as shown in Figure 2. All the other nodes that represent the exact route of the power line are dismissed as this information is not important for PSMs. The longitude and latitude coordinates of the start and end nodes are added to the corresponding way elements. For easier calculations, the x/y-coordinates of all nodes are calculated and also stored. These allow for easier calculation of the beeline distances (shortest distance from the start to end nodes) between nodes using the Pythagorean theorem. If *bool.calculate_real_line_length* is set to false, this length is also used in the output.

4.2. Module 2: Voltage selection

The second module gives the user an option to choose the voltage levels that should be considered by the GridTool.

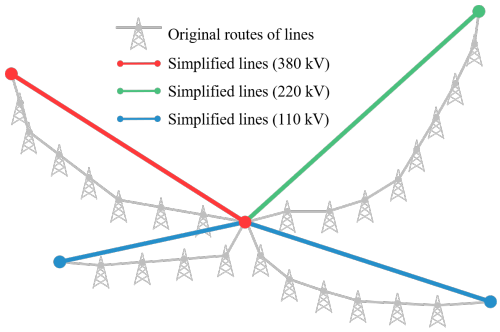


Figure 2: Visualization of the simplification process. Grey lines show original routes of lines. The simplified colored lines dismiss the exact route and only have a start and an end node. The node in the middle are actually three nodes stacked on top of each other for the three different voltage levels.

4.2.1. my_count_voltage_levels

The function *my_count_voltage_levels* reads the voltage information from the source file and adds it to the way elements. If there are lines with different voltage levels going between two nodes, they are displayed as one way-element with different voltage levels in the OSM-data. This function identifies the lines with multiple voltage levels and clones these to make separate lines for each voltage level. After that the number of way-elements for each voltage level are counted and displayed in the MATLAB console.

4.2.2. my_ask_voltage_levels

In a pop-up presented by the function *my_ask_voltage_levels*, the user can then choose the voltage levels that should be considered going forward.

4.2.3. my_select_ways

All way-elements that have one of the selected voltage levels are copied to a new data set by the function *my_select_ways*.

4.3. Module 3: Data analysis

This module does some analysis based on "tags" from the OSM-data and on heuristic assumptions to find and delete busbars, detect possible DC-lines and

count the number of systems per way-element. This module is made up of three different functions.

4.3.1. my_delete_busbars

The function *my_delete_busbars* goes through every way-element and checks if it has a tag that declares it as a busbar or bay. If the beeline length of the way-element is smaller than the length set in *busbar_max_length*, then the element is deleted. If the busbar or bay has a length greater than *busbar_max_length* a message is printed in the MATLAB console so the user can have a closer look into it. In this case, the busbar is included in the data set and considered as a normal line. This process is shown in Figure 4a picturing the detailed OSM-data of a power station and Figure 4b after removing all busbares and bays.

4.3.2. my_count_ possible_dc

As direct current (DC)-lines might need special considerations (e.g. different line parameters compared to alternative current (AC)-lines) the function *my_ count_ possible_dc* looks for three signs to identify DC-lines: It only has one cable, the frequency is "0" or the name of the line contains the phrase 'dc'. If one or more of the three conditions are true, "potentially DC" is added to the notes field of the way-element in the final result.

4.3.3. my_count_cables

The last function in this module (*my_count_cables*) counts the number of systems per way-element. The OSM-data contains information about the number of cables on a way-element. If it is a standard AC-system, three cables are needed for one system, six cables for two systems and so on. This function goes through every way-element and checks the number of cables. A new field "systems" is added to the data set for way-elements. This contains the number of systems that can be derived from the number of cables. For way-elements that have '3', '6', '9' or '12' cables the "systems" field is automatically set to '1', '2', '3' or '4' respectively. If a way-element has more than one voltage level

(and so the tool doesn't know which of the voltage levels has how many systems) or if there are way-elements with number of cables other than the ones mentioned above a warning is displayed in the MATLAB console. The number of systems is later used in the function *my_add_LtgsID_clone_ways* to double, triple or quadruple the line.

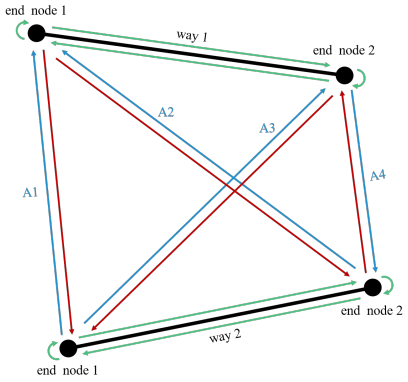
4.4. Module 4: Grouping

The fourth module is the most important one. It consists of seven functions that heuristically find concentrations of nodes that are typical for power stations and groups them together.

4.4.1. my_calc_distance_between_endpoints

In order to identify nodes that can be grouped together based on their proximity, the distances between all nodes have to be known. The function *my_calc_distance_between_endpoints* creates a matrix with the all way-elements and their end nodes in the column headers as well as in the row headers. The distances between the end nodes are the values of the matrix. As the distance from end node 1 of way 1 to end node 1 of way 2 is the same the other way around, the resulting matrix is diagonal symmetric. Therefore, all elements below the diagonal in the matrix are set to "NaN" as they do not add any value. Elements in the diagonal are set to '-1' as they either represent the distance between the same end node of a way (e.g. distance of end node 1 of way 1 to itself) or the distance between end node 1 of way 1 to end node 2 of way 1, which is the (already calculated) beeline length of the line. Only the distances between the end nodes above the diagonal are calculated using an efficient and fast matrix-calculation. To better understand this, Figure 3a shows an example of two way-elements: way 1 and way 2 each with the corresponding end nodes 1 and 2. This creates the matrix displayed in Figure 3b. The values in the diagonal are set to '-1' (green paths in Figure 3a) and the values below the diagonal are set to 'NaN' (red paths in Figure 3a). The distance calculated between end node 1 of way 1 and end node 1 of way 2 is A1 and so on. Therefore, a matrix with all the

distances between all end nodes is created.



(a) Two way element

				way 1		way 2	
				end node 1	end node 2	end node 1	end node 2
				lon	lon	lon	lon
				lat	lat	lat	lat
way 1	end node 1	lon	lat	-1	-1	A1	A2
	end node 2	lon	lat	-1	-1	A3	A4
way 2	end node 1	lon	lat	NaN	NaN	-1	-1
	end node 2	lon	lat	NaN	NaN	-1	-1

(b) Resulting matrix

Figure 3: Example of distance between end nodes with two way element.

4.4.2. my_calc_stacked_endnodes

The function *my_calc_stacked_endnodes* goes through this matrix and searches for combinations where the distance between two endpoints is exactly '0' (stacked nodes). A list with all pairs of these end nodes is created.

4.4.3. my_calc_neighbouring_endnodes

The next function *my_calc_neighbouring_endnodes* also goes through the matrix and creates a list with all pairs of end nodes that have a distance that is bigger than '0' but smaller than the distance set by *neighborhood_threshold* to identify power stations. The heuristic behind that is, that the OSM-data is so detailed, that lines that end in the same power station do not exactly have the same end nodes but form a typical concentration of nodes.

4.4.4. my_group_nodes

The function *my_group_nodes* is run two times, with the lists created by the functions above. It goes through those lists of paired end nodes and groups them. For example, the function *my_calc_stacked_endnodes* just says that endpoint A

is stacked with endpoint B. But endpoint B could also be stacked with endpoint C. The function *my_group_nodes* will then group those three nodes together and the result is a list with nodes that should be condensed to one node.

4.4.5. my_group_stacked_endnodes

In the function *my_group_stacked_endnodes* the first node of each stacked group from the resulting list of *my_group_nodes* replaces all the other nodes in this list in the way-elements, meaning that two ways that had different end nodes ending in the same power station have the same end node now (if they have the same voltage level).

4.4.6. my_group_neighbouring_endnodes

The function *my_group_neighbouring_endnodes* does the same thing for grouped neighboring end nodes. Instead of using the lon/lat-coordinates of the first member in that group, the lon/lat-values of all members in that group are used to calculate the mean lon/lat-value.

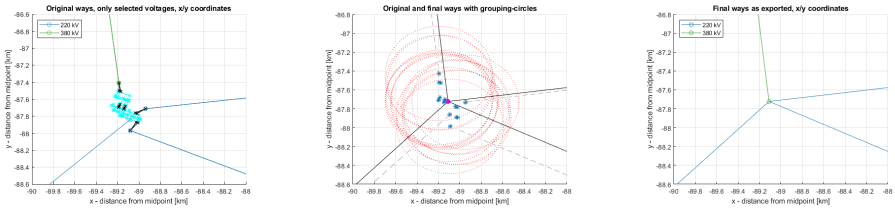
4.4.7. my_add_final_coordinates

The last function *my_add_final_coordinates* in this module updates the original data set and replaces end nodes that got grouped with the coordinates and IDs of the grouped nodes.

A visualization of the working of module 4 can be seen in the Figure 4b. The red circles represent the distance set by *neighbourhood_threshold*. All end nodes within a circle will be grouped together. The final result of this grouping is displayed in Figure 4c.

4.5. Module 5: Export

The result is prepared and exported into two Excel-files: One containing all the nodes and the second one all the lines.



(a) Original (b) Grouping stage (c) Final data

Figure 4: Resulting Excel files.

4.5.1. my_delete_singular_ways

Before the export process starts, the function *my_delete_singular_ways* deletes lines that got condensed into "singular" ways (same start and end node) after grouping.

4.5.2. my_calc_real_lengths

So far, the line length has only been calculated based on the simplified lines. As the GridTool keeps the information about the exact route of a line in the background, the function *my_calc_real_lengths* now uses this information to calculate the real line lengths based on the exact route of a line. To do so, the length of every line element (see Figure 2 grey lines) is calculated and then summed. This calculation can take some time and can therefore be turned on or off based on the setting of the boolean variable *bool.calculate_real_line_length*. If this variable is set to False, the calculated beeline length of the simplified lines is used in the result.

4.5.3. my_get_tags

The database of OSM contains lot of other information in the form of 'tags' that might be useful to the user after the GridTool is finished. Therefore, those tags are copied into a new variable by the function *my_get_tags*. In the final export process, this information is stored in 'sheet2' of the Excel file containing the lines, so that the user can review this additional information.

4.5.4. my_add_Ltgs_ID_clone_ways

In the next step, the function *my_add_Ltgs_ID_clone_ways* clones way-elements based on the number of systems (see section Module 3: Data analysis). Furthermore, this function creates a unique ID for the lines that is used in the final Excel file. In its current form the ID is composed of the letters 'LTG', a country code that can be set in the variable *export_excel_country_code* and a four-digit number starting at '0001'. Way-elements that get cloned get the same ID, but a suffix added in the form of 'a', 'b', 'c'. Figure 5 and Figure 12 show examples of the final excel sheets.

4.5.5. my_export_excel

The last function *my_export_excel* exports the final data into two Excel files. It also creates unique IDs for the nodes, that are used in those final results. The Excel file containing the node information has the columns NodeID (unique ID of the nodes), Country (country code set by *export_excel_country_code*), Voltage (voltage level of the node in kV), lat (latitude of node in °) and lon (longitude of node in °). The Excel file containing the line data has two sheets. In the first sheet the actual information is stored in the following columns: LineID (unique line id), Country (country code set by *export_excel_country_code*), fromNode (start node of the line), toNode (end node of the line), Voltage (voltage level of the line in kV), Length (line length in km), Note (additional information). The sheet also has some pre-defined columns that have a value of '0': R (resistance), XL (inductive reactance), XC (capacitive reactance), Itherm (thermal limit current) and PhiPsMax (maximum phase shifting angle). Those values are important for load flow calculations, but OSM does not provide values for those. It is up to the user to come up with plausible values.

4.6. Module 6: Visualization

This optional module is responsible for displaying and visualizing the data set.

4.6.1. my_plot_ways_original

If the boolean variable *bool.plot_ways_original* is set to 'True', the function *my_plot_ways_original* plots the original ways/lines as seen in Figure 4a, where a power station with detailed busbars is shown.

4.6.2. my_plot_ways_grouping

The function *my_plot_ways_grouping* plots the original ways with the grouping circles and final lines (dashed) as shown in Figure 4b, if the boolean variable *bool.plot_ways_grouping* is activated.

4.6.3. my_plot_ways_final

The final data set can be displayed by enabling the boolean variable *bool.plot_ways_final*. The result is shown in Figure 4c. The power station got combined into two nodes (one for 220 kV and one for 380 kV) that have the same coordinates.

5. How to run the GridTool

This provides a step-by-step instruction on how to run the GridTool. The steps are also visualized in the Figure 5.

1. Download and Unzip the GridTool code as described in Section 3.
2. Download the electricity grid data from OpenStreetMap using the OverpassTurbo website. For a detailed guide see Section 6. An example of a query to download Austria's electricity grid lines between 220 kV and 380 kV can be found in the file OverpassTurbo_Example_Query.txt inside the downloaded ZIP file. Run the query in OverpassTurbo and export the result as 'Raw OSM data', which downloads the data in the json file format. For testing purposes you can also use the *2022-08-02_Austria_220kV_and_380kV.json* file provided in the repository.
3. Open the GridTool.m file in Matlab.
4. Adjust the settings in the Initialization and Settings section of the code. A description of these settings can be found in Section 7 as well as in the MATLAB code itself.
5. Run the code by clicking the 'Run'-button or pressing 'F5'.
6. A file selection window opens to select the previously downloaded json file.
7. After a few seconds, a new window pops up to select the desired voltage levels. Multiple voltage levels can be selected by holding the 'Alt' key.
8. The GridTool is completed, when it displays 'CONVERSION COMPLETED' in the MATLAB console. The final files can be found in the folder where the GridTool.m file is saved. Be sure to check the MATLAB console for any warnings (see Section 8).

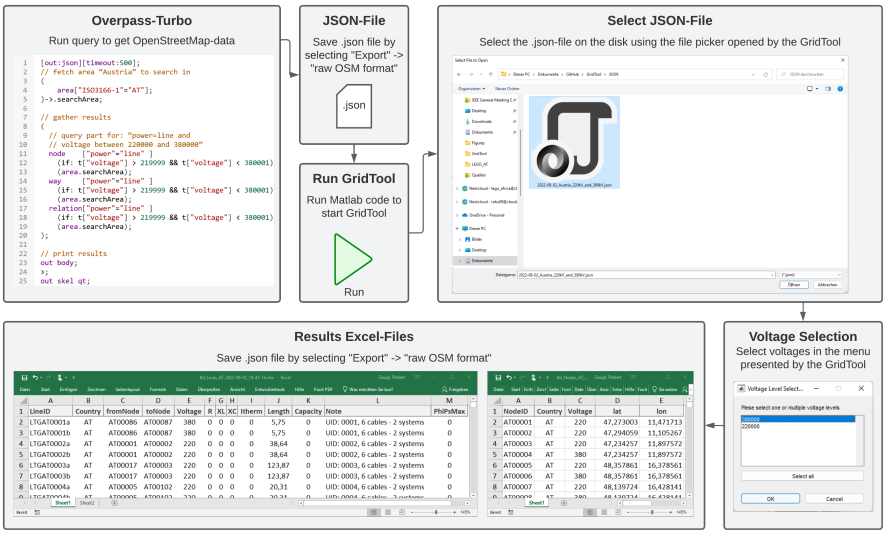


Figure 5: Step-by-step guide for how to use the GridTool.

6. Download Grid Data from OpenStreetMap using OverpassTurbo

OverpassTurbo[2] is an online data filtering tool for OpenStreetMap and is used to download the required grid data in the json file format as input for the GridTool.

OverpassTurbo can either be used with an integrated wizard or with advanced customized queries. Both ways are explained in the next subsections.

6.1. Wizard

A very simple way to download grid data from OSM using OverpassTurbo is by using the integrated wizard, which generates a query based on a few simple commands.

1. Open `https://overpass-turbo.eu/` in your browser (Mozilla Firefox or Google Chrome recommended for good performance).
2. Click on the 'Wizard' button.
3. Enter the following text to download the 220 kV and 380 kV power lines for Austria: *power=line and voltage=220000 or voltage=380000 in Austria*
4. Click the 'Create and run query' button.
5. Click 'Continue anyway' if a big data warning pops up.
6. If the query runs into a timeout, raise the '[timeout: xx]' limit (standard: 25 seconds) in the left code area of the window.
7. Click the 'Export' button and select download next to 'OSM raw data' to download the needed json file.

The result of this wizard query can be seen in Figure 6.

In Table 1 are a few important options to customize the query used in the wizard by adding the commands with 'and' or 'or', depending on your needs:

The wizard has some downsides when more complex queries are needed. With the wizard it is complicated to create a query for 'voltage bigger than' or

[2]URL: `https://overpass-turbo.eu/`

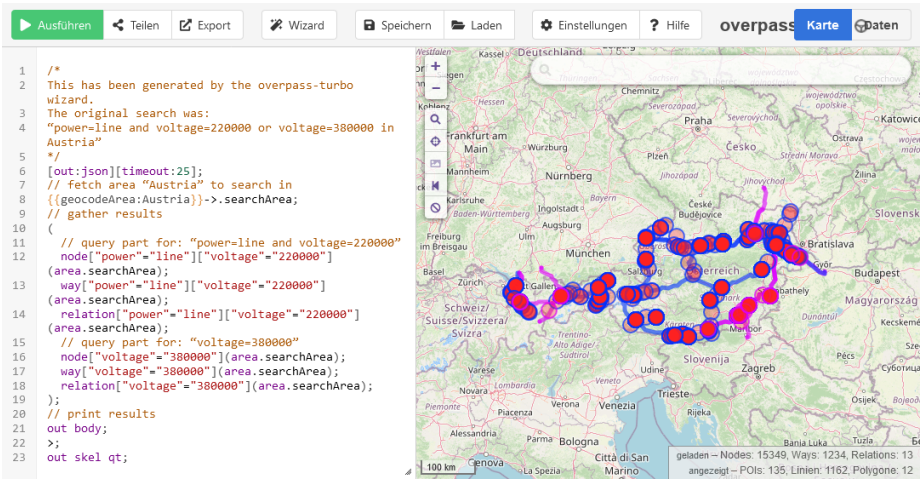


Figure 6: Result of using the wizard for the example above.

Query	Output
power=line	Overhead lines
power=cable	Cables
voltage=XXX	Only elements with the mentioned voltage level in Volt.

Table 1: Important queries for the wizard.

for more than on country/area. For these scenarios, we recommend using the advanced customized queries in Section 6.2.

A detailed explanation of the wizard can be found here: `https://wiki.openstreetmap.org/wiki/Overpass_turbo/Wizard`

6.2. Customized Queries

The custom queries allow for more advanced data queries but are also a little bit harder to use for novice users. We want to give two simple examples that the user can further customize to get the needed results.

The customize queries are written in the left code area of the OverpassTurbo window.

In the first example we replicate the query from the wizard example above. In line 1 the output is set to be in the json format and in line 2 the timeout is

set to 120 seconds. In line 4 the area is set to Austria and in lines 7-9 we gather all the necessary grid information for 220 kV lines (which inlcudes nodes, ways and relations). The '~' takes every line that contains 220 kV in its voltage-tag information. If we would use '=' only lines that only have 220 kV in its voltage-tag would be selected. In 11-13 we do the same for the 380 kV lines. Line 16 tells the program to display the results.

```
1  [out:json]
2  [timeout:120];
3
4  {{geocodeArea:Austria}}->.searchArea;
5
6  (
7      node["power"="line"]["voltage"~"220000"](area.searchArea);
8      way["power"="line"]["voltage"~"220000"](area.searchArea);
9      relation["power"="line"]["voltage"~"220000"](area.searchArea);
10
11     node["power"="line"]["voltage"~"380000"](area.searchArea);
12     way["power"="line"]["voltage"~"380000"](area.searchArea);
13     relation["power"="line"]["voltage"~"380000"](area.searchArea);
14 );
15
16 out body;
17 >;
18 out skel qt;
```

In the second example, we want to download the overheadlines and cables with a voltage level of 110 kV and above for Austria and Germany. As this is a larger query and takes longer to process, we increase the timeout to 500 seconds in line 2. In lines 3-6 we define the search area based on the ISO3166-1 country codes[3]. In our example we want to have results for Austria (AT) and Germany (DE). In line 9-11 we want to gather information for lines that have a voltage level of 110 kV and above. For this we use the 'if' statement and because bigger or equal is not supported, we say that the voltage level should be bigger than

[3]See `https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes`

109,999 Volts. In lines 13-15 we do the same for cables. In Figure 7 the result of this query can be seen.

```
1  [out:json]
2  [timeout:500];
3  (
4          area["ISO3166-1"="AT"];
5          area["ISO3166-1"="DE"];
6  )->.searchArea;
7
8  (
9    node["power"="line"](if: t["voltage"] > 109999)(area.searchArea);
10   way["power"="line"](if: t["voltage"] > 109999)(area.searchArea);
11   relation["power"="line"](if: t["voltage"] > 109999)(area.
         searchArea);
12
13   node["power"="cable"](if: t["voltage"] > 109999)(area.searchArea)
         ;
14   way["power"="cable"](if: t["voltage"] > 109999)(area.searchArea);
15   relation["power"="cable"](if: t["voltage"] > 109999)(area.
         searchArea);
16 );
17
18 out body;
19 >;
20 out skel qt;
```

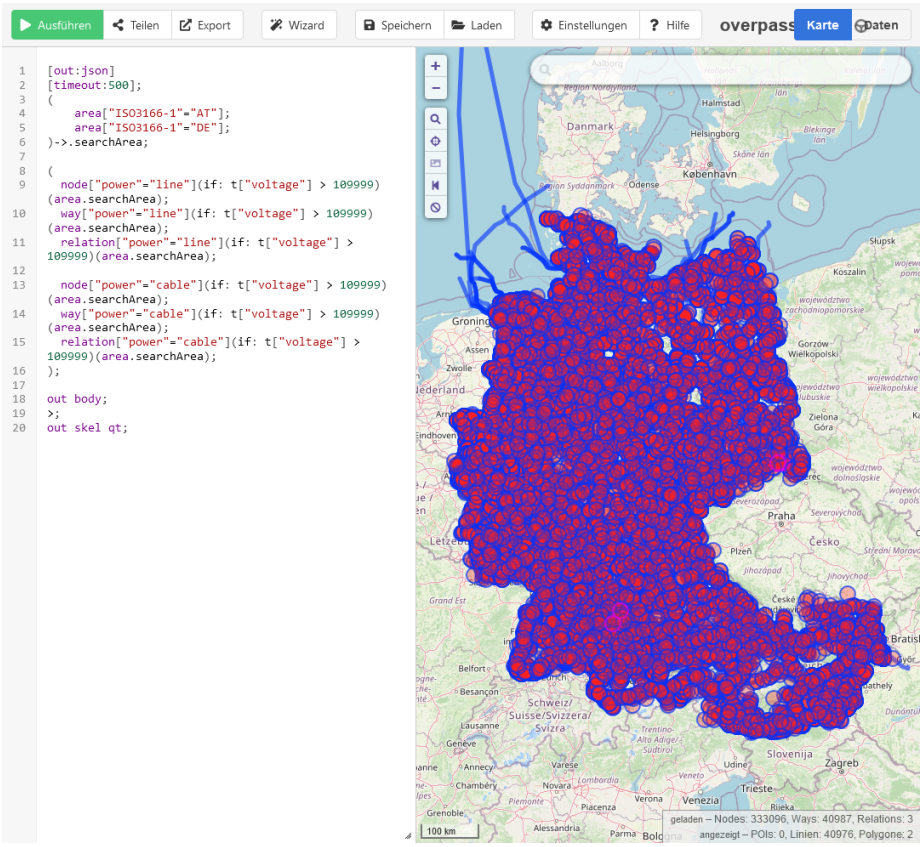


Figure 7: Result of the customized query above. The red filled circles mean, that there is more information if we zoom in closer.

7. Options

In the *Initialization & Settings* section of the MATLAB code there are some options that can be changed and modified. Table 2 gives an overview and a short description of the available options.

Option	**Standard**	**Unit**	**Explanation**
export_excel_ country_code	AT	-	Country code that will be added to NodeIDs, LineIDs in the column Country.
neighbourhood_ threshold	0.5	km	Nodes that are within that radius will be grouped together.
busbar_max_ length	1	km	Busbars with a length smaller than that will be deleted.
way_length_ multiplier	1.2	-	Multiplier factor for calculated line lengths to consider line slack.
bool.calculate_ real_line_length	True	boolean	If set True, line length will be calculated based on original data. If set False, line length will be calculated based on beeline between the start and end nodes of the simplified lines.
bool.plot_X	False	boolean	Activating or deactivating plot X (see description in code).

Table 2: Available options that can be changed in the settings part of the GridTool with short explanation.

8. Warnings

The GridTool provides the user with warnings in the MATLAB console if it detects inconsistent data or other errors. To help the user with handling the warnings, they are explained here with possible solutions.

- **ATTENTION! There is at least one way/node element which has a different amount of fields. It won't be imported!**: If the data exported from OSM has corrupted elements (hence, a field like "tags" is missing), this element will be ignored. A manual review of the *.json file is necessary.
- **ATTENTION! Way element UID XY does not contain a field "voltage". This way won't be selected**: Lines that do not have a field 'voltage' will be skipped. The lines with the printed UID should be checked manually by the user.
- **ATTENTION! UNKNOWN voltage level ("XYZ") in UID XY. This way won't be selected.**: This error can happen if the voltage level in the field "voltage" is either empty or text instead of a number. A manual review of that line with that UID in the *.json file is necessary.
- **ATTENTION! Two/Three voltage levels ("XYZ") in UID XY. This way will be duplicated/tripled**: The user is informed, that a line has two or three voltage levels and the line is doupled/tripled. No immediate user action needed, but the user can check with other sources if this is correct.
- **ATTENTION! Way Element UID XY has type "busbar" or "bay", but is too long. Length: X km of max. Y km. This way won't be added to the "busbar" exception list**: The user is informed that a line has a value "busbar" or "bay" but its length is longer than the length set in *busbar_max_length*. Therefore, this busbar is handled like a line for the following steps.

- **ATTENTION! Unknown cable number ("X") in UID XY. This way won't be cloned automatically**: This can happen if a line has more than one voltage level and more than 2 systems. Based on the data the GridTool cannot know which voltage level has how many cables. This line is not cloned. A manual review of that line in the results is necessary. A check with other sources is recommended.

- **ATTENTION: Real line lenght WON'T be calculted! Beeline-length (Luftlinie) will be used**: The user is informed, that *bool.calculate_real_line_length* is set to false and therefore the beeline-length (shortest distance between start and end nodes) is used in the results.

- **ATTENTION! More than 12 voltage levels are selected. Colors of voltage lines do repeat now! It is recommended to select max. 12 voltage levels**: For plotting the results, 12 colors for lines with different voltage levels that work good together have been defined. If there are more than 12 voltage levels the user is informed, that the colors repeat again. The user could define more colors in the MATLAB code.

9. Default behaviour for missing tag data

The GridTool also uses tag information from OSM to find out the frquency of a line, voltage level, number of systems and so on. If some of the data is missing, the GridTool will fall back to defeault behaviors (but warnings will be shown in the MATLAB console).

- **No voltage level**: The line will be skipped and a warning message (see Section 8) will be displayed to manually check the line.
- **No frequency**: It is assumed to be a 50 Hz AC-line.
- **No number of cables**: It is assumed to be just one system/line. A warning message (see Section 8) will be displayed.
- **No name**: The GridTool does not rely on the name of an element. Therefore, an empty name tag will be ignored.

10. Using Output of GridTool in Power System Models

1. Before using the data for power flow caculcations in PSMs, the results should be reviewed and rechecked with other sources (e.g. ENTSO-E Transmission System Map [2]).
2. The column 'Note' should be checked for the following comments, which indicate a problem with the line:
   (a) **multiple vlevels**: Line has multiple voltage levels assigned to it. If it has more systems than voltage levels, the line might not have been multiplied correctly. See also warning in the MATLAB console.
   (b) **potentially DC**: This indicates that the line could be a DC-line based on tag information. Recheck with other sources advised.
3. Add values for electric parameters (R, XL, XC, Itherm) to each line. Either from other sources like public TSO data or if no other sources are available, typical parameters for the voltage level can be assumed e.g. from [3] (this can make power flow calclulations less accurate).
4. To allow power flow between different voltage levels, transformers need to be added between nodes at the same power station. These can easily be identified by the same lat- and lon-coordinates. You can also recheck with public data from TSOs.
5. Integrate the data into your PSM.
6. Add electricity demand per node. This can vary greatly between models, so please refer to the manual of the PSM you are using on how to add electricity demand distribution.

11. Illustrative Example: Austrian Transmission Grid

For this illustrative example, we do a walk-through from downloading the data with OverpassTurbo to preparing them for a model with load flow calculations.

1. Go to `https://overpass-turbo.eu/` and download the Austrian transmission grid (220 kV and 380 kV lines) with the following code:

```
[out:json]
[timeout:500];
(
        area["ISO3166-1"="AT"];
)->.searchArea;

(
    node["power"="line"]["voltage"~"220000"](area.searchArea);
    way["power"="line"]["voltage"~"220000"](area.searchArea);
    relation["power"="line"]["voltage"~"220000"](area.searchArea);

        node["power"="line"]["voltage"~"380000"](area.searchArea);
    way["power"="line"]["voltage"~"380000"](area.searchArea);
    relation["power"="line"]["voltage"~"380000"](area.searchArea);
);

out body;
>;
out skel qt;
```

2. The result should look like Figure 8.

3. Download the result as a JSON file by clicking on the 'Export' button and then on 'Download' next to 'OSM raw data' (see Figure 9). The 'export.json' file will be saved in your Downloads folder or your chosen location.

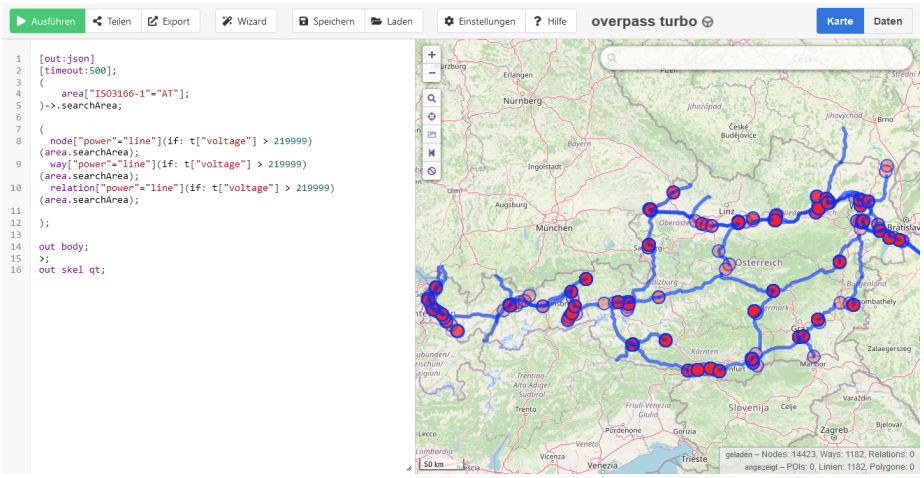


Figure 8: Result for Austria's transmission lines.

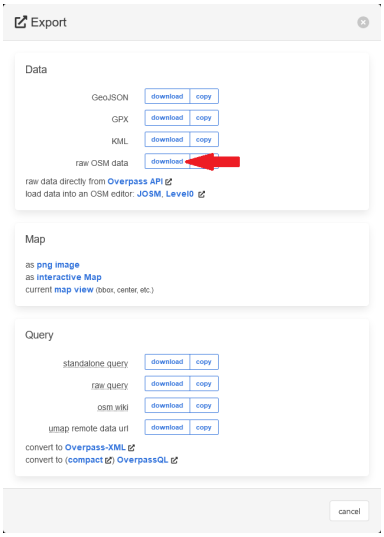


Figure 9: Downloading the JSON file with OverpassTurbo.

4. Open the GridTool with MATLAB (see Section 3 on how to download and install the GridTool) and start the GridTool by clicking on the 'Run' button or pressing 'F5'.
5. A new file selecting window pops-up, where you have to select the previously downloaded JSON file.

6. After that the window for selecting the desired voltage levels pops up. Multiple voltage levels can be selected by holding the 'Alt' key. (Note: Even though the code for OverpassTurbo only included lines with voltage levels with 220 kV and 380 kV, the window also displays 0 and 110 kV. This is because some lines have more than one voltage level assigned to it. So if the voltage tag of OSM contains 110 000 and 220 000 volts, the line will be selected by the OverpassTurbo query.) We only want the 380 000 V and 220 000 V lines. So we select these two voltage levels by clicking on them while holding the 'Alt' key.

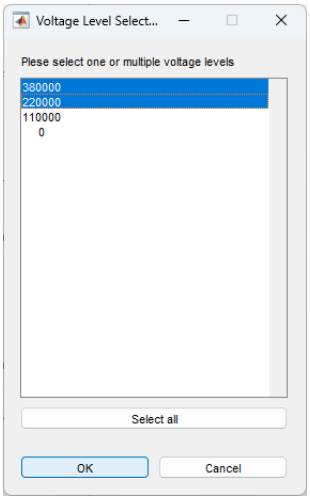


Figure 10: Selecting the desired voltage levels.

7. After a few seconds the MATLAB console displays 'COVERSION COMPLETED' and (if not turned off in the *Initialization and Settings* part of the code) the standard output figures pop up (see Figure 11 for an example).

8. The excel files with the results can be found in the folder where the GridTool.m file is located. A screenshot of those files can bee seen in Figure 12.

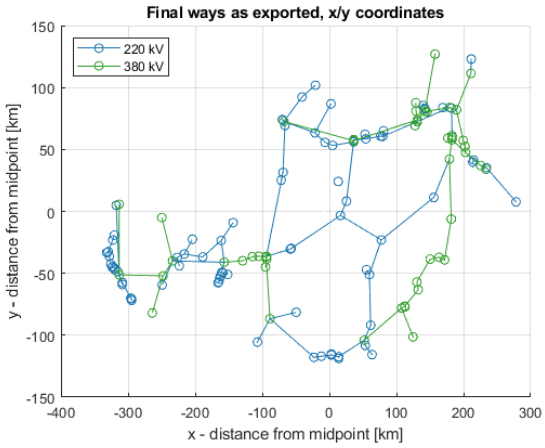
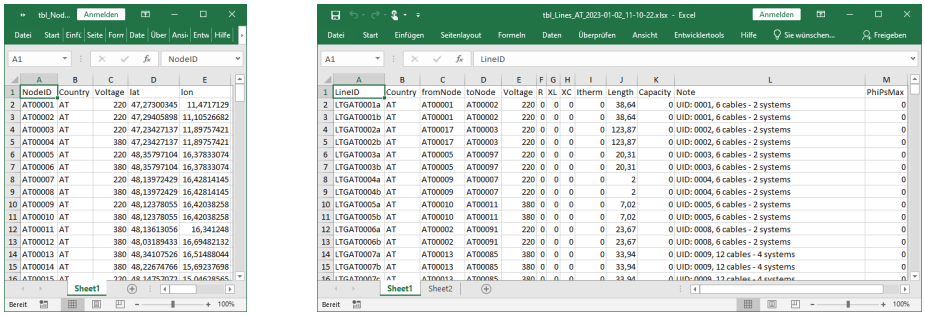


Figure 11: Plot displaying the result.



(a) Nodes

(b) Lines

Figure 12: Resulting Excel files.

9. The information from the Excel files with the nodes can be taken as is and implemented in the chosen PSM. Note: Depending on the PSM you may have to reformat the information a bit. Also: For some PSMs you have to add the electricity demand directly to the nodes.
10. In the lines-Excel we have to add some information for the electric parameters of the lines. For this Austrian example we can use public information from Austria's TSO [4]. If reliable public information is not available, we can try to come up with numbers based on other sources, for example [3] has some typical values for electric parameters of lines per voltage level.

Depending on the quality of the information, this can take a little bit of time. Note 1: This is a good step to verify the lines from the GridTool with official public data if available. Note 2: Depending on your chosen PSM, the values for R, XL and XC can be needed as primary line constants (i.e. per km) or already multiplied with the length of the line.

11. Add transformers to the data. If official public data is available, this can be used to integrate the transformers. If not, a simple first approach could be to add transformers between nodes with the same coordinates but different voltage levels. Note: Depending on the chosen PSM the exact implementation of transformers can look quite different. Please refer to the manual of the PSM you are using.
12. Run your PSM. Figure 13 shows an example of running a DC-OPF with the open-source LEGO model [5]. The grid data was obtained by the GridTool and combined with official public information from Austria's TSO for electric parameters and transformers. Power plants and demand distribution were taken from a non-public data base that our institute has gathered together manually over years.

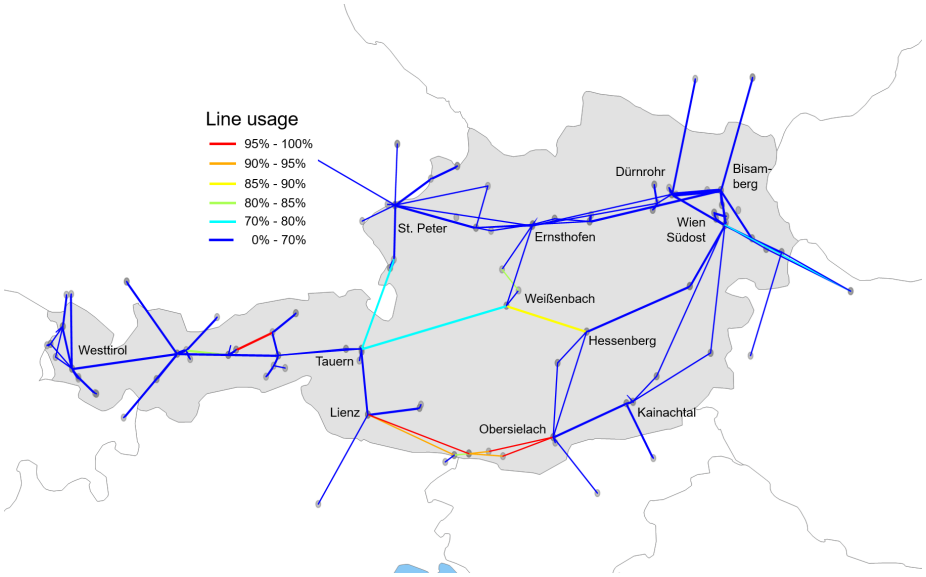


Figure 13: Example of an optimal power flow simulation with the LEGO model using grid data ouput from the GridTool.

12. APPENDIX A: Nomenclature

Acronyms:

AC/DC	Alternating/direct current
PSM	Power system model
OPF	Optimal power flow

References

[1] R. Gaugl, S. Wogrin, U. Bachhiesl, L. Frauenlob, Gridtool: An open-source tool to convert grid data (under review)., SoftwareX.

[2] ENTSO-E, ENTSO-E Transmission System Map (2019).
URL https://www.entsoe.eu/data/map/

[3] F. Kießling, P. Nefzger, U. Kaintzyk, Freileitungen: Planung, Berechnung, Ausführung, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. doi:10.1007/978-3-642-56738-4{_}3.
URL https://doi.org/10.1007/978-3-642-56738-4_3

[4] APG, Static Grid Model (2022).

[5] S. Wogrin, D. A. Tejada-Arango, R. Gaugl, T. Klatzer, U. Bachhiesl, LEGO: The open-source Low-carbon Expansion Generation Optimization model, SoftwareX 19 (2022) 101141. doi:10.1016/J.SOFTX.2022.101141.
URL https://linkinghub.elsevier.com/retrieve/pii/S2352711022000887