SciGRID_gas user manual

Release 1.0

J. C. Diettrich & W. Medjroubi & A. Pluta & J. Dasenbrock

Jan 27, 2020

CONTENTS

Impressum

DLR Institute for Networked Energy Systems
Carl-von-Ossietzky-Str. 15
26129 Oldenburg
Germany
Tel.: +49 (441) 999 060

LIST OF FIGURES

LIST OF TABLES

Summary

The goal of SciGRID_gas is to develop methods to create an automated model that can generate a gas transmission network data set for Europe. Gas transmission networks are fundamental for simulations by the gas transmission modelling community, to derive major dynamic characteristics. Such simulations have a large scope of application, for example, they can be used to perform case scenarios, to model the gas consumption, to minimizing leaks and to optimize overall gas distribution strategies. The focus of SciGRID_gas will be on the European transmission gas network, but the principal methods will be also applicable to other geographic regions.

Data required for such models are the facilities, such as compressor stations, LNG terminals, pipelines, etc. One needs to know their locations, in addition to a large range of attributes, such as pipeline diameter and capacity, compressor capacity, configuration, etc. Most of this data is not freely available. However throughout the SciGRID_gas project it was determined, that data can be found and grouped into two fundamental different groups: a) OSM data, and b) non-OSM data. The OSM data is geo-referenced facilities data, that is stored in the OpenStreetMap data base, and is freely available. However the OSM data set contains hardly any other information than the location of the facilities. The Non-OSM data set can fill some of the gaps, by supplying information such as pipeline diameter, compressor capacity and more. Part of the SciGRID_gas project is to mine and collate such data, and merge it with the OSM data set. In addition heuristic tools are required to fill data gaps, so that a complete gas network data set can be generated.

Here, this document describes the first non-OSM data set, called the "InternetData" data set, which is one of the fundamental building blocks of the SciGRID_gas project. The InternetData data set is a test data set, and is the first data set to be published as part of the SciGRID_gas project. This document explains the origin and structure of the InternetData data set.

In this document, the section "Introduction" will supply some background information on the SciGRID_gas project, followed by the section "Data structure", that gives a detailed description of the data structure that is being used in the SciGRID_gas project with respect to all gas facilities data sets. Section "Data sources" describes the InternetData data set. This is followed by section "Data input and output", which describes the options of how data can be added to the a SciGRID_gas data set, or how a resulting SciGRID_gas data set could be saved to file.

The appendix contains a glossary, references, location name alterations convention and finishes with the table of country abbreviation.

CHAPTER

ONE

INTRODUCTION

SciGRID_gas is a three-year project funded by the German Federal Ministry for Economic Affairs and Energy [BMWi] within the funding of the 6. Energieforschungsprogramm der Bundesregierung [6EnForProBunReg]

The goal of **SciGRID_gas** is to develop methods to generate and provide an open-source gas network data set and code. Gas transmission network data sets are fundamental for the simulations of the gas transmission within a network. Such simulations have a large scope of application, for example, they can be used to preform case scenarios, to model the gas consumption, to detect leaks and to optimize overall gas distribution strategies. The focus of **SciGRID_gas** will be the generation of a data set for the European Gas Transmission Network, but the principal methods will also be applicable to other geographic regions.

Both the resulting method code and the derived data will be published free of charge under appropriate open-source licenses in the course of the project. This transparent data policy shall also help new potential actors in gas transmission modelling, which currently do not possess reliable data of the European Gas Transmission Network. It is further planned to create an interface to [SciGRID_power] or heat transmission networks. Simulations on coupled networks are of major importance to the realization of the German *Energiewende*. They will help to understand mutual influences between energy networks, increase their general performance and minimize possible outages to name just a few applications.

This project was initiated, and is managed and conducted by DLR Institute for Networked Energy Systems.

1.1 Project information

- **Project title**: Open Source Reference Model of European Gas Transport Networks for Scientific Studies on Sector Coupling (*Offenes Referenzmodell europäischer Gastransportnetze für wissenschaftliche Untersuchungen zur Sektorkopplung*)
- **Acronym**: SciGRID_gas (Scientific GRID gas)
- **Funding period**: January 2018 - December 2020
- **Funding agency**: Federal Ministry for Economic Affairs and Energy (*Bundesministerium für Wirtschaft und Energie*), Germany
- **Funding code**: Funding Code: 03ET4063
- **Project partner**: DLR Institute for Networked Energy Systems

1.2 Background

As of today, only limited data of the facilities of the European Gas Transmission Networks is publicly available, even for non-commercial research and related purposes. The lack of such data renders attempts to verify, compare and validate high resolution energy system models difficult, if not impossible. The main reason for such sparse gas facility

data is often the unwillingness of transmission system operators (TSOs) to release such commercially sensitive data. Regulations by EU and other lawmakers are forcing the TSOs to release some data. However, such data is sparse, and too often not clearly understandable for non-commercial operators such as scientists.

Hence, details of the gas transmission network facilities and their properties are currently only integrated in in-house gas transmission models which are not publicly available. Thus, assumptions, simplifications and the degree of abstraction involved in such models are unknown and often undocumented. However, for scientific research those data sets and assumptions are needed, and consequently the learning curve in the construction of public available network models is rather low. In addition, the commercially sensitivity also hampers any (scientific) discussion on the underlying modelling approaches, procedures and simulation optimization results. At the same time, the outputs of energy system models take an important role in the decision making process concerning future sustainable technologies and energy strategies. Recent examples of such strategies are the ones under debate and discussion for the Energiewende [Energiewende] in Germany.

In this framework, the SciGRID_gas project initiated by the research centre DLR Institute of Networked Energy Systems in Oldenburg aims at building an open source model of the European Gas Transmission network. Releasing SciGRID_gas as open-source is an attempt to make reliable data on the gas transmission network available. Appropriate (open) licenses attached to gas transmission network data ensures that established models and their assumptions can be published, discussed and validated in a well-defined and self-consistent manner. In addition to the gas transmission network data, the Python software developed for building the model SciGRID_gas are published under the GPLv3 license.

The main purpose of the SciGRID_gas project is therefore to open the door to new gas transmission network models and innovative ideas in energy system modelling by providing freely available and well-documented data on the European gas transmission network.

The input data itself is based on data available from openstreetmap.org (OSM) under the Open Database License (ODbL) as well as Non-OSM data gathered from different sources, such as Wikipedia pages, fact sheets from TSOs or even newspaper articles.

The main workload of this project is to:

- retrieve the OSM and Non-OSM data sets for the gas infrastructure
- merge all available data sets
- build a gas transmission component data set
- generate missing data using heuristic methods
- remove all gas facilities, that are not connected to pipelines.

The first step of the project was to collate a Non-OSM data set by searching the web for metadata that will be useful for the project. This included information such as pipelines, compressors, LNG terminals, and their attributes such as diameters, capacities, etc.

In addition, Python code was written to download OSM data from the openstreetmap.org data base. The data is filtered for natural gas transmission pipelines. If needed, other key elements of the gas transmission network could also be included in future.

After the first steps have been undertaken all data will be combined to generate a first gas network data sets. The release of this simplified network data set is set for early 2020.

This will be followed by combining further data sets in a manner, that the resulting networks will become more complete especially with regards to element attributes.

This multi-stage release will allow us to easily and effectively incorporate feedback from potential users during the lifetime of the project. It is anticipated, that additional releases of the network data will occur every two to three months. Those releases can be downloaded from the SciGRID_gas webpage with documentation, and can be seen as a snapshot of the current research project state.

The final network and underlying Python code will be released around December 2020 via a git-repository.

Further information on the project can be found on the SciGRID_gas web page: https://www.gas.scigrid.de/pages/imprint.html.

Please check out the section "Archive" for the latest content releases!!

The web page is maintained throughout the project lifetime, and will also contain information on:

- General project information
- Contact details
- Presentations
- Bug/data fixes.

As part of the SciGRID_gas webpage, one can also sign up to the SciGRID_gas newsletter by sending an email to

news.gas-subscribe@scigrid.de

1.3 Project goal

The overall goals of the SciGRID_gas project are:

- **Data output**: Creation of customisable gas transmission network data sets.
- **Open source**: Any one can download the data, make changes to it, pass it on to others, or even use it in commercial projects, as long as the SciGRID_gas project is mentioned as the original source of the data (CC by).
- **Application**: The outcome of the project can be used for a variety of scientific applications (e.g. sector coupling, entry-exit models, etc.).
- **Transparency**: The Python code, the documentation and the data (that can be passed on under copyright licences) is supplied.
- **Extendability**: Every user can extend the software code to their needs. However, we would encourage users to update and maintain the original git-repository and documentation for others.
- **Feedback**: Through constant data releases, it is hoped that the output data set will improve in quality and quantity by constantly incorporating feedback from the research community.

1.4 Document overview

This is an overview of the SciGRID_gas documentation, as this will help the user to better understand the overall project, its aims and the steps that were taken to obtain/model the data set.

SciGRID_gas has been coded in Python, and hence, with that came the overall data structure that was selected for the project. As this is the most fundamental aspect for anyone wanting to use the data and the code, it is described first. Under **Data Structure** we go into the definitions of the terms *Components*, *Elements*, and *Attributes*. We also give an overview on the internal workings of the SciGRID_gas source code.

A fundamental building block for the SciGRID_gas project is the data itself. Overall, we have classified the data into two groups: OSM and non-OSM data. The chapter **Data Sources** contains background information on the OSM and non-OSM data that is being used throughout the SciGRID_gas project. Here, each individual data source that was known to the SciGRID_gas project, is described in detail. Information is supplied such as how the data was collated or downloaded and how it was implemented into the SciGRID_gas code and data sets. In addition, an overview of the extent of the data will also be given for each data source, e.g. the number of elements or the list of attributes that the data contains.

As it is important to share the data at an early stage, for reasons of data improvement and transparency, the chapter **Data tools** describes the different methods that have been implemented into the SciGRID_gas project, how the gas transmission data set can be saved to different types of files and hence be shared with others. In addition other tools, such as visualization tools will also be introduced.

The raw OSM and non-OSM data is "incomplete", even if all available data sources are being joined to a single gas transmission network. This results in missing elements and missing attribute values for a large number of elements. Hence, the chapter **Heuristic Methods** will describe the different methods that have been implemented to "create" further elements and additional attribute values.

It is anticipated that the end-user might want to generate different gas transmission network data sets than the ones generated here. In addition, further data sources, component elements, attribute, or attribute values might be found. The user needs to be able to generate the gas transmission network data set on their own. This is why the chapter **Sample gas network data set generation** is a more hands-on chapter in which the generation of a gas network data set is being demonstrated. This enables the user to generate gas network data sets for different modelling purposes.

Prior to being able to generate the gas network data sets, one needs to install Python together with some libraries. The chapter **Installation** describes the individual steps the user needs to carry out if they want to use the SciGRID_gas code. Further, the manual download of some - sometimes copyright protected - data set might be required (e.g. GB shapefile).

The document also contains the chapter **Appendix** that contains sub-sections such as *Glossary*, *References*, etc.

CHAPTER

TWO

DATA STRUCTURE

A well designed and documented data structure is fundamental in any large scale project. Good data structure in combination with tools, based on algorithms, improve the performance of any project output.

This structure needs to represent the gas flow facilities as good as possible, hence it needs to include components such as pipelines, compressors, etc. A finite number of components have been identified, that are required as building blocks of a gas network. In addition each component will contain attributes, such as pipeline diameter, maximal operating pressure, maximal capacity, number of turbines etc.

It is anticipated, that the adopted data structure can be implemented in different types of gas flow models and will be used by the research community for topics such as sector coupling or identifying gas transmission bottlenecks.

Within the SciGRID_gas project, the structure of the data model is part of classes defined within the Python code. Alterations may occur over the duration of the project, but it is envisaged, that those will be small, and that compatibility will be assured.

The goal of this section is to describe in details the data structure that has been adopted and implemented into the Python code. This will be important in understanding other aspects of this document, such as exporting the data into CSV files or generating missing values.

Prior to the description of the data structure, the overall pathway of the data flow within the SciGRID_gas project will be explained, as it is believed, that such overview will help the reader.

2.1 Data flow pathway

It is believed that a generic description of the pathway from raw data to the final SciGRID_gas output data set would be of advantage to potential users. Individual sections will explain in greater detail data sources and how each of those pathway steps has been implemented into SciGRID_gas. Here, a more general description of the data flow pathway is given, where the pathway needs to be read from top to bottom.

The overall data flow pathway is being depicted in Fig. 2.1.

The SciGRID_gas project starts off with raw data, that can be found on the www. This is symbolised by the different entities, e.g. “GSE”, “GIE”, and “EntsoG” in the “Gas data in the www” block. The pathway from the raw data to the final SciGRID_gas data set output can be broken up into 5 blocks. They will be outlined here:

1) Loading the data: So the first step is to download the data from the www into the SciGRID_gas project environment. Different data specific methods needed to be implemented to download the data. E.g. the “LKD” data set is a shape file that could be downloaded, whereas the “GIS” data set consisted of an Excel book and was additionally accessible through an API, and the “EntsoG” data was only accessible through an API. Hence for each data set a different routine needed to be developed to access the data. However, other data sets needed to be exploited as well, such as Wikipedia or press releases. They are of quite different nature, compared with the other data sources, as the information important for the SciGRID_gas project is “hidden” in normal text and tables. Hence, for those data sources

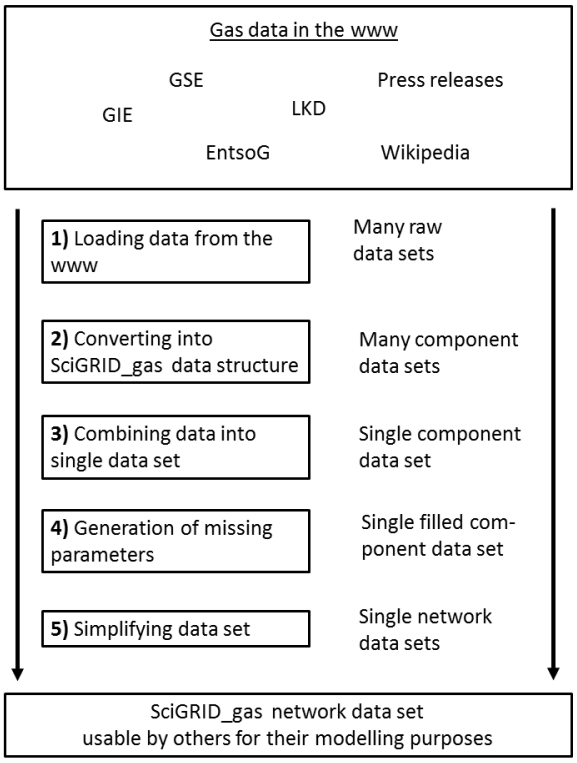


Fig. 2.1: Pathway overview of the data from raw data sources to the SciGRID_gas network data set output.

no Python code could be written, and the data was gathered manually, by reading numerous web pages and collating the data to CSV files.

2) Converting into SciGRID_gas data structure: In the next step, the data from all those different sources needs to be converted into the same data structure. The data structure itself will be described in the second part of this chapter. The conversion process is fundamental, as that allows for easier merging of the different data sets. In addition, from a programmers point of view, creating functions that can be applied to one data structure is easier, than having to write lots of slightly different routines for slightly different data sets. The outcome of this process are several component data sets of the same type. As the name states, this is a data set that consists out of components, meaning they do not need to be connected physically, e.g. compressors do not need to be connected to pipelines, yet.

3) Combining data into a single data set: Up to now, the data sets contain information on the gas facilities. E.g. the "GSE" data set contains information on the gas storages in Europe, whereas the LKD data set contains information on German facilities only. Hence, those different data sets need to be combined, carried out during the this step. This results in a single component data set, containing the data from all raw data sets, merged.

4) Generation of missing parameters: The resulting component data set consists of a lot of information, such as pipelines, compressors, and other components. Those components have associated attribute values, such as pipeline diameter or compressor power. However not all of the original raw data sets contained all required values for all entries, hence this data set also contains a lot of missing values. This step here deals with the generation of the missing values. This can be done through heuristic methods or statistical approaches.

5) Simplifying the data set: The last step is the conversion from the component data set to a network data set. The fundamental difference is that all components are connected with each other via pipelines, and that there are no islands.

The final SciGRID_gas network data set can be stored in different types of files, such as CSV, pickle and geoJASON, so that it can be used in gas flow models.

2.2 Data structure description

This section contains information about the SciGRID_gas data structure, the format, and the code that can be used to import publicly available data into the project, so that it can be used in subsequent steps. Paramount for an understanding of the data structure is a good understanding of the terminology used throughout this section and the document in general. Hence, terminology will be introduced in the following sub-section.

2.2.1 Terminology

Throughout this document certain terms will be used, which will be described below and summarized as a picture in Fig. 2.2.

Gas transmission network

The term "gas transmission network" describes the physical gas transmission grid. This does not include the distribution of gas through gas distribution companies, but includes the long distance transmission of gas from producer countries to consumer countries, as carried out by the Transmission System Operators (TSO) [RefTSO].

Gas component data set

The term "gas component data set" is used for all raw data of objects/facilities that have been loaded using SciGRID_gas tools into a Python environment. Gas component data sets are used as input into our SciGRID project. Several data sources can be loaded as gas component data sets, and then combined into a single gas component data set. However, not all elements (e.g. compressors) must be connected to pipelines, hence such a data set is referred to as a "gas component data set", and the emphasis is on the term **component**.

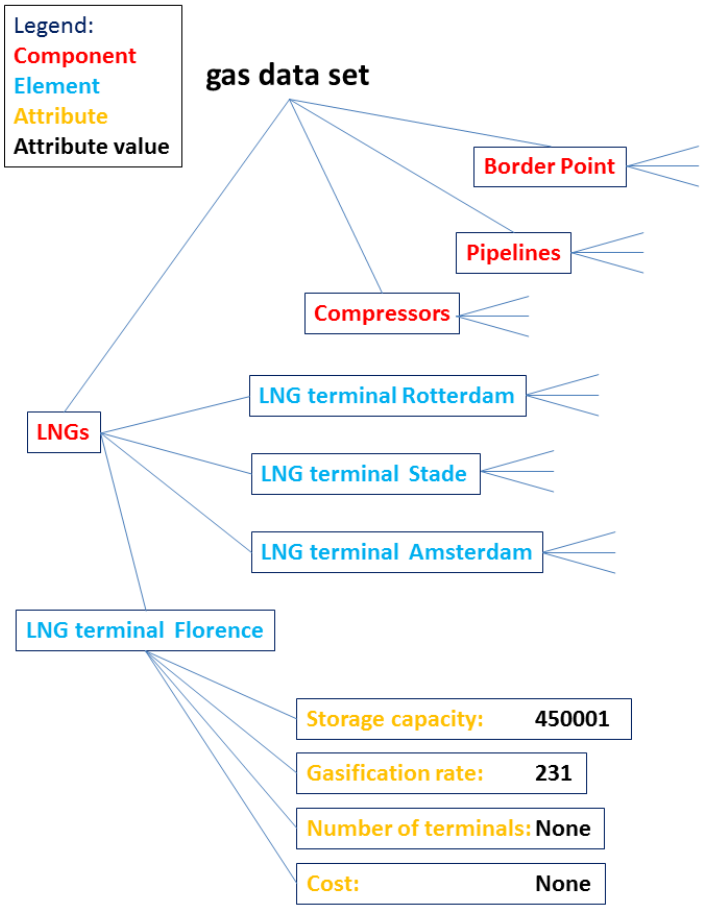


Fig. 2.2: Data structure for the SciGRID_gas data set

Gas network data set

A "gas component data set" can be converted into a "gas network data set", by connecting all non-pipeline elements to nodes and all nodes are connected to pipelines, and as part of the process all network islands have been removed, resulting in a single network. Therefore the network contains nodes and edges which are coherently connected, and all objects with the exception of pipelines are associated with nodes in this network, whereas pipelines are associated with edges. Hence the emphasis here is on the term **network**.

Component

There are several component types in a gas transmission network, such as compressors, LNG terminals, or pipelines. In Fig. 2.2 they are coloured red. Hence, whenever the word "component" is mentioned, it refers to one of these components. There are roughly a dozen different components that will form a gas network data set. They will be briefly explained below.

Element

The term "element" refers to individual facilities, e.g. the LNG Terminal in Rotterdam, or the compressor in Radeland. In Fig. 2.2 they are coloured blue. The first one is an element of the component LNG terminals, whereas the second one is an element of the component compressors. Hence many elements make up a component. However, all elements are referring to different facilities by default. This means in a single network, one can not have two elements of a component describing the same facility. The structure of elemens is described below.

Attribute

"Attribute" is a term that is being used for the individual labels of the values, that are associated with the elements. Examples for this term are gas "pipeline diameter", "maximum capacity", "max gas pipeline pressure", to name just a few and in Fig. 2.2 they are coloured yellow. Overall there will be several hundred attributes in the SciGRID_gas project. However, the same attributes can occur in more than one component, e.g. "max flow capacity" exist for pipelines and also for compressors. Throughout the project, we have tried to keep the units of such attributes the same, so that there is no unit conversion required.

Attribute value

Each attribute has a value, most likely a number or a string. In Fig. 2.2 they are coloured black. While boolean (True/False) is also allowed, more likely a "1" will stand for True and "0" for False. However, some attribute values might not be given in the data source, therefore a no value attribute value does exist. In the Python code it is termed "None".

The Fig. 2.2 depicts the relationships between the terms "gas data set", "component", "element", "attribute", and "attribute value". As can be seen, a single gas data set consists of several components, where each component contains several elements, and each element has several attributes, which each come with a value, where "None" stands for unknown value. The heuristic processes described in this document at a later stage will fill those "None" values with generated values.

Gas component types

A gas transmission network consists of different components, such as pipelines, compressors, etc. For the SciGRID_gas project a hand-full of components have been implemented, and will be described here briefly:

- **Nodes**: In a gas network, gas flows from one point to another point, which are given through their coordinates. All elements of all other components (such as compressor stations and power plants) have an associated node, which allows for the geo-referencing of each element. Overall the term "nodes" will be used throughout this document, as it aligns with graph theory aspects.
- **PipeLines**: *PipeLines* are one of the main components of the gas pipeline network. *PipeLines* allow for the transmission of the gas from one node to another. However each pipe is unique. They might have different diameter, capacity or max pressure. In addition, a single *PipeLine* can connect several nodes. Therefore it could go from "Radeland" to "Bottrub" and then follow on to "Frankfurt". However, for *PipeLines* it is not required that they connect more than 2 nodes, but they are allowed to.
- **PipeSegments**: *PipeSegments* are almost identical to *PipeLines*, however are only allowed to connect two nodes. Thus they have one start node and one end node, and are not passing via other nodes or other component elements in between, such as compressors or LNG terminals. Hence any pipeline can easily be converted to multiple pipe-segments.
- **Compressors**: *Compressor* stations are important: as the gas is travelling through the gas pipeline it loses pressure due to friction on the pipeline walls and other factors. This will reduce the throughput of the gas amount. Hence every so often (~ every 150 km), a compressor station is required, which increases the pressure of the gas, and hence allows the gas to flow through the gas pipeline. A gas compressor station contains several gas compressors units (turbines). Knowing the individual gas turbines is of an advantage, as those turbines can be combined in different ways, such as in series, or parallel, or combinations of those two options.
- **LNG terminals and LNG storages**: Some of the gas that is being used throughout Europe is supplied via ships to LNG terminals and LNG storage facilities. (From here onwards the acronym "*LNGs*" will be used instead "LNG terminals and LNG storage facilities".) As the transmission of gas would be extremely inefficient due to its volume, the gas state is changed to the liquid form (LNG gas), and then shipped. Ships arriving in Europe need special LNG terminals that can store LNG gas and subsequently re-gasify it. The storage and re-gasification of the gas are combined in the *LNGs* component and need to be part of any gas network for Europe.
- **Storages**: Part of the gas network will be gas storages. Gas storages are being used as gas pipeline capacities or gas production capacities might not be able to cover high demand periods, such as during the winter. Hence large gas storage units are being filled during the summer periods while the overall demand is low, and if capacities of net supply allow it. This gas is then used during the winter period, and can compensate for shortcomings of the gas network or gas supply. Almost every country has their own gas storage units, ranging from smaller units to compensate for daily fluctuations to larger units, which compensate seasonal fluctuations. For the SciGRID_gas model the larger seasonal storage units are of more importance than the smaller ones, as we are interested in the transmission gas pipeline network. However, any gas storage can be added and implemented into the gas network data set.
- **Consumers**: Part of the gas pipeline network is the knowledge of gas demand. Gas is added to the network at LNG terminals and European boundary cross border points. One type of users are the gas power plants. These can be added to the SciGRID_gas model, as this will specify local gas demand. In addition other consumers, such as city gas providers and large industries can also be added to the network data set.
- **Production**: These can be wells inside a country where gas is pumped out of the ground. Most of the gas used in Europe comes from outside of the EU, however there are several smaller gas production sites scattered through Europe.
- **BorderPoints**: BorderPoints are cross border points (between different countries), which are mostly for the purpose of accounting the gas flow. Most large gas pipelines have cross border stations, e.g. Ellund (lat/long: 54.80181, 9.289079) at the border between Germany and Denmark, with gas facilities on both sides of the border.
- **EntryPoints**: These are special border points, as they are at the borders of the European Union and will be the gas entry points for the SciGRID_gas model data set.

- **InterConnectionPoints**: These are points between gas transmission operators, and will be found mainly within Europe, in particular at country borders. However they can also be found within a single country, if there is more than one gas transmission operator.
- **ConnectionPoints**: These are the facilities which connect the gas transmission grid to the local gas distribution grid. Gas connection points are outside of the scope of the SciGRID_gas model.

Element structure

As described above, elements are describing individual facilities, such as compressors or LNG terminals. However, the overall structure of those elements is the same for all elements of all components. The overall structure of those elements is described in the following part:

- **id**: A string, that is the ID of the element, and must be unique.
- **name**: A string that is the name of the element, such as "Compressor Radeland".
- **source_id**: A list of strings, that are the sources of the element. As several elements from different sources could have been combined in a single element, one might need to know which are the original ids of the original sources.
- **node_id**: This is the ID of a geo-referenced point to which an element of the network is associated to. For a compressor, this will be just a single node_id, however for a gas pipeline, that starts at one point and finishes at a different point, there would be at least two node_ids.
- **lat**: This is the latitude value of an element. For pipelines, lat is a list of latitude values if known. The geo-referenced projection of the element that is being used in the SciGRID_gas project is: World Geodetic system 1984 (epsg:4326).
- **long**: The longitude, analogue to lat.
- **country_code**: This is a string indicating the official 2-digit country code (Alpha-2 code, see Appendix 9.6 for list of countries and their code). It represents the location of the element. As pipelines can pass through more than one country, the country code for pipes is the list of country codes of the countries the pipeline is passing through.
- **comment**: This is an arbitrary comment, that is associated with the element.
- **tags**: This dictionary is reserved for OpenStreetmap data. It contains all associated key:value-pairs of an OpenStreetMap item.

In addition, there are three further groups of attributes to each element. Throughout the SciGRID_gas project, they have been coded as "dictionaries". They are called:

- **param**
- **method**
- **uncertainty**.

The structure within each dictionary is the same, however their meaning is different. First of all the dictionary *param* (short for "parameter") contains a list of attributes and their values. This list of attributes will be different for each component. For the component *PipeLines* they might be pipeline diameter, max pipeline pressure, and max pipeline capacity. For the component *Compressors* they might be , such as number of turbines, overall turbine power, energy source of turbine and more. For a visual example of the param dictionary see Section 4.1.

So the other two attribute dictionaries are *method* and *uncertainty*. Each of those two dictionaries contain exactly the same list of attributes as the "param" dictionary. However their attribute values reflect the name of the dictionary. E.g. the attributes in the dictionary "method" contain the information on the method used to derive the attribute value that is stored in the param dictionary. Here methods of value generation can include heuristic methods names (in form of strings) that have been implemented in the SciGRID_gas project. However, if attribute values are not being generated

by the SciGRID_gas project, but originate from one of the input data sources, then the attribute values in the *method* dictionary is set to "raw".

Similar is the content of the *uncertainty* dictionary. It contains information on the uncertainty of the attributes from the *param* dictionary of that component. Again all attributes listed in the *param* dictionary are also present in the *uncertainty* dictionary. The attribute values here reflect the uncertainty of the attribute. Here, it is assumed, that attributes with a method of "raw" have an uncertainty of zero. Only for those attributes, which were generated during heuristic SciGRID_gas methods an uncertainty larger than zero will be specified.

2.3 Summary

The SciGRID_gas software is designed to construct a gas transmission network data set form different open source gas component data sets. The gas transmission data set needs to be available and stored in a precise and predefined way, which was described in this section. We have identified several *component* (-types) of a gas transmission network grid, like pipelines, compressor stations, LNG-terminals, etc. Each specific facility that falls under such a component is considered an *element* of that component. Each element is described by a list of *attributes* and correspondent *attribute values*.

CHAPTER

THREE

DATA SOURCES

Two thirds of the gas used in Europe is imported from non-EU states, and all gas required for the consumption needs to be distributed through the existing gas transmission pipelines in Europe. In the future gas consumption might rise, leading to additional pressure on the current infrastructure. In addition, gas facilities could play a vital role in reducing CO_2 emission, as excess electricity could be converted to gas, that could be stored and transmitted throughout Europe with the existing gas network. Hence, a reliable network data set for the European transmission network is essential. The data required for such models ranges from pipeline diameter, gas pressure within the pipeline, actual pipeline length, pipeline capacity, and underground storage volume to name just a few.

However, such data is the property of the transmission system operators (TSOs) and is therefore generally not freely available in the form and depth that is required for modelling purposes. The major reason for the difficulty of obtaining of such data is that most of the gas network infrastructure, namely pipelines, is buried underground. Thus a pipeline diameter is hard to estimate locally. In addition, almost all of the data is commercially sensitive.

However, there is a public drive to gather such data and subsequently make it available. The major platform through which this is occurring is the Open Street Map database [OSM]. OSM is a geo-referenced database through which people can supply geo-referenced information on all man-made and natural structures, ranging from mountains to buildings. To achieve this, people throughout the world wander the globe and geo-reference everything that they can find. This also includes gas-pipeline markers, compressor stations or LNG terminals. However, the major problem remains that one can not measure or estimate the diameter of the underground pipelines, or the number and size of the compressor turbines, as compressors are within buildings, which are fenced off. Hence such information is hardly supplied to the OSM platform.

Nevertheless, some data is made available by gas transmission network operators, through different channels. E.g. information on the size and number of compressors could be made public through a press release, as part of a refurbishment. An example is given below (https://www.maz-online.de/Lokales/Teltow-Flaeming/Neue-Verdichterstation-entsteht-in-Radeland):

> "Die Eugal-Pipeline dient dazu, Gas aus der neuen Ostseepipeline Nord Stream 2 bis zur tschechischen Grenze zu leiten. 275 Kilometer von ihr verlaufen in Brandenburg. Grundsätzlich soll die neue Leitung parallel zur bestehenden Opal-Pipeline gebaut werden."

In addition some information can be found on company web pages, (https://www.open-grid-europe.com/cps/rde/SID-752BB6B5-E0A975F2/oge-internet-preview/hs.xsl/NewsDetail.htm?rdeLocaleAttr=en&newsId=50190C3B-E14F-4685-9E64-E40EEAB57A28):

> Open Grid Europe (OGE) is investing roughly EUR 150 million at its compressor station in Werne to improve the security and flexibility of energy supply for North Rhine-Westphalia and Germany. The upgrade of the station, which is one of the hubs of the pipeline network, will allow gas flows to be switched (reversed) from north to south and south to north. In addition, OGE is preparing the station for the upcoming transition from L- to H-gas. Through this fitness programme, the station's transmission capacity will increase by about 500,000 to 6.5 million m^3/h, which is equivalent to the annual consumption of more than 2,100 single-family homes. The project, which is due for completion at the end of 2018, is fully on track."

The data available can be separated into two different groups:

- OSM data: Data can be found in the OSM data base. OSM data is well geo-referenced, but contains little meta-information (information on the facility attributes, such as pipeline diameter or pipeline capacity). OSM data is very helpful to obtain accurate routes of pipelines.
- Non-OSM data: Non-OSM data have in general lower geographical accuracy but contain a lot of meta-information. Unfortunately, such information is only known for a few facilities. One exception to this rule are shapefiles from TSOs. They are rare, but well geo-referenced. However, the resolution of the meta information can vary from TSO to TSO.

One of the main challenges for SciGRID_gas is that, gas transmission data is incomplete and accumulated from different sources. Also such different sources can have different properties for one and the same facility.

The following section will describe the OSM and non-OSM data in more detail.

3.1 Non-OSM data

Non-OSM data includes data from internet research, TSO press releases, TSO transparency platform, TSO public data, national open-source gas network data sets[1], etc.

Some of the TSO information had to be made available due to EU-regulations. Other information has been made public as part of a company's self presentation and advertisement. The information used by the SciGRID_gas project focuses on:

- the quality of the data
- the format of the data
- the level of representation of the data
- and the copyright restrictions on the data.

In addition, each data source is unique. Source specific tools need to be developed, so that all data sources can be made accessible for the SciGRID_gas project in the format as described in Chapter 2.

A significant portion of the project was spent on finding non-OSM data sets . Further data sources might be available, but unknown to the authors. If the authors are made aware of additional sources, the project will try to incorporate those, as this would only increase the depth of the data available and increase the applicability of the gas network data set and model.

Non-OSM data sources are very specific, addressing only certain aspects of the entire gas infrastructure. E.g. the GIE[GIE_MainRef] data set supplies information on the daily gas flow in and out of gas storages in LNG terminals. However, they fall short on specifying the fundamental information of the actual physical location. Other data sets, such as the LKD[LKD_MainRef] data set is quite detailed in respect of pipelines, compressors and consumptions, however only available for Germany.

Hence, the main task is to look closely at each data source, distil which data attribute values can be used, how it can be downloaded and incorporated into our SciGRID_gas model, and identify the copyright restrictions on the data source.

Due to copyright regulations, there are roughly two groups of data:

- Non copyright restrictive data (N-CRRD): here the copyright does not restrict the download, use and distribution of the data.
- Copyright restrictive data (CRRD): here the data can be downloaded and used internally, but not re-distributed to others.

[1] An entire gas network data set is only available from the UK, see https://www.nationalgridgas.com/land-and-assets/network-route-maps'.

The following is a list of the data sources that will be used throughout the project and an indication into which group of copyright restriction they fall:

- **OSM** (https://www.openstreetmap.org) (N-CRRD)
- **GB** (https://www.nationalgridgas.com/land-and-assets/network-route-maps) (CRRD)
- **NO** (https://www.npd.no/en/about-us/information-services/available-data/map-services/) (N-CRRD)
- **LKD** (https://tu-dresden.de/bu/wirtschaft/ee2/forschung/projekte/lkd-eu) (N-CRRD)
- **ENTSOG** (https://transparency.entsog.eu/) (CRRD)
- **GIE** (https://www.gie.eu/) (N-CRRD)
- **GSE** (https://www.gie.eu/index.php/gie-publications/databases/storage-database) (N-CRRD)
- **IGU** (https://www.igu.org/) (CRRD)
- **GasLib** (http://gaslib.zib.de/) (N-CRRD)
- **InternetData** (see Appendix 9.4) (N-CRRD).

Each data set and source comes with a different copyright regulation. The copyright can be rather non-restrictive (e.g. InternetData) or can be restrictive (IGU). It is attempted to use only freely available data, so that such data can be re-distributed. In more restrictive data cases (IGU, GB), it is not allowed to download the data and distribute it to others. However, it is allowed to let other potential users know of the location of such data and supply them with tools, that allow them to carry out the same data download and subsequent incorporation of the data into a gas network data set.

Note:

In case that other users are aware of other data sources, that might be useful to this project, please get in touch and supply us with a brief description of the data and the location of such data, so that additional tools can be developed to incorporate the data in this project. Please use the following email address: developers.gas(at)scigrid.de

3.2 InternetData

This section contains information on the content and nature of the so called **InternetData**, how this data was generated, its format, and the tools that have been designed to import the InternetData into the SciGRID_gas project.

The InternetData data set is a special data set, as it was collated from many www sources and the information has been collated into CSV files, as was briefly mentioned in Section 2.1. Please note, that throughout the project, the separator within CSV files will need to be ";". This section here will give an overview on the InternetData, its components and how the data is stored in InternetData specific CSV files. Further the importation of the data into Python will be described. Hence, based on the data flow pathway from Figure Fig. 2.1, this section here will explain in more detail the blocks "1" and "2".

Prior to the description of those processes, a general overview of the InternetData data set is given first, so that the reader gets a better understanding of the size and depth of the data.

3.2.1 Overview of the InternetData data set

The InternetData data set contains geographical and meta information on gas facilities that were found through Internet searches. The data originated from www pages, such as Wikipedia, gas transmission system operators, fact sheets and press releases and more. Hence most of the data had to be extracted manually out of text pages. To make this data available throughout the project, the data is being stored in CSV files. This also allows others to add additional

properties and values to the InternetData data set at any stage. Tools have been written to load the InternetData from those CSV files and make them accessible throughout the project[1].

The Table 3.1 summarises the number of elements for each component that has been found so far. However, this does not imply, that there is no missing data. In contrary, this data set comes with a lot of missing data:

Table 3.1: InternetData component summary.

Component Name	Count
BorderPionts	117
Compressors	237
ConnectionPoints	0
Consumers	0
EntryPoints	37
InterConnectionPoints	117
LNGs	32
Nodes	740
PipeSegments	532
Production	0
Storages	197

In addition, a map (see Fig. 3.1) visualizes these components for Europe in the figure below.

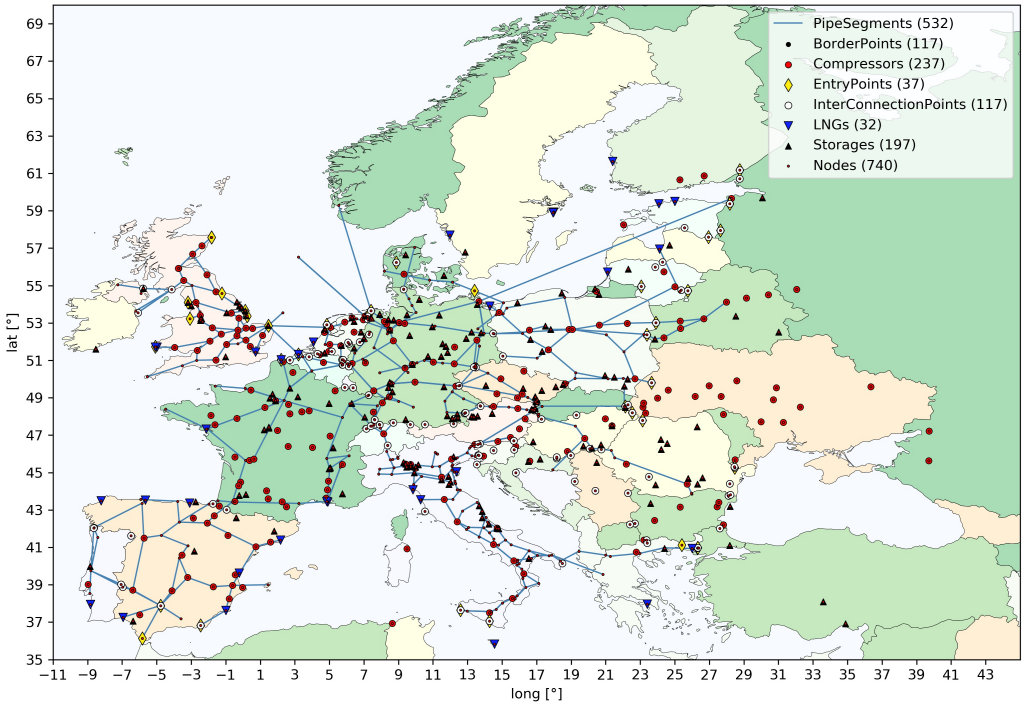
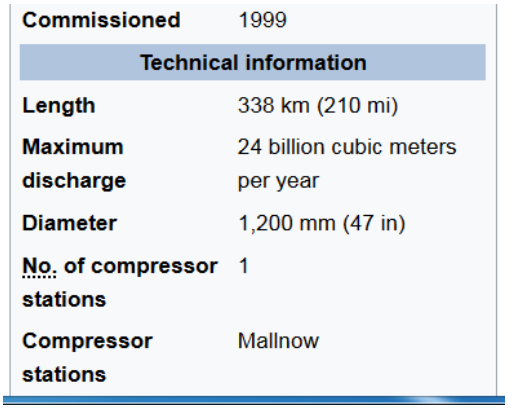


Fig. 3.1: Map of the InternetData data set. The legend contains the number of elements for each component.

[1] These tools will be made available during an upcoming release, where the InternetData data set will be jointly released with the GIE data set.

3.2.2 Origin of the data

As has been stated before, the resulting InternetData data set originated from text sources found on the www. Here. for the pipeline JAGAL an example from a Wikipedia page is given: [https://en.wikipedia.org/wiki/JAGAL]



Commissioned	1999
Technical information	
Length	338 km (210 mi)
Maximum discharge	24 billion cubic meters per year
Diameter	1,200 mm (47 in)
No. of compressor stations	1
Compressor stations	Mallnow

Fig. 3.2: Screenshot of part of the Wikipedia page for the pipeline JAGAL.

As one can see, some information is given such as location name of the compressor (Mallnow), total pipeline length (338 km), pipeline diameter (1200 mm) and maximum pipeline capacity (24 billion m^3a^{-1}). This is the information that is manually extracted from such pages and put into the CSV files. Other sources of such data are: gas facility operator press releases, fact sheets and other documents, newspaper articles, federal departmental web pages etc.

To collate the data in an orderly manner, a system of CSV files has been created and will be described below.

3.2.3 InternetData CSV file description

Each component of the InternetData data set is represented by two CSV files, a **location file** and a **metadata file**. The file structure is unique to the InternetData data set, and not to be mixed up with the CVS file structure as described in Chapter 4. With the InternetData CSV files, the user has the option of adding additional columns and rows and amend values, as long as the new data is of the same structure as described here. This might be desirable, as new data might appear. Below the location and metadata files are being explained.

Both file types have three heading lines. These are:

- Label for each column, e.g. “id” or “comment”. The entries of those fields are being used throughout the tools of the project for the InternetData data set, meaning they become variable names, and hence might reappear in the output files and are being used within the Python code. Hence do not change existing labels and assure, that each label is unique within eaah file. It is advised to incorporate the units of the attribute into the label name.
- Type for each column, a string indicating the column data type. Options are “TEXT”, “INT “and “REAL”. The correct type setting is important, as columns are imported differently depending on the type.
- Unit of the data for each column. A string in square brackets “[]” contains the unit of the data. This only applies to values and dates. This has been implemented for the user as a help so that attribute values are added in correct units. However, it will be ignored during the lading process.

Before diving into the location and metadata files, the “Loc_Nodes.csv” file for the component *Nodes* is being introduced, the only component, that is being described by only one file.

Loc_Nodes.csv file

This is a unique file, and contains information on the nodes of the InternetData data set. Nodes are such entities, to which and from where pipelines can run, or to which other facilities can be associated to. Nodes supply information on a location including its name, its latitude and longitude, and the country in which they are located. Additionally, they supply information on how exact is the location of the node could be determined. The nodes component data is supplied to the SciGRID_gas data model only via a single CSV file, and contains the following columns:

- **id**: This is a unique id of a node of type string. Most likely this will be the name of the location. White spaces are allowed in this string.
- **comment**: Here the user can place additional information on the location node.
- **country**: Here the user needs to write the 2 letter abbreviation of the country, in which this node is located (see Table 9.3 for a list of country codes used).
- **lat**: This is a number of the best estimate of the latitude of the location. Best lat value (and long value) were attempted to be generated by using metadata of the facility node and satellite maps. Using the satellite data, address information etc., it was tried to visually find the facility of the node.
- **long**: This is the corresponding best estimate of the longitude of the location derived during the same process as described under **Lat**..
- **exact**: This is a number in the range of 1 to 5, indicating how accurate the lat/longs were supplied for each node. Options are as follow:
  - "**1**": The exact location of this node is known, as one was able to verify the facility through satellite data.
  - "**2**": Here the lat/long is not known exactly, however one assumes that the location is within a small region (e.g. Krummhoern), hence not being much larger than $10km$.
  - "**3**": Here so little is known about the exact location, and one only knows, that the location is within a large region (e.g. Hamburg). Hence the actual location could be out by $10km$ or more but less than $100km$.
  - "**4**": Here so little is known about the exact location, and one only knows, that the location is within a state (e.g. Niedersachsen). Hence the actual location could be out by $100km$ or more but less than $1000km$.
  - "**5**": Here so little is known about the exact location, and one only knows, that the location is within a country (e.g. Ukraine). Hence the actual location could be out by $1000km$ or more.

All other components need two files, the location file and the metadata file, which will be described next.

Location file

The location files mainly contain information on the location of the gas facilities, hence these files only contain the following three columns:

- **id**: This is a unique identifier, which can be a string or a number, but will always be treated as a string.
- **comment**: Here the user can store additional information for this element.
- **node_id**: This links the facility to a node. Here the "node_id" value of the location file is equal to an "id" in the "Loc_Nodes.csv" file. This is depicted in the Figure Fig. 3.3 for the component *Compressors* as an example.

It should be clear and is required, that each element within a data set has to be unique. With this it is meant that each facility is only allowed to be in a component location file once. However, two different elements in the same component location file could have the same associated node_id, as two separate facilities of a component could be located at the same location/node_id.

Files of this type start with the letters "Loc" followed by an underscore and then by the name of the component, e.g. "Loc_Compressors.CSV" would contain all compressor facility location information.

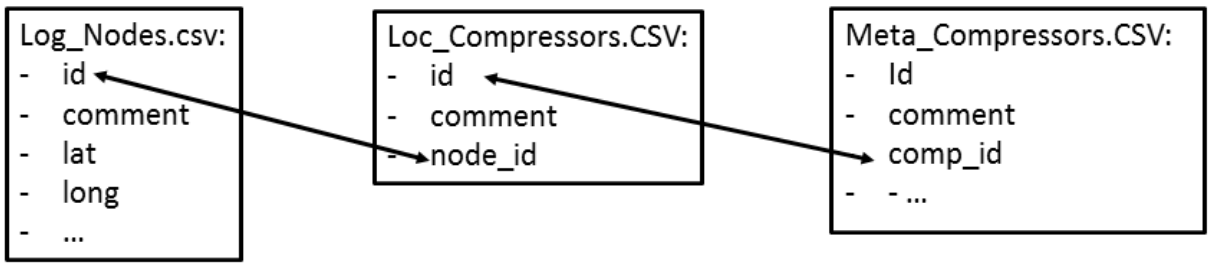


Fig. 3.3: Schematic diagram of the connection between nodes, location and metadata files, here for the example of the compressor components.

Metadata file

It is important to know the locations of the facilities, and that has been achieved with the location files. However, values such as pipeline diameter or LNG terminal storage capacity also need to be stored. For this the metadata files were introduced. These metadata files are component specific, but they are similar and are structured in the same way.

For all components of the InternetData data set, the first seven columns of the metadata files are pre-defined. A sample is given in Fig. 3.4:

	A	B	C	D	E	F	G	H	I	J	K	L	
1	id	comment	comp_id	gas_type	start_year	end_year	source	license	is_bothDirec	length	diameter_m	max_pressur	m
2	TEXT	TEXT	TEXT	TEXT	REAL	REAL	TEXT	TEXT	REAL	REAL	REAL	REAL	RE
3	[]	[]	[]	[]	[yyyy]	[yyyy]	[]	[]	[]	[km]	[mm]	[bar]	[M
40	Biccari_Candela		Biccari_Candela	H			http://www.	Snam			1219	75	
41	Blackrod_Elworth		Blackrod_Elworth										
42	Bordano_MoggioUdinesse_1		Bordano_MoggioUdinesse_1	H			http://www.	Snam			1219.2	75	
43	Bordano_MoggioUdinesse_2		Bordano_MoggioUdinesse_2	H			http://www.	Snam			1066.8	70	
44	Bosentino_Trento_1		Bosentino_Trento_1	H			http://www.	Snam			406.4	70	

Fig. 3.4: Sample Meta_PipePoint file.

Here we have a pipeline with the id "Bordano_MoggioUdinesse_1" and "Bordano_MoggioUdinesse_2" in line number 42 and 43 respectively. Both pipelines cover the distance from Bordano to Moggio Udinesse. However, they have different pipe diameter and different max pressure, as they are two parallel pipelines.

The columns of the above sample file are as follows:

- **id**: This is a *unique* identifier for each metadata element, e.g. for points, this is a unique name of the location of the point of interest, and needs to be supplied as a string. As points are located in different countries, however English being the most common second language, the 26 common letters from the English alphabet should be used. For rules, that are suggested of converting non-English location names please see Appendix 9.5.
- **comment**: Here the user can specify further information, e.g. the native spelling of the place, or further information to the place or facility. Even though this column is the second column, content from this column will not be transferred to the SciGRID_gas data base.
- **comp_id**: This is the id that is used to link the metadata entries with the component location entries, as described in Fig. 3.3.
- **start_year**: This is a year number in format [YYYY], indicating from when onwards the supplied metadata is valid.
- **end_year**: This is a year number in format [YYYY], indicating up to which year the supplied metadata is valid.
- **source**: This is a string, containing the location of where the data/information was collected from.
- **license**: This is a string, containing the information on the license type of the metadata supplied.

Each component has component specific attributes, e.g. pipelines have pipeline diameter and maximum capacity, whereas LNG terminals have information on the re-gasification capacity and storage volume. Hence for each component their component specific attributes will be listed below.

Prior to the individual description of the metadata files, a fundamental question needs to be addressed: Why are there separate location and meta files for each component? Why are location and metadata files not combined as one file per component? The answer is that the above setup allows for multiple entries of meta information per facility. For the same facility different and even "conflicting" information can be supplied, and might need to be supplied as well, as facility attributes might change over time, e.g. a compressor turbines might get upgraded, and have its capacity increased. This needs to be captured, and is done in a simple way as depicted in Fig. 3.5 below:

	A	B	C	D	E	M	N	
1	id	comment	comp_id	start_year	end_year	max_power_	num_turb	turbine_
2	TEXT	TEXT	TEXT	INT	INT	REAL	INT	INT
3	[]	[]	[]	[YYYY]	[YYYY]	[MW]	[num]	[boolea
11	Alphen		Alphen					
12	Alrewas		Alrewas	1970	2005	21.6	2	
13	Alrewas		Alrewas	2006		50	4	
14	Ananiev		Ananiev					

Fig. 3.5: Example of multi entries for the same facility, applicable to different time periods, indicated through the start_year and end_year values of the two entries for Alrewas.

The example is given for the compressor facility with id "Alrewas", in line with number "12" and "13". As can be seen, they have different "start_year" and "end_year" values, with the first one having a start year of 1970 and an end year of 2005. Hence this means, that the first entry for "Alrewas" covers the time from 1970 to 2005, whereas the second entry for "Alrewas" covers the time period of 2006 and onwards. This does not mean that the SciGRID_gas project can supply gas facility properties that change during a model run, by supplying date-stamp dependent gas facility properties. However, the user can generate SciGRID_gas data sets for different points in time. How this can be carried out will be explained in the subsection 3.2.5.

Files of this type start with the letters "Meta", followed by an underscore and then by the name of the component, e.g. "Meta_Compressors.CSV" would contain all compressor facility metadata information.

Below for each component the additional component specific attributes will be introduced in the next sub-sections.

Compressor CSV meta file

The compressor metafile ("Meta_Compressors.CSV") contains all the metadata for all known compressor stations.

In addition to the seven mandatory columns introduced above, the following columns are currently implemented, and contain the following data:

- **operator_name**: This is a string, containing the name of the operator of the compressor station.
- **pipes_name**: This is a string containing the label of the pipeline that the compressor is connected to.
- **is_H_gas**: This is a boolean, indicating if the gas is of high calorific gas type ("1") or of low calorific gas ("0").
- **max_cap_M_m3_per_h**: This is a number, which is the overall capacity of gas that can be compressed by the compressor station. Values need to be supplied in units of $[Mm^3/h]$.
- **max_pressure_bar**: This is a number, which is the maximum pressure that the gas can be compressed to. Values need to be supplied in units of $[bar]$.
- **max_power_MW**: This is a number, which is the sum of the power of all compressor units that are installed at the compressor station. Values need to be supplied in units of $[MW]$.

- **num_turb**: This is the number of compressor turbines installed at the compressor facility. This number is including the reserve turbine unit.
- **turbine_fuel_isGas_1**: This is a boolean, indicating if the turbine is powered by gas ("1"), or by electric ("0").
- **turbine_type_1**: This is a string containing additional information on the type of turbine unit, e.g. name of the turbine.
- **turbine_power_1**: This is a number, indicating the power of the turbine unit. The value needs to be supplied in units of $[MW]$.
- **turbine_fuel_isGas_2**: This is the information for the second turbine unit. Same as for *turbine_fuel_isGas_1* applies. Currently up to 6 individual units can be stored in the database, hence the last digit in the identifier can be as large as 6.
- **turbine_type_2**: This is the information for the second turbine unit. Same as for *turbine_type_1* applies. Currently up to 6 individual units can be stored in the database, hence the last digit in the identifier can be as large as 6.
- **turbine_power_2**: This is the information for the second turbine unit. Same as for *turbine_power_1* applies.
- ...
- **turbine_power_6**: This is a number, indicating the power of the sixth turbine unit. The value needs to be supplied in units of $[MW]$.

LNG CSV meta file

The LNG terminal metafile ("Meta_LNGs.CSV") contains all the metadata for the LNG terminals.

Next to the above described first seven columns the following columns are currently implemented, and contain the following data:

- **storage_LNG_Mt**: This is a number, indicating the maximum amount of liquid gas, that can be stored, after having been brought in by ship. Values need to be supplied in units of $[MtLNG]$
- **max_cap_store2pipe_M_m3_per_a**: This is a number, indicating the maximum amount of gas, that can leave the LNG terminal. This gas is in gas phase. Values need to be supplied in units of $[Mm^3/a]$

BorderPoints CSV meta file

The metafile for border points ("Meta_BorderPoints.CSV") contains all the metadata for each border point.

Next to the above described first seven columns the following columns are currently implemented, and contain the following data:

- **pipe_name**: This is a string, the name of the pipe that is passing the border point.

EntryPoints CSV meta file

The metafile for entry points ("Meta_EntryPoints.CSV") contains all the metadata for entry points of gas pipelines into Europe.

Next to the above described first seven columns the following columns are currently implemented, and contain the following data: No additional data was found for the entry points.

InterConnectionPoints CSV meta file

The InterConnectionPoints metafile ("Meta_InterConnectionPoints.CSV") contains all the metadata for interconnection points between the different operators within Europe.

Next to the above described first seven columns the following columns are currently implemented, and contain the following data: No additional data was found for the interconnection points.

Storages CSV meta file

The metafile "Meta_Storages.CSV" contains all the metadata for gas storage within Europe.

Next to the above described first seven columns the following columns are currently implemented, and contain the following data:

- **store_type**: This is a string, indicating the type of storage, such as "Leeres Gas Feld" (empty gas field), "Salz Kaverne" (salt cavern) etc.
- **is_H_gas**: This is a boolean, that indicates if the gas is of high calorific nature ("1") or of low calorific nature ("0").
- **is_onShore**: this is a number, indicating if this gas store is on land or not. Options are "**1**" and the gas store is on land, whereas the second option "**0**" indicates, that the gas store is not on land, hence will be off shore.
- **operator_name**": String, containing the name of the operator.
- **max_workingGas_M_m3**: A number indicating the maximum amount of gas that can be stored and worked with in that gas field. Values need to be supplied in units of $[Mm^3]$.
- **max_workingGas_TWh**: A number indicating the maximum amount of gas that can be stored and worked with in that gas field. Values need to be supplied in units of $[TWh]$.
- **max_cap_store2pipe_M_m3_per_d**: A number indicating the maximum amount of gas that can move from the gas store into a gas pipe. Values need to be supplied in units of $[Mm^3d^{-1}]$.
- **max_cap_store2pipe_GWh_per_d**: A number indicating the maximum amount of gas that can move from the gas store into a gas pipe. Values need to be supplied in units of $[GWhd^{-1}]$.
- **max_cap_pipe2store_GWh_per_d**: A number indicating the maximum amount of gas that can move from the gas pipeline into a gas store. Values need to be supplied in units of $[GWhd^{-1}]$.
- **access_regime**: A string containing an access regime, however it is not known, but could be used for heuristic processes.

PipeLine CSV meta file

The metafile "Meta_PipePoints.CSV" contains all the metadata for gas storage within Europe.

Next to the above described first seven columns the following columns are currently implemented, and contain the following data:

- **is_bothDirection**: This is a boolean with value of '1' or '0'. If set to '1', then the gas pipeline can be operated in both directions, whereas if set to '0', then the gas can only flow from the start point to the end point. Hence, here the order of the point_lables in the pipes file is important.
- **length**: This is the overall length of the pipeline, and NOT of the segment. The value needs to be supplied in units of $[km]$.
- **diameter_mm**: This is the diameter of the pipe in units of $[mm]$.
- **max_pressure_bar**: This is the maximum pressure of the gas within the gas pipeline in units of $[bar]$.

- **max_cap_M_m3_per_d**: This is the maximum annual gas volume that the pipe can transmit in units of $[Gm3/d]$.
- **num_compressor**: This is the number of compressors along the pipeline.

There is one more file that needs describing: the file for the component pipelines. As pipelines go from one node to another node, a special file has been created and will be described next.

PipeLine location files

It is important to understand that a single pipe starts at a point, can flow via several other points, before finishing at its end point. E.g. the gas pipeline EUGAL starts in "Lubmin", from there is runs via the nodes "Kienbaum", "Radeland", "Weissack", and "Deutschneudorf", and finishes at the node "Gebirgsneudorf" (see Fig. 3.6). Why would one want to be that specific in respect of the pathway of the pipeline? For example, at Radeland, there is a gas compressor unit, where the gas pressure is being increased. At Kienbaum there is a connection with another gas pipelines (in this case the JAGAL gas pipeline). So the gas that is flowing in the pipeline from "Lubmin" is re-compressed at "Radeland", where there is a connection with the JAGAL pipeline at the "Kienbaum" node. For a gas data model, this is very important information. Hence the term "pipeline" refers to the pipeline from the start node to the end node, and would run from "Lubmin" to "Gebirgsneudorf" in the example above. The term "pipe-segment" is also used, and refers to a segment of the pipeline, that connects one node with another node, without any intermediate nodes. It is inherent, that no metadata will change over the path of a single pipeline, otherwise a new node should be added at the location where an attribute changes and the pipeline be split up into two pipe-segments.

Hence the implementation of the location information for the gas pipelines is given below, where the name of the pipeline file is "Loc_PipePoints.CSV". The file contains the following columns:

- **id**: This is a unique identifier of a pipeline, string or number.
- **name**: This is the name of the pipe. As described above, a pipeline can pass through several points, and hence several lines in the pipes file can contain the same pipes_name. Due to convention, a pipe contains at least a starting point and an end point, and hence would need to appear twice in the pipes table. The first occurrence of the pipes name marks the start of the pipeline, whereas the last occurrence of the pipe name marks the end point of the gas pipeline. It is required that all node names between the start and the end node are listed in the table in the order of the gas flow. This entry needs to be of type string. Further convention is that each entered pipe in the pipes TXT file contains only a single physical pipe. Hence, if a pipeline has two or more pipes running next to each other, than they need to be entered individually, and naming convention could be along the lines of 'Masera_Veruno_1' and 'Masera_Veruno_2'.
- **node_id**: Here the name of the point, where the pipe is connected to, is being supplied. Those values need to be given in the "Loc_Nodes.CSV" file.
- **meta_id**: As can be seen, the pipes table does not contain information on length, diameter, pressure, and so on. However the SciGRID_gas data model also needs this information. Hence the "meta_id" string is used to make the 'connection' with entries in the pipes metafile, called the "Meta_PipePoints.csv" file. This file will be described below.

An example is depicted in the example data set below (see Fig. 3.7). Here, the "EUGAL_1" pipeline starts at the node with label "Lubmin", then passes through "Kienbaum", next is "Radeland", and then "Weissack", followed by "Deutschneudorf" and ends in "Gebirgsneudorf", as all those points have the same value under the column "name". In addition, all of them have the same meta_id value. Further, there is a parallel gas pipeline, with the name "EUGAL_2" that runs from "Lubmin" via "Kienbau" and "Radeland" and ends at "Weissack", and also has the same meta_id value.

3.2.4 InternetData data density

Now that the structure of the InternetData has been described in detail above, the "data density" of data will be presented. This is important so that one can get a better understanding of the data. Here "data density" is defined

Fig. 3.6: Schematic pipeline (blue line) pathway of the Eugal gas pipeline from Lubmin, via Kienbaum, Radeland, Weissack, Deutschneudorf, and finishes in Gebirgsneudorf ((c) Google Map contributors).

	A	B	C	D
1	id	name	node_id	meta_id
2	REAL	TEXT	TEXT	TEXT
3	[]	[]	[]	[]
155	152	EUGAL_1	Lubmin	EUGAL_1
156	153	EUGAL_1	Kienbaum	EUGAL_1
157	154	EUGAL_1	Radeland	EUGAL_1
158	155	EUGAL_1	Weissack	EUGAL_1
159	156	EUGAL_1	Deutschneudorf	EUGAL_1
160	157	EUGAL_1	Gebirgsneudorf	EUGAL_1
161	158	EUGAL_2	Lubmin	EUGAL_1
162	159	EUGAL_2	Kienbaum	EUGAL_1
163	160	EUGAL_2	Radeland	EUGAL_1
164	161	EUGAL_2	Weissack	EUGAL_1

Gas_PipePoints

Fig. 3.7: Sample Loc_PipePoint file.

as follow: this is the ratio of the number of usable (not missing) attribute values over the number of elements of the component. E.g. supposedly the InternetData would have two LNG terminals. And one of the facilities has supplied a storage volume, whereas the other one does not. Hence, the data density would be 50% for the attribute storage volume. For each component the data density for the most relevant attributes will be given next.

PipeSegments

Overall there are 532 pipe-segments in the InternetData data set.

Table 3.2 summarizes the data densities for the most important pipe-segment attributes:

Table 3.2: InternetData *PipeSegments* data density

Attribute name	Data density [%]
diameter_mm	65
is_H_gas	90
is_bothDirection	12
length	100
max_cap_M_m3_per_d	20
max_pressure_bar	48
num_compressor	5

Compressors

Overall there are 237 compressor elements in the InternetData data set. The data densities for the most important attributes is given in Table 3.3 below:

Table 3.3: InternetData *Compressors* data density

Attribute name	Data density [%]
is_H_gas	100
max_cap_M_m3_per_d	7
max_power_MW	14
max_pressure_bar	7
num_turb	16
operator_name	11
pipe_name	11
turbine_power_1_MW	8
turbine_power_2_MW	8
turbine_power_3_MW	5
turbine_power_4_MW	1
turbine_power_5_MW	1
turbine_power_6_MW	0
turbine_fuel_isGas_1	15
turbine_fuel_isGas_2	15
turbine_fuel_isGas_3	10
turbine_fuel_isGas_4	4
turbine_fuel_isGas_5	1
turbine_fuel_isGas_6	0
turbine_type_1	10
turbine_type_2	10
turbine_type_3	7
turbine_type_4	3
turbine_type_5	1
turbine_type_6	0

Nodes

Overall there are 740 nodes. As described above, the information supplied is an "id", latitude and longitude values, the country code and a value indicating the accuracy of the node location. Hence, Table 3.4 summarizes the relative number of nodes that have the exact value of 1, 2,.. , 5, given in percent:

Table 3.4: Summary for the attribute "exact" of compoent *Nodes* of the InternetData data set.

Exact value	ration of data with exact value [%]
1	34
2	61
3	18
4	7
5	7

Storages

Overall there are 197 storage elements in the InternetData data set. The data densities for the most important attributes is given in Table 3.5 below:

Table 3.5: InternetData *Nodes* data density

Attribute name	Data density [%]
access_regime	89
is_H_gas	12
is_onShore	95
max_cap_pipe2store_M_m3_per_d	77
max_cap_store2pipe_M_m3_per_d	77
operator_name	99
max_workingGas_M_m3	80
source	94
store_type	94

BorderPoints

Overall there are 117 border point elements in the InternetData data set. The data densities for the most important attributes is given in Table 3.6 below:

Table 3.6: InternetData *BorderPoints* data density

Attribute name	Data density [%]
pipe_name	15

EntryPoints

Overall there are 37 entry points elements in the InternetData data set. This component does not contain any further attributes of interest.

InterConnectionPoints

Overall there are 117 Interconnection points elements in the InternetData data set. The data density for the most important attributes is given in Table 3.7 below:

Table 3.7: InternetData *InterConnectionPoints* data summary

Attribute name	Data density
pipe_name	15

LNGs

Overall there are 32 LNGs elements in the InternetData data set. The data densities for the most important attributes is given in Table 3.8 below:

Table 3.8: InternetData *LNGs* data density

Attribute name	Data density [%]
entsog_key	44
max_cap_store2pipe_M_m3_per_d	91
max_workingGas_M_m3	97

Overall one can see that a lot of data has been collated and is made available through the InternetData data set. However, as presented in the data density tables, a lot of attributes have low data density. Chapters later in this

documentation will demonstrate how missing values can be estimated, so that the generated SciGRID_gas data set has a data density of 100% for each attribute.

3.2.5 InternetData data import steps

This section contains information on how the InternetData load process has been implemented in the Python code. As was stated in a previous section, individual facilities can be supplied with several metadata entries. As part of the InternetData load process, the user can specify several filter values, e.g. only loading metadata for a given year or with a certain license, or combinations of those settings. The following input filter options are available:

- "**requeYear**": Here the user can specify a year, for which the gas data set shall be generated for. Hence only elements and metadata, that are applicable for the user specified required year will be selected.
- "**licenseType**": Here the user can specify that the metadata shall have at least a copyright of a given level (this has not been implemented yet).
- "**GasType**": Here the user can specify for which gas type the gas data set shall be generated, with options "H" and "L".

If any of the filters shall be applied, then they will need to be supplied to the function that reads the InternetData CSV files. Default values are "2010" for "requeYear" and "H" for "GasType". There is no default value for the attribute "license", hence as a default, the metadata will not be filtered based on the licenses.

The loading process steps are as follows:

- The location information for all components is read.
- All metadata for all components is being read.
- All node information is being read.
- All location information of each component element is "linked" to a node, so that each location component element now has its lat/long values.
- The filters are being applied to the metadata for each component.
- Metadata and location information for each component are being joined, only if there is metadata available for the elements of the components.
- All pipelines are being split up into the component *PipeSegments*, whereas the component *PipeLines* is being removed.
- For all *PipeSegments*, their length is re-calculated using associated lat/long values from above.
- For all *PipeSegments*, an average lat/long value is being calculated and added as attributes "lat_mean" and "long_mean". "lat_mean" is the average of the start and end node latitude values, and "long_mean" is the average of the start and end longitude values.
- Some attribute values are being converted based on units supplied through the attribute labels. E.g. for the LNG attribute "storage_LNG_Mt", a unit conversion from $LNGMt$ to Mm^3 (in gas form) is carried out by multiplying the input value by 1442.48 and the attribute name changes to "max_workingGas_M_m3". For a list of conversions for the InternetData please see Table 3.9 below, whereas a list of all unit conversions can be found in Appendix 9.2.
- For the elements of component *PipeSegments* the attribute 'meta_id' is being removed.

Table 3.9: List of InternetData attributes where unit were converted.

Component name	Old attribute name	New attribute name
LNGs	storage_LNG_Mt	max_workingGas_M_m3
LNGs	max_cap_store2pipe_M_m3_per_a	max_cap_store2pipe_M_m3_per_d
Compressors	max_cap_M_m3_per_h	max_cap_M_m3_per_d
Storages	max_cap_pipe2store_GWh_per_d	max_cap_pipe2store_M_m3_per_d
Storages	max_cap_store2pipe_GWh_per_d	max_cap_store2pipe_M_m3_per_d
Storages	max_workingGas_TWh	max_workingGas_M_m3

The above process describes the pathway how the information collated from the www and stored in CSV gets transformed into the SciGRID_gas data structure within Python.

A map of the gas pipes and gas facilities of the InternetData data set was given in Fig. 3.1.

3.2.6 Copyright and disclaimer for the InternetData

Copyright

Based on the legal framework, all of the InternetData was collated in such a way, that it has a copyright, that does not restrict us from making the data available to other users.

Hence the following applies to the InternetData data and this document:

A list of the sources used for the generation of the InternetData data set can be found in Appendix 9.4.

Disclaimer

The InternetData data set is supplied on a best-effort basis only, using available information as documented gathered from the Internet. While every effort is made to make sure the information is accurate and up-to-date, we do not accept any liability for any direct, indirect, or consequential loss or damage of any nature–however caused–which may be sustained as a result of reliance upon such information.

3.3 GIE data

GIE is a data set with vital meta-data within the SciGRID_gas project, where GIE stands for Gas Infrastructure Europe.

'Gas Infrastructure Europe (GIE) is an association representing the sole interest of the infrastructure industry in the natural gas business such as Transmission System Operators, Storage System Operators and LNG Terminal Operators. GIE has currently 68 members in 25 European countries.' (https://www.gie.eu/).

GIE is the umbrella organisation for the following three gas components:

- **Storage**: GSE - Gas Storage Europe representing the Storage System Operators (SSO)
- **LNG**: GLE - Gas LNG Europe representing the LNG Terminal Operators (TO)
- **Transmission**: GTE - Gas Transmission Europe representing the Transmission System Operators (TSO)

The storage and the LNG information can be retrieved through an API supplied by GIE. However, there is no further information on the gas transmission part.

The APIs for the gas storage and the LNG terminals are:

- Aggregated LNG Storage Inventory (ALSI) (https://alsi.gie.eu/api/data/)
- AGSI+ AGGREGATED GAS STORAGE INVENTORY (https://agsi.gie.eu/api/data/)

Documentation for the APIs can be found on the web under: https://agsi.gie.eu/GIE_API_documentation_v001.pdf.

3.3.1 Pre requirements for accessing the GIE data set

A private key is required for the GIE data set so that one can download data from the GIE API.

As stated in the documentation for the GIE_API:

> The API service is available to the public free of charge. Registration on the AGSI+ or ALSI website is mandatory for non-data providers to be able to use the API. Registration will result in a personal API key that is required within the API url. The only purpose of this registration is to enable us to assess and improve the performance of our systems where and if required (user count, user activity, most popular data set types). Your account information and settings can be updated (and cancelled) at any time after signing in. Your data will be stored and securely handles as long as your account remains active.

For this you will need to go to the following link: https://agsi.gie.eu/#/login where on the right hand side you will need to fill in the registration details.

Under “Access to:” please select “Both AGSI+ and ALSI”

After registration you will have access to your private key. Copy the key and past it into the following file:

/SciGRID_gas/Eingabe/GIE/GIE_PrivateKey.txt

This is your private key, hence do not share it with others.

Data manipulation of the GIE data set

Gas infrastructure providers are requested to publish certain gas flow information. This data is accessible via the GIE URLs, and contains a vast amount of meta-data for storages and LNG terminals throughout Europe. Whenever the data is downloaded from the GIE API, the data needs modification, so that it is conform with the SciGRID_gas data model. Several tools have been written to achieve this. The GIE specific tools are described below fpr **Storages** and **LNGs**.

Processes for retrieving the data from the GIE API

First of all, one could access some meta-data on the storages and LNG terminals through the following internet links:

- LNG: https://www.gie.eu/index.php/gie-publications/databases/lng-database
- Storages: https://www.gie.eu/index.php/gie-publications/databases/storage-database

These data sets come as Excel books and contain information such as name of facility, country, type of facility, and eic_code. Other information such as max hourly capacity, or LNG storage capacity is also given, however discarded due to copyright reasons.

LNG terminals The following information are used from the LNG table above:

- country
- type
- eic_code
- short_name
- name
- nameShort

This information (column heading and column data) needs to be placed into a CSV document. This file needs to be named "GIE_LNG.csv" and needs to be stored in the folder "/SciGRID_gas/Eingabe/GIE/".

Other additional fields from this table are:

- Region
- Status
- Investment
- Start-up year
- Type
- Operator short name
- Max. Hourly Cap. [$m^3(N)/hour$]
- Nom. Annual Cap. billion [$m^3(N)/year$]
- Possible additional Nom. Annual Cap. billion [$m^3(N)/year$]
- LNG storage capacity [m^3 LNG]
- Number of tanks
- Max. ship class size receivable [m^3 LNG]
- Number of jetties
- Min. sea depth alongside [m]
- Max. send out pressure [bar]
- TPA regime
- PCI list
- Operator long name

This list can change during the runtime of the project.

The EIC-code, facility code and country code is subsequently used to request time series information for each location from the GIE API.

The retrieved time series contain two useful values:

- the working LNG volume in the LNG storage tank
- the gas flow amount from the storage to the gas-pipeline, in units of *GWh/d* (see Table 3.10)

From the so created time series, one can determine the maximum working gas volume in the LNG storage tank in units of million LNG cubic meters. Also the maximum and medium gas flow from the storage to the gas pipeline is determined, in units of *GWh* per day.

Prior to estimating the maximum/medium/median value from the retrieved time series, the time series was quality assured. This was done by removing any outliers/spikes.

Table 3.10: GIE incorporated attributes

Field identifier	Description	Units	Example
status	E (estimated) C (confirmed) N (no data)	E / C / N	C
gasDayStartedOn	The start of the gas day reported upon	YYYY-MMDD	2015-11-02
lngInventory	The aggregated amount of LNG in the LNG tanks at end of the previous gas day	$10e+3m^3$ LNG (2 digits behind decimal point)	5373.25
sendOut	The aggregated gas flow out of the LNG facility within the gas day	GWh/d (1 digit behind decimal point)	976.5
dtmi	Declared Total Maximum Inventory	$10e+3m^3$ LNG (2 digits behind decimal point)	8898.99
dtrs	Declared Total Reference SendOut	GWh/d (1 digit behind decimal point)	6650.0
info	Service Announcement (if applicable)	url	https://alsi.gie.eu/#/news/184

The following information is incorporated into the SciGRID_gas data structure: name, max storage volume, max and medium gas flow volumes, facility code, country code, and EIC-code.

Subsequently, the LNG storage volume and flow are converted to their corresponding gas phase values. The final units of the measurements were $[Mm^3d^{-1}]$. No geo-coordinates are given for LNG terminals within the GIE data set. This information has been retrieved from the InternetData data set by a comparison of the name and the country code of the facility.

3.3.2 Storages

A meta-data set for the storages is available as Excel book and can be downloaded from the following URL: (https://www.gie.eu/maps_data/downloads/2018/Storage_DB_Dec2018.xlsx)

In this Excel book, the sheet "Storage DB" contains the following columns:

- Country: string indicating the country of the storage
- Concatenate: –missing description–
- Country Code: two letter acronym for country code
- Company code: number of company
- Facility code: code of the facility
- Operator: name of operator
- Facility/Location: name of facility location
- Status: status of storage unit (operational/under construction/planned)
- Investment: string indicating the investment (existing/expansion/new facility)
- Start-up year: year when started operation
- Type: storage type, e.g. depleted field, salt cavern,...
- Notes:
- onshore/offshore: either onshore of offshore location of the gas storage

- Working gas (technical) TWh: maximum working gas volume in units of [TWh]
- Working gas TPA TWh: maximum working gas volume under TPA in units of [TWh]
- Working gas no TPA TWh: maximum working gas volume not under TPA in units of [TWh]
- Withdrawal technical GWh/day: maximum withdrawal rate of gas in units of [GWh/d]
- Withdrawal TPA GWh/day: maximum withdrawal rate of gas under TPA in units of [GWh/d]
- Withdrawal no TPA GWh/day: maximum withdrawal rate of gas not under TPA in units of [GWh/d]
- Injection technical GWh/day: maximum injection rate of gas in units of [GWh/d]
- Injection TPA GWh/day: maximum injection rate of gas under TPA in units of [GWh/d]
- Injection no TPA GWh/day: maximum injection rate of gas not under TPA in units of [GWh/d]
- Access regime: access regime with two options "nTPA", "rTPA", and "No TPA"
- in EU28 number: string ("n" or "y") if part of the 28 EU members
- in EU28 SUM: string ("n" or "y") if part of the 28 EU members
- EU 28 filter: string ("NO" or "YES") if part of the 28 EU members.

A subset of the above data needs to be saved as column data into a CSV file, containing only the following parameters with their data:

- Country
- Type
- EIC Code
- Short Name
- Name
- nameShort

This file needs to be name "GIE_Storages.csv" and needs to be stored in the folder "/SciGRID_gas/Eingabe/GIE/".

These is subsequently used to get access to the time series of the storage data set, by using the facility code, the country and the EIC-code.

All time series consist of the daily "working gas volume", the "daily injection capacity" and the "daily withdraw capacity". Maximum values for each of those parameters are extracted from those time series.

However, as was carried out for the LNG data set, the same testing of the goodness of the data was carried out.

In addition, gas flow values were converted from [GWh/d] to [Mm^3d^{-1}].

Data overview of the GIE data set

All GIE components (**Storages** and **LNGs**) have the following mandatory attributes:

- **id**: unique identifier
- **name**: name of the pipe-segment
- **source_id**: a source id
- **node_id**: the id of the start and the end node of the pipe-segment
- **lat**: a list of latitude values
- **longitude**: a list of longitude values

- **country_code**: a string pair indicating the country code of the start and the end point
- **comment**: a user comment.

3.3.3 LNGs

Overall, there are 21 LNG terminals in the GIE data set. In addition to the default attributes, the following non-standard attributes (see Table 3.11) are supplied and partially populated with data:

Table 3.11: GIE *LNGs* data density

Attribute name	Description	Units	Data density [%]
eic_code	EIC code of LNG terminal		100
facility_code	unique facility code		100
max_cap_store2pipe_M_m3_per_d	maximum gas flow from storage to pipeline	Mm^3d^{-1}	100
max_workingGas_M_m3	maximum stored gas in LNG terminals	Mm^3	100
median_cap_store2pipe_M_m3_per_d	medium gas flow from storage to pipeline	Mm^3d^{-1}	100
name_short	short name of the facility		100

3.3.4 Storages

Overall, there are 99 storage points in the GIE data set. In addition to the default attributes, the following non-standard attributes (see Table 3.12) are supplied and partially populated with data:

Table 3.12: GIE *Storages* data density

Attribute name	Description	Units	Data density
eic_code	EIC code of storage facility		100
facility_code	unique facility code		100
max_cap_pipe2store_M_m3_per_d	maximum gas flow from pipeline to storage	Mm^3d^{-1}	75
max_cap_store2pipe_M_m3_per_d	maximum gas flow from storage to pipeline	Mm^3d^{-1}	72
max_workingGas_M_m3	maximum working gas in storage	Mm^3	77
name_short	short name of the facility		100

3.3.5 Nodes

Overall, there are 98 node points in the GIE data set. In addition to the default attributes, the following non-standard attributes (see Table 3.13) are supplied and partially populated with data:

Table 3.13: GIE *Nodes* data density

Attribute name	Description	Units	Data density [%]
exact	boolean indicating that storage is planed		100
eic_code	EIC code of storage facility		100
facility_code	unique facility code		100
name_short	short name of the facility		100

3.3.6 Data availability and data usage

The API of the GIE (Gas Infrastructure Europe (ASBL)) web portal allows for the user to download time series on the daily gas amount (stored or available) for gas storages and LNG terminals. Here, we do not pass on the downloaded time series information, but only other retrieved constant metadata, such as maximum storage capacity, or maximum re-gasification from LNG into the gases state that were derived using the time series data.

Copyright

The generally valid copyright regulations for databases apply.

3.3.7 Data disclaimer

In addition the data disclaimer is given as under [link](https://agsi.gie.eu/#/disclaimer):

> "All data is provided by the contributors on a voluntary basis and free of charge. The data provided by AGSI is for information purpose only. GSE is using reasonable efforts to invest in ensuring the correctness, completeness, and timeliness of the information provided herein. Data have been carefully checked, are updated at regular intervals and may be subject to changes, removal, or amendments without prior notice. GSE neither assumes any warranty or liability for the correctness and completeness of information/services and entries nor for the mode of presentation."

Summary GIE data

The GIE data set supplies information on gas infrastructure facilities all over Europe, such as gas storages and gas LNG terminals. Data for those facilities are accessible by the SciGRID software through special CSV data files, that are downloaded from the GIE web page. The information in those facilities are automatically filtered and reshaped to the data structure of SciGRID_gas. Units are partially converted to align with the other project data. The facilities are further geo-reference by the SciGRID_gas software with the help of the InternetData data set.

Below a table summarises the number of elements for each component found:

Table 3.14: GIE component summary

Component Name	Count
BorderPionts	0
Compressors	0
ConnectionPoints	0
Consumers	0
EntryPoints	0
InterConnectionPoints	0
LNGs	21
Nodes	98
PipeSegments	0
Production	0
Storages	99

In addition, the map Fig. 3.8 visualizes the data for Germany.

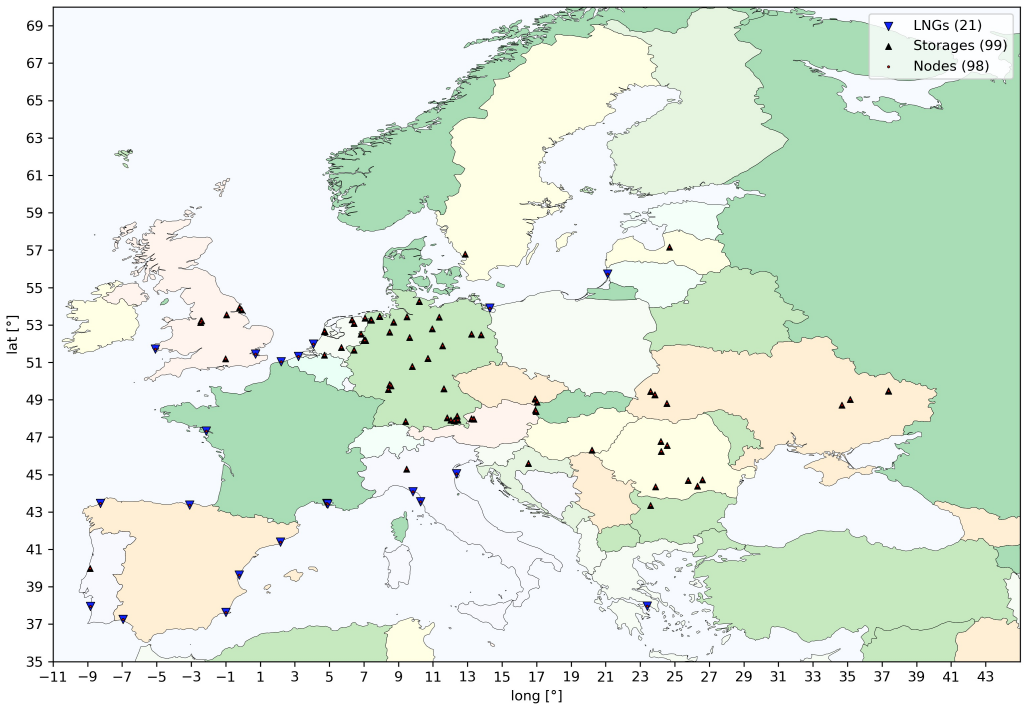


Fig. 3.8: Overview map of the GIE data set for Europe.

3.4 Non-OSM data summary

SciGRID_gas is based on open source data. To generate a gas pipeline network data set, one needs to access different data sets, that were found throughout the project and presented here. Emphasis was given to depict the number of elements per component and the data density for each data set.

3.5 Summary

Gas component data sets come in different forms, licenses, formats and detail. the SciGRID_gas project can process such data and combine them to a consistent and reliable network data set.

The underlying gas component data sets were categorized into two different groups:

- OSM data: This is data originating from the OSM data base, containing well geo-referenced locations of gas facilities, such as pipe locations or gas storage facilities. However, it comes with very few meta information.
- Non-OSM data: These are all other data sources, which can "supply" detailed information on some of the gas facilities attributes. However, this information is sparse, as published only for a few facilities. Here, the InternetData data set was introduced as an example of the non-OSM data set, and the pathway of converting the raw data from the www into SciGRID_gas project component structure.

Here detailed information on all available data sources have been given, and should be used as a reference for later data processes.

CHAPTER

FOUR

DATA TOOLS

This section supplies information on loading and saving SciGRID_gas component data sets from and to files. Currently import export options for CSV, Pickle and GeoJSON have been implemented and will be described here. In addition functions to visualize the data graphically and content wise will also be presented. Further, the comparison of elements from different data source is an important part of the SciGRID_gas project, where some fundamental tools will be presented here.

4.1 Reading and writing CSV files

The CSV file format is a very common file format, where the data is stored using ASCII characters. This type of file allows for quick verification of correct data content, whereas the quick verification of values within a binary file can not be easily carried out. Tools have been implemented within the SciGRID_gas project to read and write data from the project from and into CSV files.

Description

The CSV files here are significantly different to the InternetData CSV files. The location information and metadata have been combined into a single file for each element, the number of header lines have been reduced to one, most relevant attributes are stored as a Python dictionary variable structure, and additional attribute groups have been generated for uncertainty of the attribute values and the way that the attribute values were generated. However, there will be some similarities, as will be described below.

First of all, the filename structure is described:

- The files are of type CSV and need a CSV extension, with ";" as the column separator.
- The first four characters of the file names need to be "Gas", followed by an underscore.
- The following characters of the file names need to be the name of a SciGRID_gas components, such as *Compressors* (resulting in the name "Gas_Compressors.CSV") or *BorderPoints* (resulting in the name "Gas_BorderPoints.CSV").

The below Fig. 4.1 depicts a sample CSV file for the component *Compressors* (from the LKD data source):

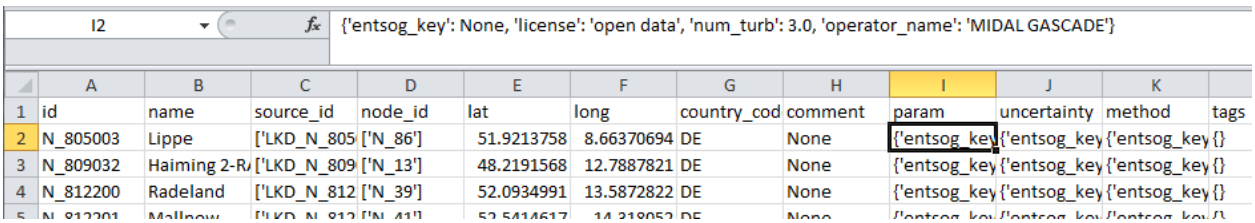


Fig. 4.1: Sample CSV file.

As stated above, the CSV files only contain one header line. The header line contains the attribute labels, which will also be used as variable names within the SciGRID_gas project Python code. Hence, changing label names could result in breaking of the SciGRID_gas code.

One of the similarities to the previously introduced InternetData data files are the first eight columns. They are as follow:

- **id**: This is the unique id of the element, of type string.
- **name**: This is a name given to the element.
- **source_id**: This is a list of source id strings, that describes the origin of the values of the element. As it is a list, the entry needs to be surrounded by square brackets, e.g. "['LKD_N_809032']". Here an example is given with two entries, as each element needs to be separated by a comma "['N_289', 'N_300']".
- **node_id**: This is the corresponding node id, of the node where this element is located. The same rules regarding the list for the *source_id* apply.
- **lat**: This is the single (or list for pipe-segments) latitude value.
- **long**: This is the single (or list for pipe-segments) longitude value.
- **country_code**: The two letter country code as listed in Table 9.3.
- **comment**: A comment from the user. However, it will not be used within the Python code.

In addition, there are four further columns:

- **param**
- **uncertainty**
- **method**
- **tags**.

The connection between *param*, *uncertainty* and *method* has been described in the data structure chapter (see Chapter 3). Here we will concentrate on the syntax in the CSV files, by looking at the following example for the compressor at location Lippe (first line in CSV example file Fig. 4.1). The *param* entry is as follow:

- {'entsog_key': None, 'license': 'open data', 'num_turb': 3.0, 'operator_name': 'MIDAL GASCADE'}.

One can see curly brackets ("{" and "}") at the start and the end of the entry. They indicate to the Python program, that an entry of type dictionary is given.

Within the dictionary, one can see pairs of entries, separated by a colons. The first entry is referred to as the key, and is a string, thus needs to be placed in quotes. In the above example we have four keys. Those keys are being converted to attribute labels when read in by the SciGRID_gas program. They are:

- 'entsog_key'
- 'license'
- 'num_turb'
- 'operator_name'.

And the corresponding attribute value can be found behind the colons. For the example above the values are:

- None
- 'open data'
- 3.0
- 'MIDAL GASCADE'

The attribute 'entsog_key' has been given no value, thus the attribute value is "None". The 'license' has an attribute value of 'open data', the attribute 'num_turb' (number of turbines at the compressor facility) has a value of 3.0, and the attribute 'operator_name' has a value of 'MIDAL GASCADE'.

The same syntax has been used for the elements *uncertainty* and *method*. The element *uncertainty* contains the same attribute keys as in the *param* dictionary, and contains absolute values of uncertainties for each attribute. For a data set that has not been modified and only read in from its original source, such as the InternetData data set, the uncertainty values will be "0", as it is assumed that all original data has no uncertainty. In addition all data found on the internet for the InternetData data set was supplied without an error value. However at a later stage, when missing values will be generated, the uncertainty value for each attribute will be simulated and will depend on the the method and the input data used to generate the missing attribute values.

The element "method" contains information on the origin of the attribute value. For a data set that has not been modified and is only read in from its original source, then the entries will be "raw". However, other entries will exist, as will be described in a chapter describing the heuristic data generation.

Currently the column *tags* contains an opening and closing curly bracket only ("{}"). This is a placeholder for later additional information.

A schematic overview of how the component CSV files are connected is given in Fig. 4.2.

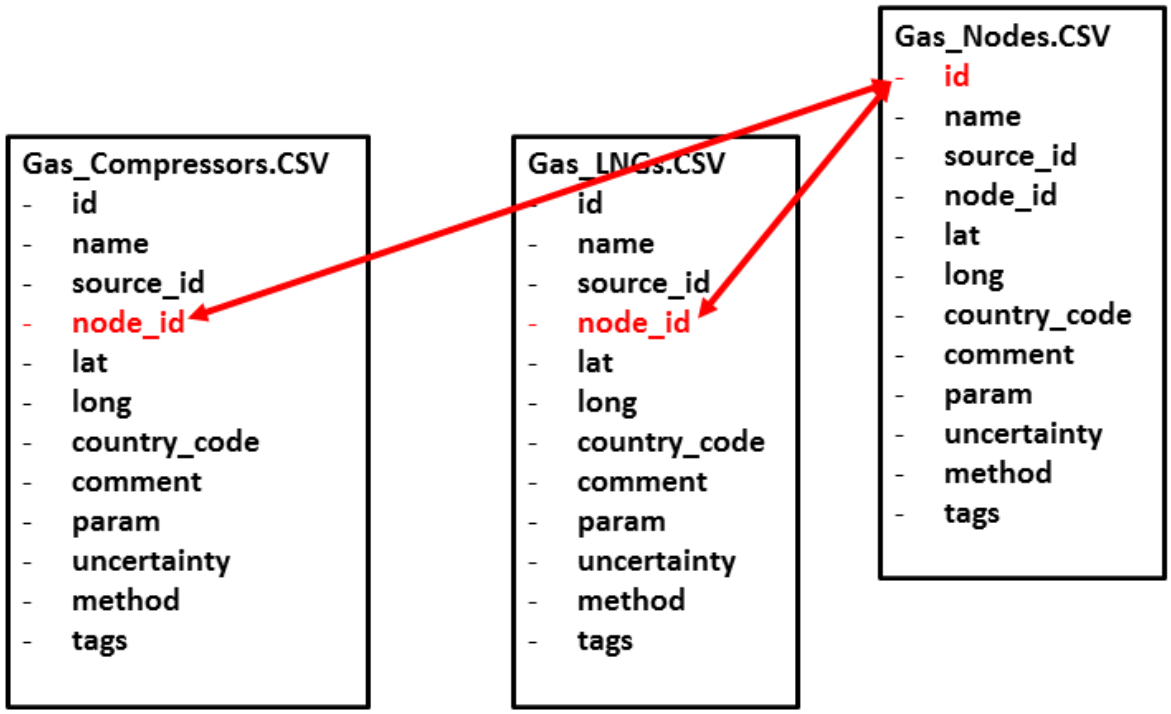


Fig. 4.2: Schematic of the CSV files and their connection to the nodes information supplied through the "Gas_Nodes.CSV" file.

The CSV file for the component *Nodes* contains a unique identifier "id", as depicted above. All other components, such as the component *Compressors* contain next to their own unique id "id" a variable with label "node_id". This "node_id" contains an "id" from the CSV file "Gas_Nodes.CSV", indicating that the compressor is located at the node with id "id".

Read CSV files

The following Python code within the SciGRID_gas code can be used to read data that is stored in a CSV files. This can be achieved by executing the following (as long as this data sets exists at the location selected in the example):

```
import Code.M_CSV as M_CSV

Netz        = M_CSV.read('Eingabe/InternetDaten/RawData/')
```

With the above two code lines one can read in all the component files from a given directory, and the data set will "end up" in the variable "Netz", as long as their file name convention follows the rules as described above.

The main code line that is reading in the data is:

```
Netz    = M_CSV.read('Ausgabe/InternetDaten/RawData/')
```

whereas the first line in the above example declares a module that is needed for executing the last line of code, hence start with the word "import". Th last line of code reads the gas data from the location given as a string to the function. The word "Netz" is German and stands for "network".

Write CSV files

Section 2.1 describes the process of transforming the raw gas data into a data set that can be used by the SciGRID_gas project, where this process has been divided into 5 blocks. In-between those blocks one might want to save the data, or switch to another data set that was previously generated and stored. Hence, one can store data sets. Here the actual commands of reading and writing the data will be described briefly.

Assuming that a gas dataset is stored in the variable "Netz" in the current Python session, the following Python code of the SciGRID_gas project can be used to write the SciGRID_gas "Netz" gas data set to CSV files:

```
import Code.M_CSV as M_CSV

M_CSV.write('Ausgabe/InternetDaten/RawData/', Netz)
```

With the above two code lines one can write all components of the SciGRID_gas "Netz" network data set to files, where the location of the files is given through the first argument, being a relative directory name. The generated files will have the same structure as described in the section "CSV input", and the filename convention will be the same as described in that same section.

The main code line that is writing the data is:

```
M_CSV.write('Eingabe/InternetDaten/RawData/', Netz)
```

whereas the first line in the above example declares the module that is needed for executing the second line of code, hence start with word "import".

4.2 Reading and writing Pickle files

Pickle files are a further file type that the SciGRID_gas project can utilise. Pickle "serialises" the object first before writing it to file. Pickling is a way to convert a python object (list, dict etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script [Python_DC]. Hence Pickle has been implemented into the SciGRID_gas project and can be used as a data storage option. Tools have been written to write to Pickle files and read SciGRID_gas gas data from Pickle files. Here an example of storing and loading the "Netz" object from the previous example with Pickle is given.

```
import * from Code.M_PCKL   as M_PCKL
M_PCKL.picklesavelist(Netz,'GasDataSet', 'Ausgabe/InternetDaten/PickleData/')
Netz = M_PCKL.pickleloadlist('Ausgabe/InternetDaten/PickleData/', 'GasDataSet')
```

4.3 Writing GeoJSON files

GeoJSON is similar to shape files, but it is based on an open standard format designed for representing simple geographical features, along with their non-spatial attributes. Features include points, lines, polygones and multi-part collections of these types. Hence, a gas data set from the SciGRID_gas project could be converted into a GeoJSON format and saved. Hence, it is possible to export all elements of a Netz-object to separate GeoJSON-files with the commands presented below:

```
import Code.M_CSV as M_CSV
from Code.export_geojson import export_geojson

export_geojson(Netz, 'Netz', path='../Ausgabe/GeoJSON')
```

Files generated with the GeoJSON routines can be loaded and visualized by other applications, and it a well accepted format. Currently a reading option from a GeoJSON file format back into the Python SciGRID_gas environment has not been implemented.

4.4 Network data visualization

Throughout the SciGRID_gas project different data sets of gas facilities are being merged, compared, loaded, accessed and so on. Hence tools are required that aid in the comparison of network or to check resulting networks.

There are currently three different methods implemented in the SciGRID_gas project that help the user:

- displaying component lists on the Python console
- plotting the network on a map
- writing network data to a CSV file.

The first two methods will be explained in the subsequent methods sections, whereas the last method has been described in Section 4.1.

A sample data set it being supplied to help in working through the network visualization process. For the following sections it is assumed that you can work in a Python prompt so that one can follow those step-by-step instructions. Prior to demonstrating the methods, the sample data set needs to be loaded. This has been explained in Section 4.1.

4.4.1 Displaying component lists on the Python console

Here it is assumed, that you have access to the sample data set, that should have been shipped with this documentation. The sample data set is located in the following location: '/Eingabe/Sample/'. Using the M_CSV.read function this data set can be loaded as depicted below. After having loaded the sample data set, one can access information on the data set by the execution of the following Python command:

```
TestNet    = M_CSV.read('Eingabe/Sample/')
TestNet.all()
```

Here "TestNet" is the label that was given to the sample data set. Hence if the user choose a different label, e.g. "Test2", then the user would need to execute the following line:

```
Test2.all()
```

After pressing the return key, the user should see the following on the Python prompt:

```
--------------------------------------
Source
total component type count          9
--------------------------------------
BorderPoints                      117
Compressors                       237
ConnectionPoints                    0
Consumers                           0
EntryPoints                        37
InterConnectionPoints             117
LNGs                               33
Nodes                             740
PipeLines                           0
PipePoints                          0
PipeSegments                      531
Processes                           0
Productions                         0
Storages                          197
--------------------------------------
Length of PipeLines    [km]         0
Length of PipeSegments [km]     46034
```

Here, the number of elements for each component are being displayed. Hence, it can be seen that there are 237 *Compressors* element in the loaded data set. In addition, this sample data set contains 46034 km of gas pipelines in form of pipe-segments.

On the coding level, here the variable (instance of a network class) **TestNet** contains all the information. And the function (method) **all()** creates the printout on screen by looking at the data and getting the appropriate information needed.

4.4.2 Displaying element attribute values on the Python console

This function **all()** also exists for each individual element. As Python is zero based each component array starts with 0. Hence, if one wants to get information on the first element of the compressors one needs to execute the following command on the Python prompt:

```
TestNet.Compressors[0].all()
```

The Python prompt should display the following information:

```
comment: None
country_code: GB
id: Aberdeen
lat: 57.13600455
long: -2.379727732
name: Aberdeen
node_id: ['N_424']
source_id: ['IN_Aberdeen']
```

The output for the "method" dictionary is:

```
method: {'comment': None, 'end_year': None, 'is_H_gas': 'raw', 'license': None, 'max_
↪cap_M_m3_per_d': None, 'max_power_MW': None, 'max_pressure_bar': None, 'num_turb':␣
↪None, 'operator_name': None, 'pipe_name': None, 'source': 'raw', 'start_year': None,
↪ 'turbine_power_1_MW': None, 'turbine_power_2_MW': None, 'turbine_power_3_MW': None,
↪ 'turbine_power_4_MW': None, 'turbine_power_5_MW': None, 'turbine_power_6_MW': None,
↪ 'turbine_type_1': None, 'turbine_type_2': None, 'turbine_type_3': None, 'turbine_
↪type_4': None, 'turbine_type_5': None, 'turbine_type_6': None}
```

(continues on next page)

(continued from previous page)

The output for the "param" dictionary is:

```
param: {'comment': None, 'end_year': None, 'is_H_gas': 1, 'license': None, 'max_cap_M_
↪m3_per_d': None, 'max_power_MW': None, 'max_pressure_bar': None, 'num_turb': None,
↪'operator_name': None, 'pipe_name': None, 'source': 'Googlemaps', 'start_year':␣
↪None, 'turbine_power_1_MW': None, 'turbine_power_2_MW': None, 'turbine_power_3_MW':␣
↪None, 'turbine_power_4_MW': None, 'turbine_power_5_MW': None, 'turbine_power_6_MW':␣
↪None, 'turbine_type_1': None, 'turbine_type_2': None, 'turbine_type_3': None,
↪'turbine_type_4': None, 'turbine_type_5': None, 'turbine_type_6': None}
tags: {}
```

The output for the "uncertainty" dictionary is:

```
uncertainty: {'comment': -999, 'end_year': -999, 'is_H_gas': 0, 'license': -999, 'max_
↪cap_M_m3_per_d': -999, 'max_power_MW': -999, 'max_pressure_bar': -999, 'num_turb': -
↪999, 'operator_name': -999, 'pipe_name': -999, 'source': 0, 'start_year': -999,
↪'turbine_power_1_MW': -999, 'turbine_power_2_MW': -999, 'turbine_power_3_MW': -999,
↪'turbine_power_4_MW': -999, 'turbine_power_5_MW': -999, 'turbine_power_6_MW': -999,
↪'turbine_type_1': -999, 'turbine_type_2': -999, 'turbine_type_3': -999, 'turbine_
↪type_4': -999, 'turbine_type_5': -999, 'turbine_type_6': -999}
```

In addition, one can see that the compressor has a country code ("country_code") of "GB", as it is located in the UK, with its latitude of 57.13600455 and a longitude of -2.379727732.

To get the attribute values of the last compressor element one would need to execute the following command:

```
TestNet.Compressors[236].all()
```

as there are 237 compressor elements.

4.4.3 Data density on the Python console

Further information that might be of interest to the user is the data density as defined in Section 9.1. For this a function has been written as part of the SciGRID_gas project, which will return the number of non-missing attribute values of elements. This can be executed with the following command:

```
TestNet.get_AttribDensity(CompName = 'Compressors')
```

The output would look as follow:

```
{'id': 237,
'name': 237,
'source_id': 237,
'node_id': 237,
'lat': 237,
'long': 237,
'country_code': 237,
'comment': 15,
'uncertainty': 237,
'method': 237,
'tags': 237,
'end_year': 0,
'is_H_gas': 236,
'license': 0,
```

(continues on next page)

(continued from previous page)

```
'max_cap_M_m3_per_d': 19,
'max_power_MW': 33,
'max_pressure_bar': 18,
'num_turb': 38,
'operator_name': 27,
'pipe_name': 26,
'source': 108,
'start_year': 19,
'turbine_fuel_isGas_1': 36,
:
:
'turbine_type_6': 0}
```

Here the user can see that there are 237 elements of type *Compressors*, as there are 237 compressor ids, however for the attribute 'max_cap_M_m3_per_d', only 19 elements have a non-None value. Here, numbers are give as absolute values. This function will give the user a better understanding of the quantity of the data.

4.4.4 Plotting a network map

While generating a gas transmission network the user might want to visualize the locations of component elements on a map. Hence several tools have been created that aid in achieving this within the Python coding environment. Here it is assumed that you have executed the three lines from sub-section "Loading sample data set".

The simplest commands to execute and plot the data are:

```
import Code.M_Visuell    as M_Visuell

M_Visuell.quickplot(TestNet)
```

One should see the following map (Fig. 4.3):

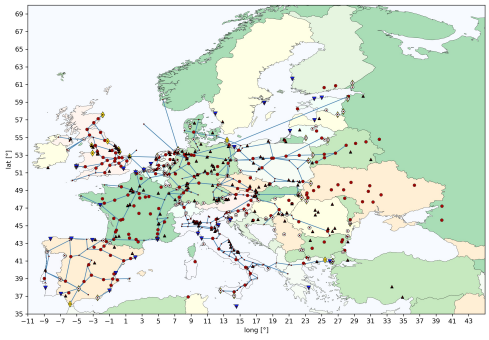


Fig. 4.3: Map of the loaded data, generated with the command "M_Visuell.quickplot()", that was specified during the method call "quickplot".

There are several options such as map extend, legend, and more. A few will be described here briefly, where a more detailed and in depth function and method description is give through the function documentation. The parameters that can be supplied to the plotting routine that are being described here are:

- legend on the map: By adding the following text "LegendStr = '', LegendStyle = 'Str(Num)'," to the function call, a legend will be added as depicted below.

- saving of the map: By adding to the function call the following text "Save = True, savefile = *path and file name*", the plot will be saved to the location that is specified by *path and file name*.
- cursor for the map: An interactive cursor can be switched on as well, by adding the following settings to the function call "Cursor = True". Hence by clicking with the left mouse button on an element, a brief ID of the selected element will be displayed on the map, while executing this in the Python environment.

Here, all three options have been added in the call below:

```
RelDirNamePlot      = 'SampleMap_2.jpg'
M_Visuell.quickplot(TestNet, LegendStr = '', LegendStyle = 'Str(Num)', Cursor = True,␣
↪Save = True, savefile = RelDirNamePlot)
```

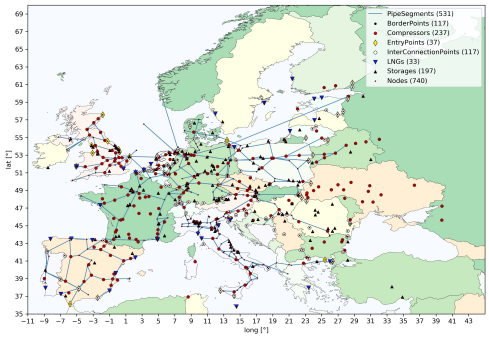


Fig. 4.4: Map of the loaded data, generated with the amended command "M_Visuell.quickplot(TestNet, LegendStr = '', LegendStyle = 'Str(Num)', Cursor = True, Save = True, savefile = RelDirNamePlot)".

Summary of data visualization

Three different methods of "visualization" of gas data sets were described. They include a generation of a visual geo referenced map, and the generation of a data component summary and data displays within Python.

4.5 Element comparison tools

The same facility can be described and listed in different data sets originating from different data sources. However to create the best possible final gas data set, one needs to combing all possible data sets. As facilities can be described in several data sets, but are only allowed occur once in the final data set, those elements need to be found and combined. This section here will describe some of the tools and concepts implemented into the SciGRID_gas project, to achieve this task.

Description of problem

Throughout the SciGRID_gas project different sources of data need to be merged. One of the main tasks is to merge data, component by component. For a better understanding the problem is described with the example below.

Here, we start by assuming that there are two different data sets, D1 and D2 respectively, given as:

- The data set D1 has 10 elements of the component *Compressors*.
- The data set D2 has 15 elements of the component *Compressors* as well.

So the fundamental problem is, how can those two data sets be merged. Should the resulting data set have 25 elements of the component *Compressors* or less, as some of the elements could be describing the same facility?

Hence, several processes have been written for SciGRID_gas in Python that help in determining if two elements from different sources are describing the same facility. The methods designed will be described in the following sub-section.

4.5.1 Element matching functions

Below different functions are introduced briefly, that help in merging different data sets. They are being used to estimate if two elements from different sources are describing the same facility, and hence should be merged. The functions introduced here are applicable to all components except **PipeLines** and **PipeSegments**. Different functions look at different attribute values, such as **names**, **LatLong** or **country code**. Each function returns a score between 0 and 100, with 0 indicating that there was no match between the attribute values supplied, whereas a score of 100 refers to a perfect match between the attribute values. In a second step, the user can set thresholds, so that a function returning a value larger than the set threshold is assumed to be describing the same facility. However, as single attribute comparison might be misleading (e.g. very similar place name in two different countries), multiple attribute comparisons have also been coded.

getMatch_Names

This function helps to determine if two elements describe the same facility by comparing their location **names**. For the value of 100, the location names are identical, whereas for a value of 0 there is no similarity between the names at all. The names of "UGS Stollen" and "Stollen" would return a score of about 70, as the word "Stollen" is partially in the name "UGS Stollen". Here an external Python module "FussyWuzzy" is being used.

getMatch_Names_CountryCode

This is a function that builds on the function **getMatch_Names**, where the return score also depends on the country code of the two elements. Here again the total score returned can be 100, however half of the final score is determined if the two elements have the same country code or not. The other half of the return value is determined through the method **getMatch_Names**.

getMatch_LatLong

This function determines if two elements can be classed as the same by their geo-reference location, by calculating the distance between the two locations. E.g. if the distance is 500 km, then it would be highly unlikely that the elements are of the same facility. However, if the distance between the locations is 10 km or less, then in the scheme of Europe they might be describing the same element. For this function different methods have been implemented, so that the distance can be weighted in different ways, such as just the inverse distance, or the inverse power distance or the inverse log distance. Different methods work best for different components. Here again, if the lat/long of two elements is equal, then this function would return a value of 100, whereas if the distance between two elements is several 100 or even 1000 km, then this function would return a very small value such as 10 or 5, depending on the method selected.

getMatch_LatLong_CountryCode

This is a function that builds on the function **getMatch_LatLong**, where the return score also depends on the country code of the two elements. Here again the total score returned can be 100, however half of the final score is determined if the two elements have the same country code or not. The other half of the return value is determined through the method **getMatch_LatLong**.

getMatch_LatLong_Threshold

This function only checks if the distance between two locations is smaller (returns a score of 100) or larger (returns a score of 0) than a user supplied value.

Above, several methods were described that help in determining if two elements are representing the same facility. Overall, one can group those methods into two groups. One group deals with the names of locations, whereas the other group deals with the lat/long of the locations. To increase the likelihood of "correct" matches, all of the above methods can be combined. By combining two or more methods, their individual scores was averaged in the process.

4.5.2 Pipe-segment matching functions

The above function only work for non-pipe components, as they are a single point, whereas pipe-segments and pipelines connect at least two points. In addition, in the real world, pipe-segments can run parallel to each other. However they might have different attribute values, such as pipe diameter or maximum operating pressure. If one wanted to merge two data sets of pipe-segments, one needs to consider those different operating conditions as well, to differentiate between different pipe-segments.

So how does one determine that two pipes from different sources are different pipes? To achieve this one will need to look at the following attributes:

- "diameter"
- "max operating pressure"
- "length"
- "gas type".

A further complication arises, as one data set describes a network with fewer nodes and long pipe-segments, whereas the other data set contains more nodes and shorter pipe-segments. The possible solution to this problem is that one does not just compare one pipe-segment from one data source with only one pipe-segment from another data source, but that one allows for multiple pipe-segment selections to be compared. As depicted in Fig. 4.5, the blue and the red pipe-segments connect point "A" and point "B". However, the blue pipe-segments also passes through nodes "C" and "D".

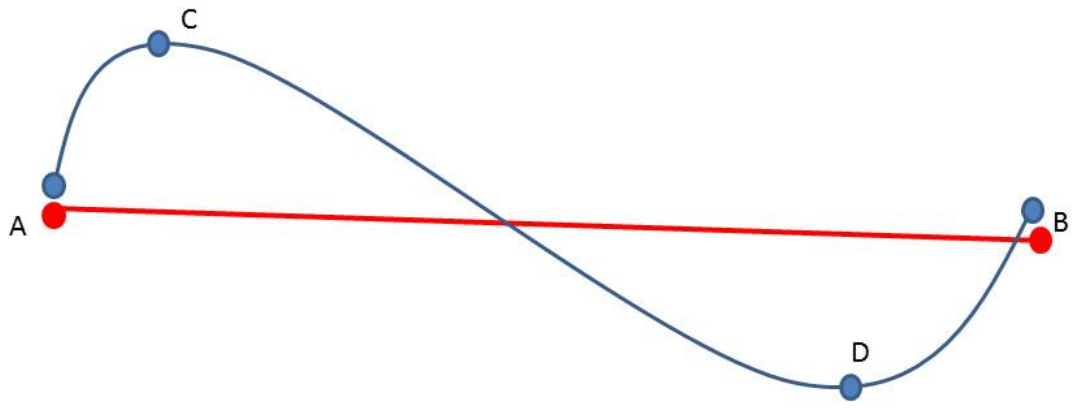


Fig. 4.5: Schematic sample of two pipelines connecting locations "A" and "B", where the pipe-segments from one data set also passes through locations "C" and "D".

For the above example, even though that the blue data set contains a different path and length of the connection between node "A" and "B", both data sets of pipe-segments might describe the same pipe-segments.

A further problem is that not all attribute values are supplied for each element from the different data sets. Hence, the following rules are applied:

- If both elements contain the attribute, and both have valid attribute values (number or string), then it can be used for a similarity test (e.g. both data set have an attribute **diameter** and both have a valid number as the attribute value).
- If only one data set contains the attribute, then this attribute will not be considered for any similarity tests (e.g. the first data set has the attribute **diameter**, whereas the second one does not have that attribute).

- If both elements contain the attribute, but at least one does not have a valid attribute value, then this attribute will not be considered for any similarity tests (e.g. both data sets have the attribute **diameter**, however at least one is missing an entry value).

Below the individual steps are being described to determine if pipe-segments from different data sources are the same. A flowchart of the steps is also depicted in Fig. 4.6.

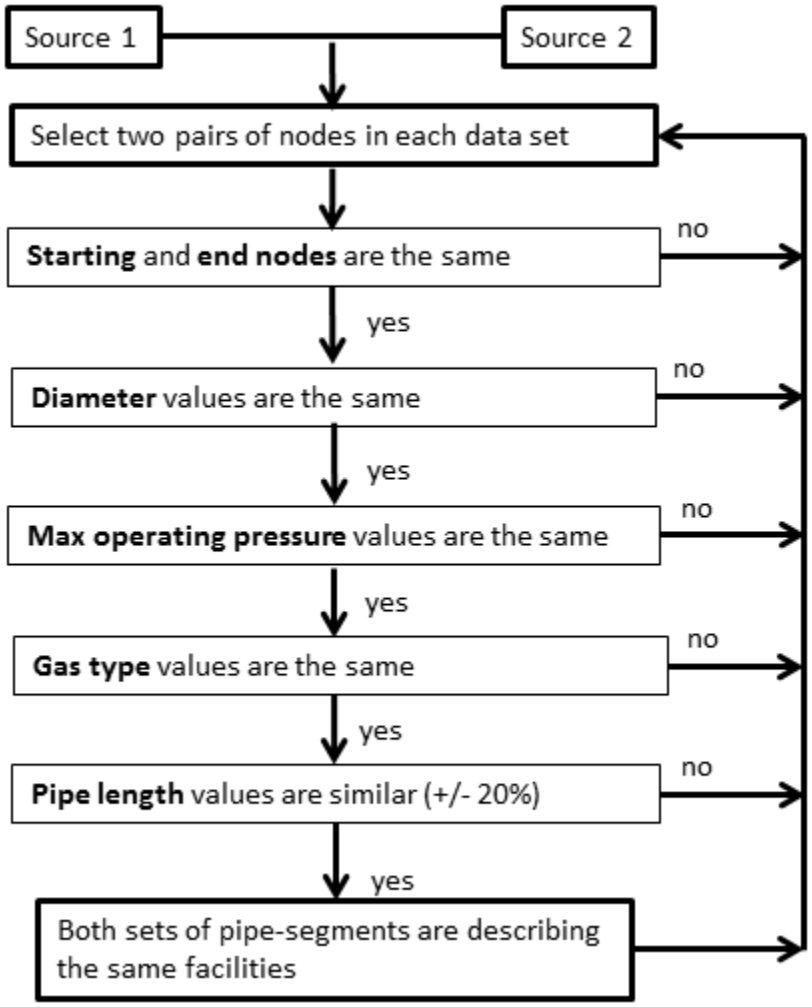


Fig. 4.6: Comparing pipe-segments from different data sources decision flow chart.

For all attributes, except pipe-segment length, the values if given need to be identical. Only for the pipe-segment length comparison, a difference up to 20% is tolerated.

If all of the checks above are passed, then it is assumed, that the pipe-segments originating from different data sources are describing the same pipe segments, hence can be merged.

4.6 Summary

The main task of the SciGRID_gas project is to generate a gas network data set that can be used in gas transmission models. Several data storage options have been coded for the SciGRID_gas model. They include simple CSV files, Python Pickle files and GeoJSON files. They have been described for the reading and writing processes for the SciGRID_gas project in this chapter.

In addition, further SciGRID_gas functions have been described, for the "visualization" of data. Further functions in determining of elements are describing the same facilities, originating from different sources have been introduced.

CHAPTER

FIVE

HEURISTIC METHODS

Gas facility data sources have been described in Chapter 3. However, those data sources might not describe the same facilities and might not contain the same attributes. Hence those data sets need to be combined. Such a combined data set will still contain a large number of missing attribute values. These values need to be generated. This chapter here will describe the current implemented heuristic methods that can be used to fill missing values and elements.

5.1 Attribute value generation

The SciGRID_gas project has been set up to deliver a data set of the European gas transmission network. Despite merging several data sources of gas facilities, the resulting gas component data set will contain a large number of missing values. This section here describes how missing attribute values can be generated through different heuristic methods. The tools developed here have been designed for the non-OSM data sets. However, they should be applicable to any gas component data set.

In this section, the problem of missing data is described with the help of some mock data set. This is followed by the description of the heuristic methods that have been implemented, and the general pathway that the user needs to undertake to eliminate missing values. A chapter later in this document will give hands on examples of code that needs to be executed to generate missing values.

Problem description

In the Fig. 5.1 the problem that the SciGRID_gas project is facing is depicts. There are different elements of different components, where many attributes values could not be found. The example given in the following sections shall depict the gas pipelines, where the attributes in question are *diameter*, *capacity* and *pressure*. The data is summarized in Table 6.8.

Table 5.1: Summary of data of the seven sample pipelines from Fig. 5.1.

Pipeline name	*capacity* $[Mm^3d^{-1}]$	*pressure* $[bar]$	*diameter* $[mm]$
Jagal	76	80	1200
RHG		84	800
Midal 1	35		900
Midal 2	35		1000
Midal 3	35		800
Midal 3	35		800
Stegal			800
Stegal			800
Wedal	27		800

As can be seen, all pipe-segments contain an attribute value for the attribute *diameter*. For the attribute *capacity* three values are missing, and for the nine pipe-segments, only two have a value for the attribute *pressure*. So the task will be to fill as many missing attribute values as possible. The corresponding data densities for the attributes *capacity*,

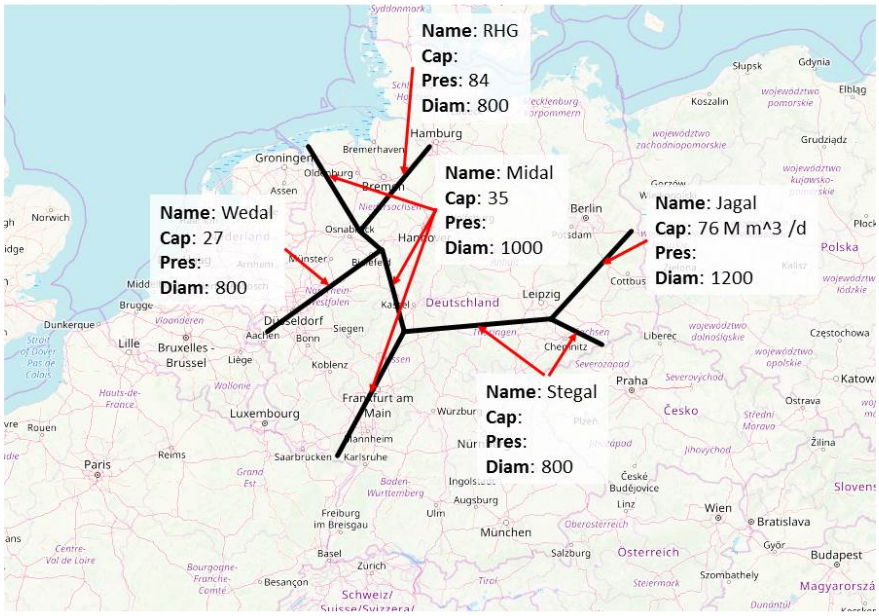


Fig. 5.1: Map of some of the larger pipelines in Germany, with corresponding attributes *capacity* (cap), *pressure* (pres), and *diameter* (diam).

pressure and *diameter* are 67 %, 22 % and 100 % respectively. And the overall goal will be to achieve a data density of 100 % for all attributes.

5.1.1 Fill value methods

In the case of the pipe-segment attribute *diameter*, no heuristic method needs to be applied, as no attribute values are missing.

In the case of the pipe-segment attribute *pressure*, only two pipe-segments supplied a value. To find values for all other pipe-segments one has the following options:

- Not to fill the missing values, as one is not confident enough that those two values are representative enough.
- Assume a value, that could be used as a fill value. Here one could use a value of 82 bar as this is the average value of the two supplied values. However, it would be very hard to determine an uncertainty error for this approach, as one does not know anything else about this attribute.

As one can see, the *capacity* attribute value is given for six of the nine facilities. Hence, there are several options of determining the missing value. A simple solution would be to use the average or median of the input values as a fill value. Here the *mean* and *median* value would be $41\ Mm^3d^{-1}$ and $35\ Mm^3d^{-1}$, respectively. However, selecting the best approach can be difficult, and needs to be transparent. Hence, an "error" term could help to assess these methods.

In the world of statistics, one normally deals with a lot of data, hence they would have 1000 and more data sets. Then one would split the data into a training data set and a test data set. In the first step a method (e.g. median) is applied to the training data set. In the second step, the fitted method results are used to predict the responses for the observations in the test data set. In the third step an absolute error can be determined between the test data set and the method predictions. The smaller the error the better the method. However, the SciGRID_gas project only contains small data sets. Any splitting of the input data set into a training and test data set would create a too small a data set for training purposes. E.g. there are only 35 LNGs terminals in Europe, and splitting such data set would result in 10 or less

values for training or testing. Hence, throughout the SciGIRD_gas project the "Leave-one-out" method will be used (see Appendix `Anhang_StatisticalBackground`), and the *error* is calculated using the mean absolute error (*MAE*), where the absolute error is the absolute difference between a single input data value and the model estimation of that value instance.

The *MAE* for the *mean* and the *median* method is $25\,Mm^3d^{-1}$ and $16\,Mm^3d^{-1}$, respectively. Hence, based on the *MAE*, it would be best to use the *median* method approach, and one could fill all missing values with the value of $35\,Mm^3d^{-1}$, with an *MAE* of $16\,Mm^3d^{-1}$. As the *MAE* is very large in respect of the actual *capacity* value, other method approaches also need to be investigated.

An attribute could have a linear correlation with one or several other attributes. Here, one could use a linear regression. However, linear regressions tend to weight the independent feature data equally, if more than one are given. Other methods, such as the Lasso-linear regression tend to weight the independent variables unequally and can even suggest the removal of explanatory variables. Hence, if not stated otherwise, the Lasso-linear regression will be used here, instead of a simple linear regression.

Here, the Lasso-linear regression is applied to the *capacity* variable (also referred to as the "predictor" or "regression input"), and the variable diameter is the independent variable (also referred to as the "feature" variable). For the pipe RHG, where the *capacity* value is missing and a *diameter* value of $800\,mm$ is given, the Lasso-linear regression estimated the following *capacity* value: $28\,Mm^3d^{-1}$ with a *MAE* of $1\,Mm^3d^{-1}$. As one can see, the *MAE* is significantly smaller when compared with the methods of *mean* and *median*. However, this example should not lead to the assumption, that a Lasso-linear regression is always better than a simple estimation using a *mean* or a *median* value. In addition, for some attribute values the methods of *mean* or *median* might have to be used, due to lack of feature data.

Here, three different methods of generating missing values have been described. They have been compared in respect of their resulting *MAE*. The following section will describe the implemented method pathway within the SciGRID_gas project code.

5.1.2 Attribute value generation pathway

This section describes how the generation of the missing attribute values has been implemented into the SciGRID_gas project. Overall, there are six steps that need to be carry out. They are described in more detail in the following sub-sections:

1) Loading network data

2) Setting up the setup files

3) Generation of plots for data QA

4) Generation of parameters for the heuristic methods

5) Selecting method and feature variable combinations for each attribute

6) Simulation and filling of attributes values

In addition see Fig. 5.2 for a flow chart of the six steps.

1) Loading network data

As the first step, the data needs to be loaded into memory (see Section 4.1 for the functions that have been supplied to read the data from different data sources)

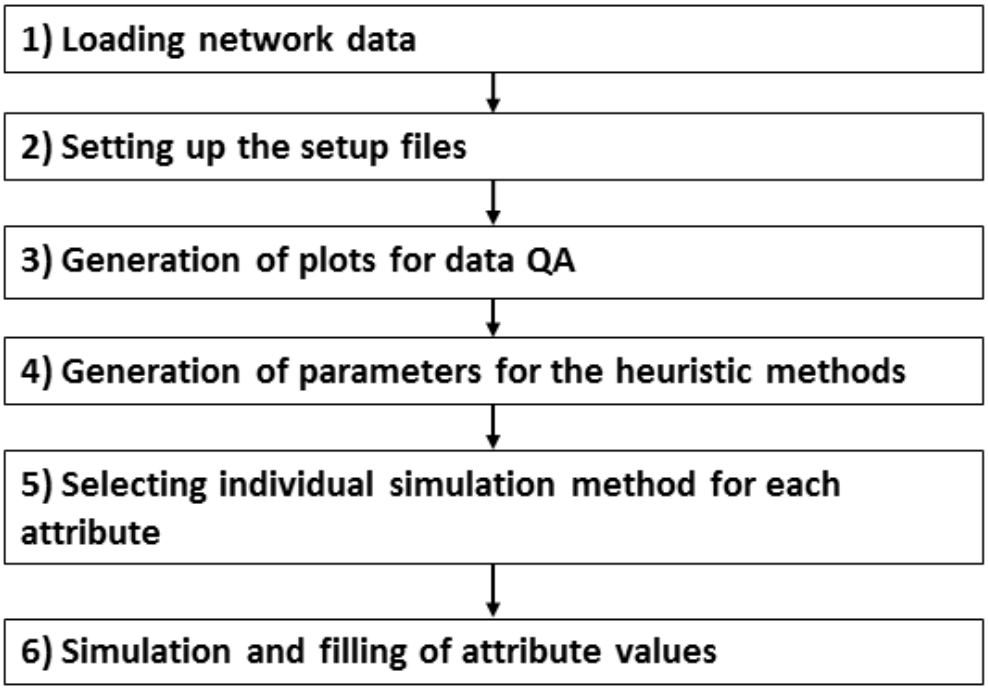


Fig. 5.2: Flow chart overview of the method-pathway that is implemented for the generation of missing attribute values.

2) Setting up the setup files

This part describes how to set up the required setup files. There are two setup files as part of the SciGRID_gas project: "StatsMethodsSettings.csv" and "StatsAttribSettings.csv". In the first one, meta information for each method (e.g. mean, median) is being supplied in addition to other settings. The second setup file contains a list of attributes, including attribute specific metadata. Both setup files are described in more detail below.

StatsMethodsSettings.csv

Through this file the user selects which method (e.g. "Lasso") shall be used for testing for the data. A sample method setup file is given in Fig. 5.3.

	A	B	C
1	MethodNam	Param	ToBeApplied
2	Lasso		1
3	LogisticReg	{'solver':'lbfgs'}	1
4	Mean		1
5	Median		1
6	Min		1
7	Max		1

Fig. 5.3: Sample file of the file "StatsMethodsSettings.csv"

Column "A", which has the label "MethodName", contains the heuristic method names that have been implemented into the SciGRID_gas project code. Currently the following methods have been implemented:

- **Lasso**: A type of linear regression that uses shrinkages. It is described by the scikit-learn.org web portal as [scikit_Reg]: "The Lasso is a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer non-zero coefficients, effectively reducing the number of features upon which the given solution is dependent. For this reason Lasso and its variants are fundamental to the field of compressed sensing. Under certain conditions, it can recover the exact set of non-zero coefficients."
- **LogisticReg**: Here the attributes to be predicted are of binary type or are multiple discrete values. The Logistic-regression is described by the scikit-learn.org web portal as [scikit_Reg]: "Logistic regression, despite its name, is a linear model for classification rather than regression. In this model, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.".
- **Mean**: Calculation of the *mean* attribute value of the predictor attribute values.
- **Median**: Calculation of the *median* value of the predictor attribute values.
- **Min**: Calculation of the *min* value of the predictor attribute values.
- **Max**: Calculation of the *max* value of the predictor attribute values.

The second column with the label "Param" contains possible parameters, that are applied to the method within the SciGRID_gas Python code. Here for the LogisticReg a solver needed to be specified. As can be seen, the following entry was supplied:"{'solver':'lbfgs'}". All other methods currently do not need additional parameter settings.

The third column with the label "ToBeApplied", contains the setting if the method shall be part of the testing suit (set to "1"), or not (set to "0").

StatsAttribSettings.csv

In this CSV file ("StatsAttribSettings.csv"), the metadata in respect of the attributes is being supplied. Here the user selects the attributes (e.g. "max_cap_M_m3_per_d"), which shall be used during the heuristic testing suit A sample of such file is presented in Fig. 5.4.

	A	B	C	D	E	F
1	CompName	AttribName	Features	Predictors	Convert2Float	RegressionType
2	Compressors	max_cap_M_m3_per_d	1	1	0	lin
3	Compressors	is_H_gas	1	1	0	log
4	Compressors	max_power_MW	1	1	0	lin
5	Compressors	max_pressure_bar	1	1	0	lin
6	Compressors	num_turb	1	1	0	lin
7	Compressors	turbine_fuel_isGas_1	1	1	0	log
8	Compressors	turbine_fuel_isGas_2	1	1	0	log
9	Compressors	turbine_fuel_isGas_3	1	1	0	log

Fig. 5.4: Sample file of the file "StatsAttribSettings.csv"

The file consists of seven columns and they are described as follow:

- **CompName**: This column contains the component name. Options are all component names as introduced in Chapter 2.
- **AttribName**: This column contains the attribute names of the component given under "CompName", that can be part of the heuristic testing process.
- **Features**: This is a boolean value ("0" and "1"), which indicates that the attribute values shall be loaded to be a feature variable ("1") or not ("0").

- **Predictor**: This is a boolean value ("0" and "1"). It indicates if this variable shall be tested ("1") or not ("0"). Attributes with value settings of "1" will be loaded, independent of the settings under "Feature".
- **Convert2DiscreteValue**: This is a boolean value ("0" or "1"). It indicates, if the loaded data consists of strings instead of numbers. If set to "1", a sub-routine is called within the heuristic process as part of the Python code. In this sub-routine, the strings are converted into numbers (dummy variables). Below in Fig. 5.5 an example is given of a column of "gender" entries and "age" values. The columns "age" contains numbers only, whereas the column "gender" contains strings. Here the "gender" entries will need to be converted from strings to boolean values. There are two unique values: "Male" and "Female". Hence, out of the column "Gender", two columns are being generated with labels "Gender_Male" and "Gender_Female". (Three columns would have been generated, if a third unique value would have been found e.g. "indifferent".) Wherever the column "Gender" contained the entry "Male", the new column "Gender_Male" will contain a "1", and a "0" otherwise. The values for the column "Gender_Female" is carried out in the same way, where ever the column "Gender" contained an entry for "Female" an "1" is placed in the column "Gender_Female", and "0" otherwise.

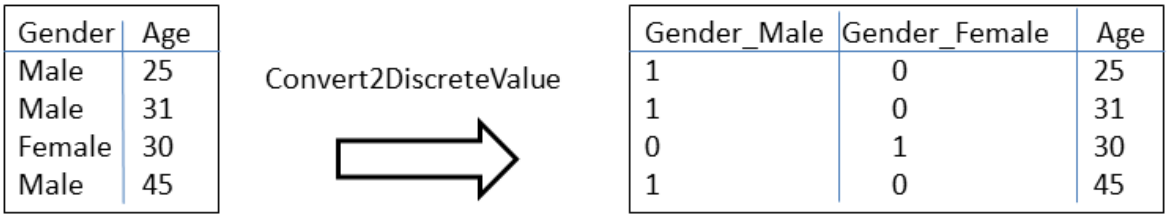


Fig. 5.5: Example of converting strings attributes to number attributes.

However, it is best to input as many attributes as floats or boolean values as possible. E.g. the attribute "is_H_gas" used to be called "gas_type" and used to contain values of "H" and "L". However the current variable "is_H_gas" contains only "1" and "0". This contains the same information, just represented in a different form.

RegressionType This is a string, indicating the regression method to be applied to the data. The following two options are currently implemented:

- **lin**: This stands for "linear regression", and includes the Lasso linear regression, median, min and max sample values.
- **log**: This stands for "logistic regression", and refers to a logistic regression.

Hence, with the above settings, the user can supply all information required for the testing phase, and the user has the option of modifying the testing runs by excluding certain variables. The following section will describe further required steps the user will need to undertake as part of the attribute value generation.

3) Generation of plots for data QA

Having a good understanding of the quality and quantity of the data is very important, before one can apply any heuristic methods for generating missing attribute values. Hence **THE USER NEEDS TO (VISUELLY) INSPECT THE DATA THAT CAN BE USED AS INPUT FOR THE HEURISTIC PROCESSES!**

This is carried out with the function **M_Stats.gen_DataHists**. It requires the following inputs:

- a copy of the network
- a list of components to be visualized
- a relative path to the above setup file "StatsAttribsSettings.csv"
- a relative path to a main folder, where the plots will be stored to. After the plot generation, this folder will contain the following:

- **HistPlots**: This is a subfolder containing further subfolders, one for each component, and each subfolder will contain the corresponding plots. Each plot consists of a scatter-plot of the data, and a histogram, see Fig. 5.6 as an example. In addition, the title contains the name of the attribute next to information in respect of the attribute data density.
- **Overview.png**: This is a file with the name "Overview.png" and contains pair-plots of attributes. (See Fig. 5.7 as an example). Here each attribute of a component is plotted against all other attributes of the same component. This can be used to investigate if there are correlations between individual attributes already.

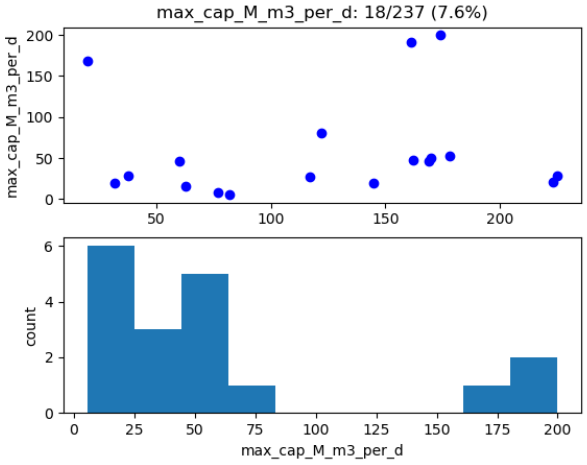


Fig. 5.6: Example histogram plot of the *Compressor* attribute *max_cap_M_m3_per_d*.

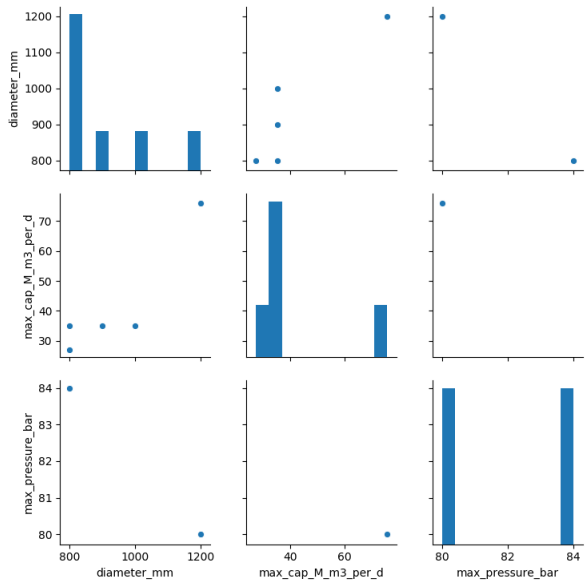


Fig. 5.7: Overview of the mutual attribute relations for the component *Compressors*.

With the help of the plots and data density values, the user can embark on the following steps:

- Correct any wrong data.
- Spend more time in adding more data where data density is low, if possible.

- Select and unselect attributes from the subsequent steps due to data distribution and data density issues, resulting in changes in the setup files.

4) Generation of parameters for the heuristic methods

After the data has been loaded into the Python memory and the setup files have been adjusted, the methods in conjunction with the feature attributes will be used to test to generate missing values (predictors). For this the function **M_Stats.gen_StatsParam** has been generated. It needs the following inputs:

- **Netz**: A copy of the gas component data set.
- **CompNames**: A list of component names for which this process is to be carried out.
- **StatsInputDirName**: A relative path, where both input setup files can be found.
- **DataStatsOutput**: A relative path, where output information will be written to.
- **MaxCombDepth**: This gives the level of the maximum possible feature combinations used for the heuristic process. The term is described with an example as follows. There shall be four different attributes with labels: "A", "B", "C", and "D". There are 10 different combinations, how the attributes could be combined: "A", "B", "C", "D", "AB", "AC", "AD", ..., "ABCD", where here it is stipulated, that "ABD" is equal to "BDA" or "DBA". In addition, each attribute is only allowed to appear ones per combination, meaning "AA" is not an allowed option. So the value supplied via *MaxCombDepth* restricts how many different feature attributes are allowed as input. E.g. for a value of 2, only the following combinations would be allowed: "A", "B", "C", "D", "AB", "AC", "AD", "BC", "BD", and "CD". A smaller *MaxCombDepth* value results in less combinations and hence in less simulations, and reduced computational time, and should be considered when running these processes here. However, more significant is, that larger "MaxCombDepth" can lead to over-fitting.

The output of this function "M_Stats.gen_StatsParam()" will be two fold: additional plots and measures of fit values.

A sample of such a plot is given in Fig. 5.8. Each predictor (here *max_cap_M_m3_per_d*) is plotted against the selected features that were used to determine the predictor attribute (here *max_power_MW*). Here the predictor is plotted on the y-axis, whereas the feature is plotted on the x-axis. The solid line is the estimation of the method used. The title contains information on the method selected, and an R-square value of the fit that was determined using the method.

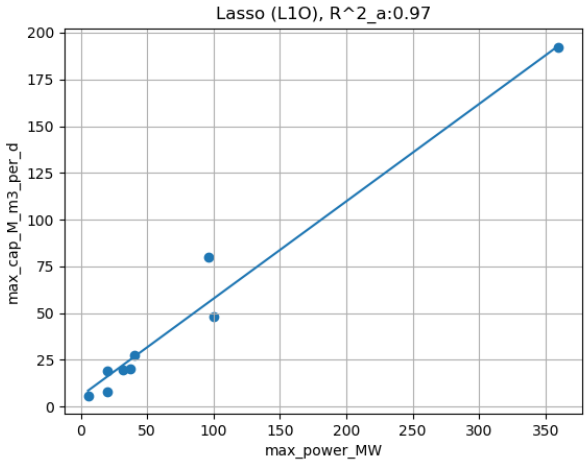


Fig. 5.8: Example of attribute *max_power_MW* versus *max_cap_M_m3_per_d* from the component *Compressors*. The solid line represents the fit of the Lasso method to the data.

In addition to the graphical output, additional simulation information is stored in individual CSV files fore each component. These files are described in more detail below.

An example output file is given in Fig. 5.9 for the component *LNGs*.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	CompName	AttribName	NumElements	ModelName	NumFeatures	FeatureNames	Plots	NumSamples	NumFill	BIC	MeanAbsError	R_2	R_2_adj	ReplaceType	ModelParam
2	LNGs	max_cap_store2pipe_M_m3_per_d	32	Lasso	1	["max_workingGas_M_m3"]	./LNGs/Attril	28	3	135.770712	6.783590107	0.5981701	0.5827151		{"SC_Mean": [2563
3	LNGs	max_cap_store2pipe_M_m3_per_d	32	Lasso	1	["median_cap_store2pipe_M_m3_per_d"]	./LNGs/Attril	21	0	106.326975	7.014177949	0.5128181	0.48717695		{"SC_Mean": [29.3
4	LNGs	max_cap_store2pipe_M_m3_per_d	32	Lasso	2	["max_workingGas_M_m3", "median_cap_store2p	./LNGs/Attril	21	0	108.153992	6.444420605	0.54026	0.48917777		{"SC_Mean": [3076
5	LNGs	max_cap_store2pipe_M_m3_per_d	32	Mean	1	["max_cap_store2pipe_M_m3_per_d"]	./LNGs/Attril	29	3	166.274701	12.37202938	0	-0.03703704		{"SC_Mean": [0], "S
6	LNGs	max_cap_store2pipe_M_m3_per_d	32	Median	1	["max_cap_store2pipe_M_m3_per_d"]	./LNGs/Attril	29	3	166.764196	12.08707833	-0.01702239	-0.05468989		{"SC_Mean": [0], "S
7	LNGs	max_cap_store2pipe_M_m3_per_d	32	Min	1	["max_cap_store2pipe_M_m3_per_d"]	./LNGs/Attril	29	3	194.801311	21.4659326	-1.67426935	-1.77331637		{"SC_Mean": [0], "S
8	LNGs	max_cap_store2pipe_M_m3_per_d	32	Max	1	["max_cap_store2pipe_M_m3_per_d"]	./LNGs/Attril	29	3	235.23244	51.92614793	-9.78173034	-10.1810537		{"SC_Mean": [0], "S
9	LNGs	max_workingGas_M_m3	32	Lasso	1	["max_cap_store2pipe_M_m3_per_d"]	./LNGs/Attril	28	1	1038.39483	72031088.77	0.60179855	0.58648311		{"SC_Mean": [23.1
10	LNGs	max_workingGas_M_m3	32	Lasso	1	["median_cap_store2pipe_M_m3_per_d"]	./LNGs/Attril	21	0	764.014682	69013582.41	0.78651105	0.77527479		{"SC_Mean": [29.3
11	LNGs	max_workingGas_M_m3	32	Lasso	2	["max_cap_store2pipe_M_m3_per_d", "median_ca	./LNGs/Attril	21	0	765.83054	65467405.92	0.79864345	0.7762705		{"SC_Mean": [27.6
12	LNGs	max_workingGas_M_m3	32	Mean	1	["max_workingGas_M_m3"]	./LNGs/Attril	31	1	1180.27533	149945270.6	0	-0.03448276		{"SC_Mean": [0], "S
13	LNGs	max_workingGas_M_m3	32	Median	1	["max_workingGas_M_m3"]	./LNGs/Attril	31	1	1181.666	147160879	-0.04588166	-0.08194655		{"SC_Mean": [0], "S

Fig. 5.9: Example CSV output of heuristic model results for the component *LNGs*.

These files are written to the folder "*/Ausgabe/Sample/StatsData/*". All generated files start with the name "*RetSummary*" and are followed by the name of the component, separated by an underscore. E.g. for *LNGs* the CSV file name is "*RetSummary_LNGs.csv*".

The files contain information on attributes, methods, errors and parameter settings, where each line is a single run/test result. The columns are as follow:

- **CompName**: Name of the component.
- **AttribName**: Name of the predictor attribute.
- **NumElements**: Number of elements of this component.
- **MethodName**: Name of the method used, that was selected through the setup file **StatsMethodsSettings.csv**.
- **NumFeatures**: Number of features used to estimate the predictor.
- **FeatureNames**: List of feature attribute names, that were used to estimate the predictor.
- **Plots**: A link to the individual plots of the features and attribute relationship, where the hyperlink currently works under Excel on Windows only.
- **NumSamples**: Number of samples of the feature data, which were used as part of this method evaluation (this number can never be larger than the value in column **NumElements**).
- **NumFill**: Number of elements for which the predictor attribute can be simulated with the method, where the attribute had missing values. (This value also includes the value given under **NumSamples**.)
- **BIC**: Indicator for the goodness of fit of the model using the BIC (Bayesian information criterion) value.
- **MeanAbsError**: Measure of goodness of fit of the model using the mean absolute error (MAE).
- **R_2**: R-square model value.
- **R_2_adj**: Adjusted R-square value.
- **ReplaceType**: An empty column that will be used at a later stage.
- **ModelParam**: An entry containing all the fitting parameters used by the methods and attributes, the scaling values of the features attributes ("SC_Mean", and "SC_Scale"), and the method parameters ("Intercept", "Coef").

With the above information, visual and values, the user can make an informed decision in respect of which method could be used with which attribute to fill missing values.

5) Selecting individual simulation methods for each attribute

As described in the above processes, the function **M_Stats.gen_StatsParam** generates plots and CSV files containing information on the goodness of the methods for each attribute. With this information the user can decide which method

could be used with which feature attribute values to generate the missing attribute values.

Hence in the next step the user needs to create a setup file, that contains the settings for methods and attributes that will be used for the real generation of those missing attribute values. For this the user can use the output files from the previous step, by carrying out the following actions:

- Copy the above output the CSV files to a new location.
- Open one of those files after the other, and carry out the next steps for each file:
  - With the information of the graphs and the indicator of goodness of fit values (e.g. BIC), remove all those method attribute combinations, that shall not be used for any heuristic processes.
  - Place one of the following key words into the column "**ReplaceType**":
    - **replace**: Here for all attribute values a new value is being simulated and all values will be replace with the simulated value. Even the original input data will be replaced by the newly simulated values.
    - **fill**: Here only for the missing attribute values will be determined. This means, that original input data will not be overwritten, in contrast to option "replace".

However, independent on the replace type setting, some elements might not get their attribute values filled. This is due to the fact that some feature attribute values required during this process were also missing. Hence the user might select several different methods for the generation of the same attribute, by retaining several different method lines for a single attribute in the CSV file. Here it is important to notice, that the attribute values are generated in the order as they appear in the CSV input file. Hence the user should order the methods in such as way, that the method with the "best" predictions are being carried out first. This could be followed by methods that generate attribute values with a larger errors. To assure that there are no further missing values one could retain the **mean** or **median** method as the last method, filling any values that were left unfilled by any previous estimation.

6) Simulation and filling of attributes values

The actual simulation and filling of the attributes is carried out with the function **M_Stats.pop_Attribs** and is the last step in getting attribute values filled in a gas component data set. This function requires the following input:

- **Netz**: A copy of the component data set.
- **CompNames**: A list of components for which the element's attributes shall be generated and filled.
- **StatsInputDirName**: A relative path name of the location of the above modified setup files.

The return of this function is a component data set, where all attributes have been filled according to the setup files.

5.1.3 Summary

Here a method pathway has been described, that attempts to find heuristic methods that can be used to fill missing attribute values. This is a complex process, where the Python code generates plots and model output values, that need to be considered by the user. With the information on hand, the user can decide if certain missing attribute values should be estimated using implemented regression methods. Even though the difference between input data and estimated values might be large (large errors), as a first guess this process might be a first good approach in filling missing values.

CHAPTER

SIX

SAMPLE GAS NETWORK DATA SET GENERATION

Gas network data sets are required for the simulation of gas flows. As there are no complete gas network data sets openly available, one needs to merge several different gas component data sets. However, some data sources might only supply information for a single country (e.g. UK) or only for a specific components, such as LNG-terminals. Hence, the main task of the SciGRID_gas project is to generate a merged complete gas network data set, from multiple smaller and most likely incomplete data sources.

Chapter 3 described the raw data sources that can be accessed and introduced source specific tools to read the raw data into Python memory. In addition, Chapter 4 described some tools to load, save data from and to CSV file, next to some visualization tools. Chapter 5 introduced the pathway of how missing attribute values can be generated.

This chapter here will combine all those parts and demonstrate on an example how a single complete gas network data set can be generated from several raw gas data sources. Hence the main focus of this chapter here are the functions written as part of the SciGRID_gas project, the order of those function calls and the parameters that need to be supplied to those functions, so that the user ends up with a generated complete gas network data set.

Introduction to the pathway steps

Several steps need to be taken to combine several gas data sources into a complete gas network data set. This has been summarised in the following graph:

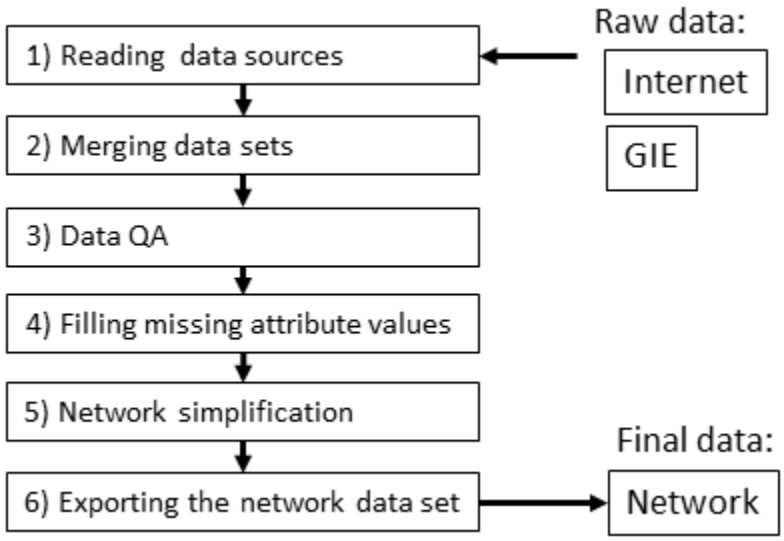


Fig. 6.1: Flow chart of the steps required to generate a complete gas network data set from several data sources.

To describe the individual steps required, two data sets were selected, and will be used as the sample data sets within this remainder of this section. Those two data sets are:

- Internet
- GIE

Other data sets will be included at a later stage.

6.1 Reading data sources

The data can be loaded from their original sources (e.g. API or download of the data), however this can be time consuming. Another option is using pre-processed data, data that has been supplied from the SciGRID_gas project through CSV files. In case that the data can not be supplied, the user might need to access the raw data themself, subsequently save the data as CSV files and load the data from those files at a later stage, as described below.

Converting raw data to SciGRID_gas data

Here an example is given to load the raw GIE data and store the data as CSV files:

```
import Code.M_GIE       as M_GIE

dirNameDestination      = 'Ausgabe/GIE/Data_Raw/'

Netz_GIE                = M_GIE.read()

M_CSV.write(RelDirName = dirNameDestination, Netz_GIE)
```

The command "Netz_GIE = M_GIE.read()" reads in the data for the GIE data source. However, the module "M_GIE" needs to be declared and imported to the Python system, which is done with the first line "import Code.M_GIE as M_GIE". The last line "M_CSV.write(RelDirName = dirNameDestination, Netz_GIE)" writes the data to several CSV files, using the format that is being used throughout the SciGRID_gas project into the folder specified through "dirNameDestination". At a later stage, this pre-processed data can be loaded quickly, whereas the original data access can take a long time (minutes to hours).

Loading SciGRID_gas data

For the loading process of the two sample data sets (InternetData and GIE), the following Python commands need to be executed, as the data will have been supplied as part of the project:

```
RelDirName_GIE        = 'Ausgabe/GIE/Data_Raw/'
RelDirName_Internet   = 'Ausgabe/InternetDaten/Data_Raw/'

Netz_GIE              = M_CSV.read(RelDirName = RelDirName_GIE)
Netz_Internet         = M_CSV.read(RelDirName = RelDirName_Internet)
```

After the execution of those two commands, the two data sets from the two sources are in the Python memory accessible through the following two variable names:

- Netz_GIE
- Netz_Internet.

To verify that the data has been loaded, the user can execute some of the "visualization" tools introduced in Chapter 4.

Now the user is ready for merging elements of different data sources.

6.2 Merging data sets

As previously stated, different data sets from different data sources might describe the same facilities. Hence, individual elements need to be merged. So first of all one needs to understand, which components from which data source can be combined. For this, the user can choose the *all()* function:

```
Netz_GIE.all()
Netz_Internet.all()
```

which will return the number of elements for each component. Values have been summarized in Table 6.1, where both data sources supply elements for the same component. In addition, the "InternetData" data source contains further elements for the components *BorderPoints*, *Compressors*, *EntryPoints*, *InterConnectionPoints* and *PipeSegments*.

Table 6.1: List of components with number of elements, where the data sources "InternetData" and "GIE" both contain data.

Components	InternetData	GIE
LNGs	32	21
Nodes	739	98
Storages	197	99

As one can see, there are three components, that are in both data source: *LNGs*, *Nodes*, and *Storages*.

The following steps are being carried out:

LNGs

Here, the "InternetData" and the "GIE" data sources contain elements for the component *LNGs*, with 32 elements from the InternetData and 21 elements from the GIE data source. The above described Python functions **getMatch_Name** and **getMatch_LatLong** were used to determine, if any elements from those two data sources are describing the same facility. Here a threshold of 20 was selected, as external testing resulted in best matches between the two data sources.

Storages

For *Storages*, both data sources contain elements. The functions **getMatch_Name** and **getMatch_LatLong** were used in the merge process. Tests were carried out and it was found that when merging the "GIE" with the "InternetData" data source, an averaged threshold of 30 was best.

Nodes

For the component *Nodes*, both data sources contained elements, and the following methods were applied in the merge process: **getMatch_Name** and **getMatch_LatLong**. When merging the "GIE" *Nodes* with the "InternetData" *Nodes*, a threshold of 30 gave best results.

6.2.1 The merge process

Now the GIE and InternetData data sets need to be stored into the Python dictionary. This is carried out by the following process:

```
sourceList             = ['InterNet',  'GIE']

DataDict              = {}
DataDict['InterNet'] = Netz_InterNet
DataDict['GIE']      = Netz_GIE
```

The actual merging process is carried out with the following function call:

```
CompNames2Join  = ['LNGs','Storages', 'Nodes']
ThresValues     = [20, 30, 30]
Netz_Merged     = JoinNetz.join(data_sources = DataDict, scenaName = 'Scen_1',␣
↪CompNames = CompNames2Join, ThresValues = ThresValues)
```

This function returns the merged gas component data set, where the computational process time could take several minutes. The resulting merged network data set can be saved to a CSV file with the following command:

```
dir_Data_Raw_Merged = 'Ausgabe/GeneratedNetz/02_NonOSM/Data_Merged/'

M_CSV.write(dir_Data_Raw_Merged, Netz_Merged)
```

6.2.2 Results after merging process

After all those merging processes, the resulting network data set contains the following number of elements for each component, as summaries in Table 6.2.

Table 6.2: Resulting merged component data set component summary.

Components	Merged network
LNGs	32
Nodes	739
Storages	197

As can be seen, the number of elements for each component did not increase (compare with Table 6.1), meaning that the elements for components *LNGs*, *Nodes* and *Storages* were complementing each other.

So why was the merge process initiated? To come to an answer, one needs to have a look at the number of non-missing attributes values. As an example the results for the component *Storages* will be presented. The following table (see Table 6.3) depicts which data source has how many non-missing attribute values. The merged data set is also been included:

Table 6.3: Number of non-missing attribute values for component *Storages* for the data sources "InternetData", "GIE" and the resulting merged data set.

Attribute	InternetData	GIE	Merged
number of elements	197	99	197
access_regime	176	–	176
end_year	5	–	5
is_H_gas	24	–	24
is_onShore	188	–	188
max_cap_pipe2store_M_m3_per_d	151	74	161
max_cap_store2pipe_M_m3_per_d	151	71	161
max_workingGas_M_m3	158	76	166
operator_name	195	–	195
start_year	192	–	192
store_type	185	–	185
eic_code	–	99	99
facility_code	–	99	99

By merging the "GIE" *Storages* component data set with the "InternetData" data source, one increases the number of non-missing attribute values for several attributes, e.g. for *max_cap_pipe2store_M_m3_per_d* the number of attribute values increased from 151 to 161 of non-missing values. Hence one was able to fill 10 missing values.

The resulting merged data set, which will be referred to as "Netz_Merged", is depicted for Europe in the following map (Fig. 6.2).

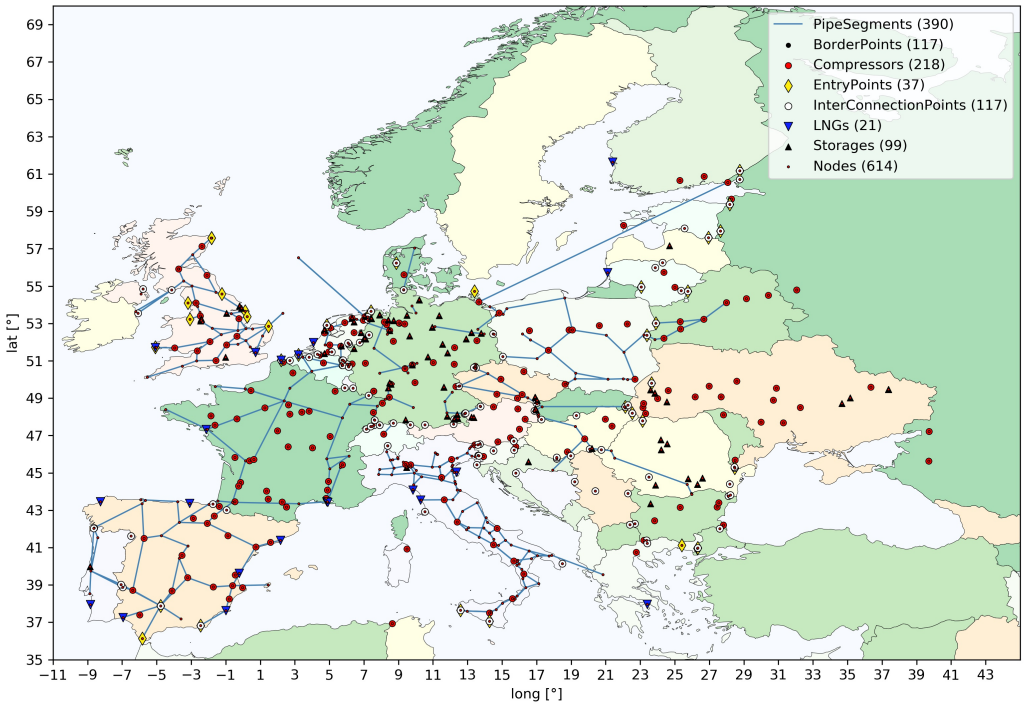


Fig. 6.2: Map of raw data forming a gas component data set.

Even though merging data sets can reduce the number of missing values, the resulting data set still contains a large number of missing values. Hence additional data needs to be generated. For the example of *Storages*, the attribute *max_workingGas_M_m3* only consists of 166 non-missing attribute values from a total of 197 elements. An in depth description of heuristic data generation has been presented in Chapter 5. Here the function calls will be listed and applied to the merged data set.

6.3 Data QA of attribute values

Now that the data has been merged into one data sets, one needs a good understanding of the quality and quality of the data. As the source data sets will have contained incomplete data, where some or most attribute values are missing, so will the resulting merged data set. The missing values need to be filled. However, to do this, one needs input data of good quality. Hence, the data that could be used for the generation of the missing values needs to be "checked". With "checking" two separate aspects are meant:

- data density (number of elements, number of non-None values)
- data distribution (Gaussian, random, histogram).

As previously described, the user has the option via the setup files to specify which attributes are to be included in the data QA process (see Section 5.1.2). The command to be executed so that the plots for the data QA are being generated are as follow:

```
StatsInputDirName  = 'Ausgabe/GeneratedNetz/02_NonOSM/Method_Testing'
DataStatsOutput    = 'Ausgabe/GeneratedNetz/02_NonOSM/StatsData'

M_Stats.gen_DataHists(Netz_Merged, StatsInputDirName = StatsInputDirName,␣
↪DataStatsOutput = DataStatsOutput)
```

In the example given here, all the graphs are saved to subfolders within the above given folder 'Ausgabe/Sample/StatsData' and can be viewed by the user. Examples of the plots that are being generated through this process have been presented as Fig. 5.6 and Fig. 5.7. The location of those files is being supplied as argument *DataStatsOuput* to the function call.

6.4 Filling missing attribute values

Prior to generating missing values, different regression methods in combination with different input attribute value data sets need to be tested, as was described in Section 5.1.2. For this a further setup file needed to be populated, as was described in the same section.

6.4.1 Testing of parameters for the heuristic methods

After having populated the setup file as described in Section 5.1.2, the user can start the testing process by executing the following commands:

```
import Code.M_Stats    as M_Stats

DataStatsOutput    = 'Ausgabe/GeneratedNetz/02_NonOSM/StatsData'
StatsInputDirName  = 'Ausgabe/GeneratedNetz/02_NonOSM/SetupFiles'
CompNames          = ['Compressors', 'LNGs','Storages', 'PipeSegments']

M_Stats.gen_StatsParam(Netz_Merged, CompNames = CompNames, SavePlots = True,␣
↪MaxCombDepth = 1, StatsInputDirName  = StatsInputDirName, DataStatsOutput =␣
↪DataStatsOutput)
```

In the next step, as described in the heuristic process pathway section, the user needs to explore the attribute data with the help of the generated histograms plots (see Fig. 5.8). Further the executed "M_Stats.gen_StatsParam" process generated output files (see Fig. 5.9), containing method test results for each attribute, including goodness of fit values such as the R-square and adjusted R-square value. A single file was created for each component. With the information at hand the user needs to adjusts those files, specifying which attribute value should be generated by which method. This process converts the output files into input files. this process was described in detail in Section 5.1 and hence will not be repeated here.

However, here the best combination of methods and feature attributes is given (as of 21-01-2020) that were used for the sample merged data set. There are four tables (Table 6.4 to Table 6.7) for the components *Compressors*, *LNGs*, *PipeSegments*, and *Storages*.

Table 6.4: List of setup setting for *Compressors* attributes.

AttribName	MethodName	FeatureName
max_pressure_bar	Lasso	max_power_MW
max_pressure_bar	Median	max_pressure_bar
max_cap_M_m3_per_d	Lasso	max_power_MW
max_cap_M_m3_per_d	Lasso	turbine_power_1_MW
max_cap_M_m3_per_d	Lasso	num_turb
max_cap_M_m3_per_d	Mean	max_cap_M_m3_per_d
max_power_MW	Lasso	max_cap_M_m3_per_d
max_power_MW	Lasso	turbine_power_1_MW
max_power_MW	Mean	max_power_MW
num_turb	Median	num_turb
turbine_power_1_MW	Lasso	max_power_MW
turbine_power_1_MW	Mean	turbine_power_1_MW
turbine_power_2_MW	Lasso	max_power_MW
turbine_power_2_MW	Median	turbine_power_2_MW
turbine_power_3_MW	Lasso	max_power_MW
turbine_power_3_MW	Median	turbine_power_3_MW

Table 6.5: List of setup setting for *LNGs* attributes.

AttribName	MethodName	FeatureName
max_cap_store2pipe_M_m3_per_d	Lasso	max_workingGas_M_m3
max_workingGas_M_m3	Lasso	max_cap_store2pipe_M_m3_per_d
max_workingGas_M_m3	Lasso	median_cap_store2pipe_M_m3_per_d
max_workingGas_M_m3	Mean	max_workingGas_M_m3
median_cap_store2pipe_M_m3_per_d	Lasso	max_cap_store2pipe_M_m3_per_d

Table 6.6: List of setup setting for *PipeSegments* attributes.

AttribName	MethodName	FeatureName
diameter_mm	Lasso	max_cap_M_m3_per_d
diameter_mm	Mean	diameter_mm
is_H_gas	LogisticReg	diameter_mm
is_bothDirection	LogisticReg	diameter_mm
max_cap_M_m3_per_d	Lasso	diameter_mm
max_cap_M_m3_per_d	Mean	max_cap_M_m3_per_d
max_pressure_bar	Lasso	diameter_mm
max_pressure_bar	Median	max_pressure_bar

Table 6.7: List of setup setting for *Storages* attributes.

AttribName	MethodName	FeatureName
max_workingGas_M_m3	Mean	max_workingGas_M_m3
max_cap_store2pipe_M_m3_per_d	Lasso	max_workingGas_M_m3
max_cap_store2pipe_M_m3_per_d	Mean	max_cap_store2pipe_M_m3_per_d
is_H_gas	LogisticReg	max_cap_pipe2store_M_m3_per_d

The “ReplaceType” column value was set to “fill” for all attributes.

6.4.2 Simulation and filling of attribute values

After having transformed those files into setup files, they are being used in the next step of generating missing attribute values, by executing the following commands:

```
import Code.M_Stats    as M_Stats

SettingsFileName    = ('Ausgabe/GeneratedNetz/02_NonOSM/SetupFiles/Value_Generation/
↪RetSummary_LNGs.csv')
SimSettings         = M_CSV.read_CSV_raw(SettingsFileName)
Netz_Merged         = M_Stats.pop_Attribs(Netz_Merged, CompNames = [compName],␣
↪SimSettings = SimSettings)

SettingsFileName    = ('Ausgabe/GeneratedNetz/02_NonOSM/SetupFiles/Value_Generation/
↪RetSummary_Compressors.csv')
SimSettings         = M_CSV.read_CSV_raw(SettingsFileName)
Netz_Merged         = M_Stats.pop_Attribs(Netz_Merged, CompNames = [compName],␣
↪SimSettings = SimSettings)

SettingsFileName    = ('Ausgabe/GeneratedNetz/02_NonOSM/SetupFiles/Value_Generation/
↪RetSummary_PipeSegments.csv')
SimSettings         = M_CSV.read_CSV_raw(SettingsFileName)
Netz_Merged         = M_Stats.pop_Attribs(Netz_Merged, CompNames = [compName],␣
↪SimSettings = SimSettings)

SettingsFileName    = ('Ausgabe/GeneratedNetz/02_NonOSM/SetupFiles/Value_Generation/
↪RetSummary_Storages.csv')
SimSettings         = M_CSV.read_CSV_raw(SettingsFileName)
Netz_Merged         = M_Stats.pop_Attribs(Netz_Merged, CompNames = [compName],␣
↪SimSettings = SimSettings)

Netz_Component = Netz_Merged
```

As each component has its own setup file, the attribute values generation function (*pop_Attribs*) was called repeatatly for all components.

For some attributes, no estimation method could be set up, as it was not possible to determine meaningful relations between those attribute and other attributes. Hence for those final missing attribute values, a user supplied value will be applied, which is described in the following sub-section.

6.4.3 Further attribute filling

Here a brief process is demonstrated, how attribute values can be filled with a single user specified values per attribute.

Here the attribute "start_year" shall be set to a value of "1983" whereas the attribute "end_year" shall be set to a value of "2050". In addition, for all elements of type *Compressors*, the power source of the turbines, given through *turbine_fule_isGas_1* (going up to a value of 6) is to be set to "1". This can be achieved with the following commands:

```
Netz_Component.make_Attrib([], '',    'end_year',   'const', 2050)
Netz_Component.make_Attrib([], '',    'start_year', 'const', 1983)
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_1', 'const', 1,
↪'fill')
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_2', 'const', 1,
↪'fill')
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_3', 'const', 1,
↪'fill')
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_4', 'const', 1,
↪'fill')
```

(continues on next page)

(continued from previous page)

```
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_5', 'const', 1,
↪'fill')
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_6', 'const', 1,
↪'fill')


Netz_Component = Netz_MergedPartFilled

M_CSV.write(dir_Data_Raw_Component, Netz_Component)
```

After all required attribute values have been determined, the network needs simplification, as some components are not connected to pipe-segments. This is described in the following text.

6.5 Network simplification

Up to this point, the data set that has been created is, labelled a "gas component data set". In a component data set, some of the elements are not connected to pipe-segments. Those elements either need to be moved to a start or an end location of a pipe-segment or they need to be removed from the data set. The resulting data set is termed a "gas network data set".

Hence, an additional Python function, that has been created as part of the SciGRID_gas project, can be applied. It is named "moveComp2Pipe". This function moves the geographic location to a nearby pipe-segment start or end node, if the distance moved is less than the value specified by the user. Here, a maximum value of $100km$ will be used as the max distance. The following code line carries out such a movement for the *Compressors* component elements:

```
import Code.M_Shape    as M_Shape

Netz_Component  = M_Shape.moveComp2Pipe(Netz_Component.copy2(), compName =
↪'Compressors', pipeName = 'PipeSegments', maxDistance_km = 100.0)
```

The main line of code calls the function "moveComp2Pipe". A copy of the network is supplied to the function ("Netz_Component.copy2()"), and three further parameters are also supplied. These are:

- **compName**: This is the name of the component elements.
- **pipeName**: Here the elements of the component *PipeSegments* are the elements, to which the compressor elements will be moved to.
- **maxDistance_km**: Moving of the *Compressors* elements shall be carried out, if the distance between those element location and a start or an end location of a pipe-segment is less than value given in units of km.

This process will also need to be carried out for the other component:

- *Storages*
- *ConnectionPoints*
- *EntryPoints*
- *BorderPoints*
- *LNGs*.

This is achieved by replacing the component name *Compressors* in the function call with the names from the list above. The Fig. 6.3 below shows the resulting complete gas transmission network data set.

As can be seen a significant number of elements were moved and removed. The final gas network data set consists of data, where all elements are connected via pipe-segments and are summarized in the Table 6.8.

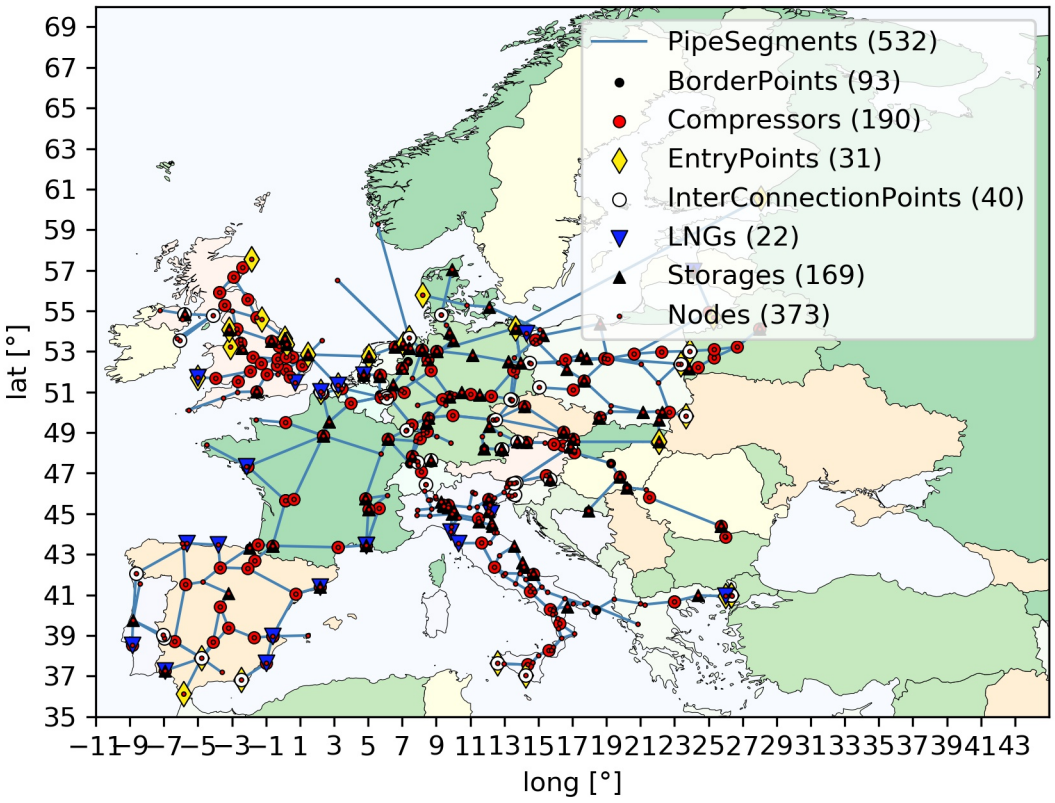


Fig. 6.3: Map of final gas network data set.

Table 6.8: Summary of data of the seven sample pipelines from Fig. 5.1.

Component name	Number of elements prior to moving and removing process	Number of elements after moving and removing process
EntryPoints	37	31
InterConnectionPoints	117	40
Storages	197	169
LNGs terminals	32	22
Compressors	237	190
PipeSegments	532	532

6.6 Exporting the network data set

As a last step the generated gas network data set needs to be saved to a file, where different methods have been discussed in Section 4.1.

To save the resulting gas network data set into CSV files, the following commands needs to be executed:

```
dir_Data_Raw_Final  = 'Ausgabe/GeneratedNetz/02_NonOSM/Data_Final/'

M_CSV.write(dir_Data_Raw_Final, Netz_Component)
```

Now the complete gas network data set has been saved to a CSV file and can be used for gas models.

Example Code/Settings The snippets of code that were listed throughout this chapter here are collated into a single code block, that is listed in the Appendix 9.7. In addition the code is also available as a Jupyter Notebook file, which will be part of the download from the SciGRID_gas GitHub [02_NonOSM]

Summary of generating a gas network data set

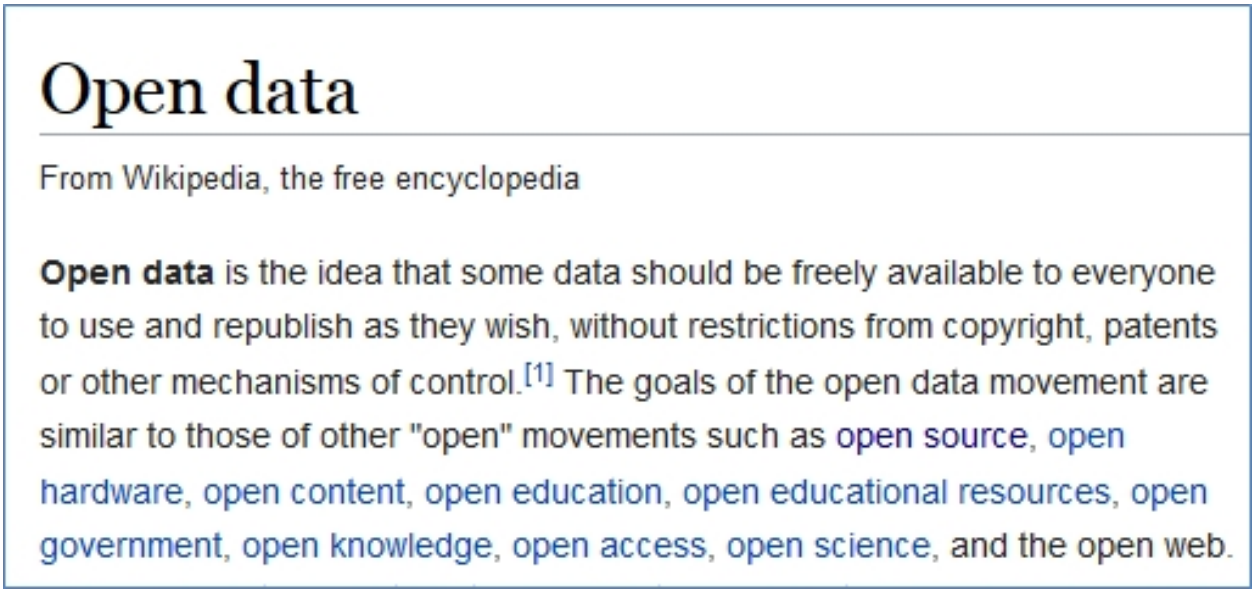
In this chapter two different data sources were merged: the "InternetData" and the "GIE" data sets. They contained complementary information for the components *LNGs*, *Storages*, and *Nodes*. This chapter demonstrated how those two data sets can be merged and missing values be generated using heuristic processes and user input. Further, all elements were connected or removed from the data set. The result was a gas network data set, that is usable in gas flow models.

CHAPTER

SEVEN

COPYRIGHT

SciGRID_gas is an open source gas data model, using open data. Wikipedia defines open data is given in Fig. **??**:



Open data

From Wikipedia, the free encyclopedia

Open data is the idea that some data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control.[1] The goals of the open data movement are similar to those of other "open" movements such as open source, open hardware, open content, open education, open educational resources, open government, open knowledge, open access, open science, and the open web.

Further Wikipedia defines Open Source model is given in Fig. **??**.

With the above definitions, it should be obvious, that data or processes that are not open source cannot be combined with open source data, and hence cannot be made available freely and without any limitations. And data from the general internet can be freely available, but might not have a license, that would allow for the data to be distributed to others. The Wikipedia definition of open-source license is given in Fig. **??**.

Even though SciGRID_gas is an open source model, meaning that only open source data can be passed on with the SciGRID_gas environment to external users, some information does not need to have this required level of openness. Some data might be copy right protected, but can be implemented in some of the SciGRID_gas processes. E.g. the information on the compressor station power values might be copy right protected, and even though supplied to SciGRID_gas is not allowed to be passed through to other SciGRID_gas users. However, such information can be used during the heuristic SciGRID_gas process, and results from that process can be passed on to users. However the following can be put in place:

- One passes on the location of the data that is copy right protected, e.g. via URL link.
- This data can be downloaded by each user, as this is allowed.
- The user is given the heuristic processes.
- The user can apply the heuristic processes on the downloaded data, and generate new data or use the raw data.

Open-source model

From Wikipedia, the free encyclopedia

> This article **possibly contains synthesis of material which does not verifiably mention or relate to the main topic.** Relevant discussion may be found on the talk page. *(March 2016) (Learn how and when to remove this template message)*

The **open-source model** is a decentralized software development model that encourages open collaboration.[1][2] A main principle of open-source software development is peer production, with products such as source code, blueprints, and documentation freely available to the public. The open-source movement in software began as a response to the limitations of proprietary code. The model is used for projects such as in open-source appropriate technology,[3] and open-source drug discovery.[4][5]

Open source promotes universal access via an open-source or free license to a product's design or blueprint, and universal redistribution of that design or blueprint.[6][7] Before the phrase *open source* became widely adopted, developers and producers used a variety of other terms. *Open source* gained hold with the rise of the Internet.[8] The open-source software movement arose to clarify copyright, licensing, domain, and consumer issues.

Open-source license

From Wikipedia, the free encyclopedia

An **open-source license** is a type of license for computer software and other products that allows the source code, blueprint or design to be used, modified and/or shared under defined terms and conditions.[1][2] This allows end users and commercial companies to review and modify the source code, blueprint or design for their own customization, curiosity or troubleshooting needs. Open-source licensed software is mostly available free of charge, though this does not necessarily have to be the case. Licenses which only permit non-commercial redistribution or modification of the source code for personal use only are generally not considered as open-source licenses. However, open-source licenses may have some restrictions, particularly regarding the expression of respect to the origin of software, such as a requirement to preserve the name of the authors and a copyright statement within the code, or a requirement to redistribute the licensed software only under the same license (as in a copyleft license). One popular set of open-source software licenses are those approved by the Open Source Initiative (OSI) based on their Open Source Definition (OSD).

- The only restriction remains, that the user is not allowed to pass on the copy right protected data. However any results generated with the copy right protected data can be passed on.

Source is the other important information within the SciGRID_gas data sets. With "source" we mean, where does this data come from, so that another person receiving data could navigate to the same location in the internet and verify the information supplied. Hence most likely this will be an URL link, but might also be links to folders containing documents, that have been downloaded already, and will be part of the SciGRID_gas download.

CHAPTER

EIGHT

INSTALLATION

The SciGRID_gas project aims to develop a tool set, so that an open source gas transmission-network data set can be generated. For this the coding language Python was selected. Python is an open free coding language that can be executed on Windows and Linux computers. Hence a large set of Python code has been written that reads in, downloads, and converts gas data and merges the data into a single data set. In addition heuristic methods have also been coded to generate data for the gas transmission network data set. However the fundamental environment this has been designed in is Python. However, additional functionality is being used from other Python tools, such as *scipy*, *fuzzywuzzy* and *pyshp* to list just a few. These need to be installed into the users Python environment. The list of additional packages is given later.

Part of generating the gas transmission network data set, is the requirement, that the user will need to access third party data. The SciGRID_gas project has written tools to access the data, however individual access rights are required for each user. Hence part of this subsequent documentation is a list of tasks, that each user needs to carry out, so that the supplied code can access the data of interest.

In addition, there is a SciGRID_gas data base, that might be part of the SciGRID_gas data project. Some documentation has been written in the section *Make File*.

8.1 Install of Python environment

To run SciGRID_gas, one needs to install Python, version 3.6 or higher. You can choose Anaconda 2019.03 for Windows or any other environment. Here, we have been running our Python within Spyder 3.3.2.

When using Anaconda, one can use pip within the "Anaconda Prompt" environment.

SciGRID_gas uses a large set of code from third parties. This functionality needs to be linked with your Python runtime environment. The packages are operating system independent, and include libraries, such as: matplotlib, scipy, networkx 2.3, and pyproj. A pip install requirements file has been created and added to the project, so that all required libraries, including the correct version numbers, can be installed with a single command.

The file is named "**requirements.txt**", and is located in the main directory of the SciGRID_gas project. By running the command, all libraries, including underlying libraries that are required are being installed. The command for a unix environment is as follow:

:: pip install -r requirements.txt

whereas for a Windows computer, running Spyder 3.2, the following command should be used:

:: pip install -r requirements_pip.txt conda install -r requirements_conda.txt

There are no known packages, that need to be uninstalled, so that SciGRID_gas can be executed under Python.

CHAPTER

NINE

APPENDIX

9.1 Glossary

The glossary terms can be found in Table 9.1.

Table 9.1: Glossary

Name	Abbreviation	Description	Units
component		A gas network consists of different components, such as pipelines, compressors LNG terminals and more. However, for a gas transmission network, there is a handful of components only: pipeline, compressor, LNG terminal, storage, entry point, border point, connection point, consumer, node, and production	
element		Elements are instances of component. Hence we speak of 10 compressor elements, if we have a data set that has 10 compressors. Here then we can refer to the first or the last or any element of such component	
attribute		Gas facilities such as pipelines or compressors can be described with a large number of parameters, such as pipeline diameter, or compressor capacity. Those parameters are referred to as attributes. Hence each component has a list of properties, which are different from one component to another component	
facility		General term used for a gas appliance, such as compressor element, or LNG terminal	
pipeline		This is a gas pipeline entity, which has one start and one end point, however can run via many nodes, compressors and other gas network elements	
pipe-segment		This is a gas pipeline, that has only one start and one end point, but no nodes in-between, hence only goes from one point to the next point	
LNG	LNG	Liquefied natural gas	
CNG	CNG	Compressed natural gas	
flow duration curve	FDC	It is the cumulative frequency curve that shows the percent of time specified flow were equal or exceeded during a given period. The information on occurrence of events is lost	
Energiewende		German term for the change in using primary energies, the move away from coal to renewable energies, such as wind or solar	
gas component data set		Raw input data, associated with components of the gas transmission grid	
gas network data set		Output data, a coherent network of gas transmission components	
OSM	OSM	Data that is available from the openstreetmap.org	
non-OSM	Non-OSM	Data that is not part of the OSM data set	
gas type		There are two types of gas High (H) and Low (L) calorific gas	
mean absolute error	MAE	mean difference between input values and estimated values	
data density		This is the ratio of the number of usable (not missing) attribute values over number elements of the component	
Transmission System Operators	TSO	This is an entity entrusted with the transportation of natural gas/electricity, as defined by the European Union	
gas transmission network		This describes the physical gas transmission grid, however excludes any facilities/components that would be part of a distribution network and their facilities. This projects goal is to create an open source gas network data set that can be used to describe the European gas transmission network	

9.2 Unit conversions

Table 9.2: Unit conversions

From Unit	To Unit	MultiVal
LNG Mt	LNG Mm^3	2.47
gas tm^3/h	gas Mm^3/d	24/1000
LNG Mm^3	gas Mm^3	584
LNG Mt	gas Mm^3	1442.48

9.3 References

[02_NonOSM]: https://github.com/jdlr01/OpenSciGRID_gas

[6EnForProBunReg]: (https://www.bmwi.de/Redaktion/DE/Artikel/Energie/Energieforschung/energieforschung-6-energieforschungsprogramm.html)

[BMWi]: (https://www.bmwi.de/Navigation/DE/Home/home.html)

[Energiewende]: (https://www.bundesregierung.de/Webs/Breg/DE/Themen/Energiewende/_node.html)

[GIE_MainRef]: GIE: Gas Infrastructure Europ, https://agsi.gie.eu.

[LKD_MainRef]: Friedrich Kunz, Mario Kendziorski, Wolf-Peter Schill, Jens Weibezahn, Jan Zepter, Christian von Hirschhausen, Philipp Hauser, Matthias Zech, Dominik Möst, Sina Heidari, Björn Felten and Christoph Weber: "Electricity, Heat, and Gas Sector Data for Modeling the German System", 2017, Data Documentation 92, lk DEU: langfristige Planung und kurzfristige Optimierung des Elektrizitaetssystems in Deutschland im europaeischen Kontext.

[Kunz2017]: Friedrich Kunz, Mario Kendziorski, Wolf-Peter Schill, Jens Weibezahn, Jan Zepter, Christian von Hirschhausen, Philipp Hauser, 'Electricity, Heat, and Gas Sector Data for Modeling the German System', Deutsches Institut für Wirtschaftsforschung, Daten Documentation 92, 2017 Matthias Zech, Dominik Möst, Sina Heidari, Björn Felten and Christoph Weber

[OSM]: (https://www.openstreetmap.de/)

[Python_DC]: https://pythontips.com/2013/08/02/what-is-pickle-in-python/.

[RefTSO]: https://en.wikipedia.org/wiki/Transmission_system_operator.

[Schmidt2017]: Martin Schmidt , Denis Aßmann, Robert Burlacu, Jesco Humpola, Imke Joormann, Nikolaos Kanelakis, Thorsten Koch, Djamal Oucherif, Marc E. Pfetsch, Lars Schewe, Robert Schwarz and Mathias Sirvent, 'GasLib—A Library of Gas Network Instances', 2017, MDPI, Data, MSC: 90-08; 90C90; 90B10, DOI: 10.3390/data2040040.

[SciGRID_power]: (https://power.scigrid.de)

[scikit_Reg]: https://scikit-learn.org/stable/modules/linear_model.html

[WikiJackKnifeResampling]: https://en.wikipedia.org/wiki/Jackknife_resampling.

[WikiLeavePOutCrossValidation]: https://en.wikipedia.org/wiki/Cross-validation_(statistics)#Exhaustive_cross-validation.

[WikiLeaveOneOutCrossValidation]: https://en.wikipedia.org/wiki/Cross-validation_(statistics)#Exhaustive_cross-validation.

9.4 References for InternetData data set

Below a list of those sources used to generate the InternetData data set.

- Google (n.d.). [Google Maps of Europe]. Retrieved between 15.July.2018 and 31.June.2021
- http://belarus-tr.gazprom.ru/
- http://corporate.vattenfall.com/about-vattenfall/operations/market-transparency/gas-storage/
- http://en.gaz-system.pl//
- http://gaslager.energinet.dk/EN/Pages/default.aspx
- http://interfaxenergy.com/article/19138/lng-better-than-norway-pipeline-ex-polish-pm
- http://ir.gasplus.it/home/show.php?menu=00002
- http://italgasstorage.it/eng/progetto.html
- http://media.edfenergy.com/Misc/AboutUs.aspx
- http://mmbf.hu/en/company/gas_storage
- http://mndgsgermany.com/
- http://mysolar.cat.com/cda/files/870985/7/Solar_Turbines_5000_Gas_Compressor_News_Rel_Sent.pdf
- http://sse.com/whatwedo/ourprojectsandassets/thermal/Aldbrough/
- http://sse.com/whatwedo/wholesale/gasstorage/
- http://www.bayernugs.de/4-1-Home.html
- http://www.bendisenergy.com.tr/tr/Projelerimiz/24-toren-dogalgaz-depolama-ve-madencilik-as
- http://www.berliner-erdgasspeicher.de/en/Pages/default.aspx
- http://www.botas.gov.tr/
- http://www.bulgartransgaz.bg/en/pages/transstorge-110.html
- http://www.caythorpegasstorage.com/caythorpe/
- http://www.centrica-sl.co.uk/
- http://www.ceskaplynarenska.cz/en/ultimate-speed-underground-gas-storage
- http://www.dea-speicher.de/en
- http://www.depomures.ro/
- http://www.edisonstoccaggio.it/en
- http://www.ekb-storage.de/de/home/
- http://www.emplpipeline.com/en/the-gas-pipeline/
- http://www.emplpipeline.com/en/the-gas-pipeline/,Extra
- http://www.enagas.es/enagas/en/Transporte_de_gas/Almacenamientos_Subterraneos
- http://www.energystock.com/
- http://www.ewe-gasspeicher.de/english/index.php
- http://www.fluxys.com/belgium/en/Services/Storage/Storage
- http://www.gasnaturalfenosa.com/en/activities/lines+of+business/1285338591925/supply+and+transportation+of+gas.html

- http://www.gasspeicher-hannover.de/startseite.html
- http://www.gasstorage.cz/en/operation-information/available-firm-capacity/
- http://www.gasstoragebergermeer.com/gas-storage-bergermeer-2/
- http://www.gastrade.gr/en/the-company/the-project.aspx
- http://www.gas-union-storage.de/
- http://www.gatewaystorage.co.uk/
- http://www.gazprom.com/about/production/underground-storage/
- http://www.geogastock.it/ITA/Home.asp
- http://www.grtgaz.com/en/major-projects/beynes-compressor-station/presentation/news/compressor-station-at-the-beynes-site.html
- http://www.grtgaz.com/fileadmin/plaquettes/en/2017/Essentiel-plaquette-institutionnelle-EN-2017.pdf
- http://www.gsa-services.ru/
- http://www.halite-energy.co.uk/our-project/project-overview/
- http://www.hradf.com/en/portfolio/south-kavala-natural-gas-storage
- http://www.humblyenergy.co.uk/about-us#useful-information
- http://www.islandmageestorage.com/
- http://www.kge-gasspeichergesellschaft.de/
- http://www.kgsp.co.uk/
- http://www.kingstreetenergy.com/
- http://www.kinsaleenergy.ie/gas-storage.html
- http://www.le.lt/index.php/projects-in-progress/syderiai-underground-gas-storage/535
- http://www.lg.lv/index.php?id=3376&lang=eng
- http://www.magyarfoldgaztarolo.hu/en/Lapok/default.aspx
- http://www.mnd.eu/en/2014-11-26-12-13-02/mnd-group-companies
- http://www.nafta.sk/en/about-gas-storage
- http://www.nam.nl/en/about-nam/facts-and-figures.html
- http://www.omv.com/portal/01/com/gas/storage
- http://www.petroceltic.com/operations/bulgaria.aspx
- http://www.pgnig.pl/reports/annualreport2012/en/ar-obrot-magazynowanie-2.html
- http://www.pozagas.sk/en/?PHPSESSID=8d83cc7890abb7980f6793cf56633a72
- http://www.psp.hr/home
- http://www.rag-energy-storage.at/en.html
- http://www.romgaz.ro/en/content/ugs-n-366-new-gas-storage-facility-romania
- http://www.romgaz.ro/en/content/ugs-n-371-sarmasel-storage-facility-upgrading
- http://www.romgaz.ro/en/inmagazinare
- http://www.rwe.com/web/cms/de/37110/rwe/presse-news/pressemitteilungen/?pmid=4005467,http://digitalnewsservice.net/clients/net4gas-nimmt-neue-hochdruck-gas-pipeline-gazelle-in-betrieb/

- http://www.rwe.com/web/cms/en/531750/rwe-gasspeicher/
- http://www.scottishpower.com/pages/hatfield_moor_gas_storage_facility.asp
- http://www.snam.it/en/transportation/Thermal_Year_Archive/Thermal_Year_2013_2014/Info-to-users/index.html
- http://www.sppstorage.cz/#
- http://www.srbijagas.com/naslovna.1.html
- http://www.stogit.it/en/index.html
- http://www.storengy.com/countries/deutschland/en/products-services.html
- http://www.storengy.com/countries/france/en/
- http://www.storengy.com/countries/unitedkingdom/en/oursites.html
- http://www.taqaglobal.com/our-regions/netherlands/gas-storage/peak-gas-installation/overview?sc_lang=en
- http://www.terranets-bw.de/en/gas-transmission/we-transport-your-natural-gas/
- http://www.thueringerenergie.de/Unternehmen/Ueber_uns/Geschaeftsfelder/Speicher.aspx
- http://www.tigf.fr/en/what-we-can-offer/storage.html
- http://www.tpao.gov.tr/eng/?tp=m&id=31
- http://www.tpao.gov.tr/eng/?tp=m&id=84
- http://www.trianel-gasspeicher.com/
- http://www.ugs-katharina.de/en/unternehmen.html
- http://www.vng-gasspeicher.de/content/en/Speicher/index.html
- https://de.wikipedia.org/wiki/Baumgarten_an_der_March
- https://de.wikipedia.org/wiki/Erdgasleitung_Jamal%E2%80%93Europa,https://en.wikipedia.org/wiki/Yamal%E2%80%93Europe_pipeline
- https://de.wikipedia.org/wiki/Hungaria-Austria-Gasleitung
- https://de.wikipedia.org/wiki/MIDAL,https://www.gascade.de/netzinformationen/unser-leitungsnetz/midal/
- https://de.wikipedia.org/wiki/Mittel-Europ%C3%A4ische_Gasleitung,https://en.wikipedia.org/wiki/MEGAL_pipeline
- https://de.wikipedia.org/wiki/NEL_(Pipeline),https://en.wikipedia.org/wiki/NEL_pipeline,https://www.fluxys.com/nel/en/NELSystemInfo/AboutNEL
- https://de.wikipedia.org/wiki/Norddeutsche_Erdgas-Transversale,https://en.wikipedia.org/wiki/Netra
- https://de.wikipedia.org/wiki/OPAL_(Pipeline),https://en.wikipedia.org/wiki/OPAL_pipeline,https://www.opal-gastransport.de/netzinformationen/ostsee-pipeline-anbindungsleitung/
- https://de.wikipedia.org/wiki/Penta-West
- https://de.wikipedia.org/wiki/Rehden-Hamburg-Gasleitung,https://en.wikipedia.org/wiki/Rehden%E2%80%93Hamburg_gas_pipeline
- https://de.wikipedia.org/wiki/Trans_Austria_Gasleitung,https://en.wikipedia.org/wiki/Trans_Austria_Gas_Pipeline
- https://de.wikipedia.org/wiki/Trans-Adria-Pipeline,https://en.wikipedia.org/wiki/Trans_Adriatic_Pipeline
- https://de.wikipedia.org/wiki/Trans-Europa-Naturgas-Pipeline

- https://de.wikipedia.org/wiki/Transgas-Pipeline,Extra
- https://de.wikipedia.org/wiki/WEDAL,https://www.gascade.de/netzinformationen/unser-leitungsnetz/wedal/
- https://de.wikipedia.org/wiki/West-Austria-Gasleitung
- https://en.wikipedia.org/wiki/Arad%E2%80%93Szeged_pipeline
- https://en.wikipedia.org/wiki/BBL_Pipeline
- https://en.wikipedia.org/wiki/BRUA_Pipeline
- https://en.wikipedia.org/wiki/Europipe_II
- https://en.wikipedia.org/wiki/Giurgiu%E2%80%93Ruse_pipeline
- https://en.wikipedia.org/wiki/MEGAL_pipeline,https://de.wikipedia.org/wiki/Mittel-Europ%C3%A4ische_Gasleitung
- https://en.wikipedia.org/wiki/National_Transmission_System
- https://en.wikipedia.org/wiki/Scotland-Northern_Ireland_pipeline,E127
- https://en.wikipedia.org/wiki/South_Wales_Gas_Pipeline
- https://en.wikipedia.org/wiki/Transitgas_Pipeline,https://web.archive.org/web/20070808033932/http://www.swissgas.ch/en/3_2.php
- https://en.wikipedia.org/wiki/Transitgas_Pipeline,transitgassystem_en.gif,https://web.archive.org/web/20070808033932/http://www.swissgas.ch/en/3_2.php
- https://en.wikipedia.org/wiki/V%C3%A1rosf%C3%B6ld%E2%80%93Slobodnica_pipeline
- https://www.fnb-gas.de/media/FNB_GAS_Projekte_2014_02_17_anlage_6_nep-gas-2014_projekt-steckbriefe.pdf
- https://globallnghub.com/wp-content/uploads/2018/09/King.pdf
- https://globallnghub.com/wp-content/uploads/2018/09/King.pdf
- https://news.err.ee/610905/vopak-to-build-initially-4-000-cubic-meter-lng-terminal-at-muuga
- https://globallnghub.com/wp-content/uploads/2018/09/King.pdf
- https://www.lngworldnews.com/tallinna-sadam-alexela-to-work-on-paldiski-lng-terminal/
- https://globallnghub.com/wp-content/uploads/2018/09/King.pdf
- https://www.sourcewatch.org/index.php/Delimara_Malta_LNG_Terminal
- https://theodora.com/pipelines/france_and_belgium_pipelines.html
- https://theodora.com/pipelines/france_and_belgium_pipelines.html, E21
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E10
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E106
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E11
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E112
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E113
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E13
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E14
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E16_1

- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E16_2
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E17_Skikda
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E22_Arzew_1
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E22_Arzew_2
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E48
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E64
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E65
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E78
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E80_1
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E80_2
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E81
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E85
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E86
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E87
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E88
- https://theodora.com/pipelines/france_and_belgium_pipelines.html,E91
- https://transparency.entsog.eu/
- https://www.astora.de/storage-locations/haidach-storage-facility.html?L=1
- https://www.astora.de/storage-locations/jemgum-storage-facility.html?L=1
- https://www.astora.de/storage-locations/jemgum-storage-facility.html?L=2
- https://www.astora.de/storage-locations/rehden-storage-facility.html?L=1
- https://www.baltic-pipe.eu/de/das-projekt/
- https://www.enagas.es/stfls/ENAGAS/Transporte%20de%20Gas/Documentos/CAT_English.pdf
- https://www.enbw.com/unternehmen/konzern/geschaeftsfelder/speicher/gasspeicher/zahlen-daten-fakten.html
- https://www.eneco.nl/over-ons/projecten/gasspeicher/voorraadniveau
- https://www.energiefachmagazin.de/Branchen-News/VNG-Gasspeicher-GmbH-legt-Erdgasspeicher-Buchholz-still
- https://www.enovos.de/industrie/ueber-uns/erdgasspeicher
- https://www.eugal.de/eugal-pipeline/,https://www.mdr.de/nachrichten/politik/regional/baubeginn-erdgastrasse-eugal-umstritten-100.html
- https://www.europipe.com/de/referenzen/referenzprojekte/
- https://www.eustream.sk/en_transmission-system/en_transmission-system
- https://www.fluxys.com/belgium/en/about%20fluxys/infrastructure/network/network
- https://www.gascade.de/en/our-network/compressor-stations/radeland/
- https://www.gascade.de/en/our-network/our-pipelines/jagal/
- https://www.gascade.de/netzinformationen/unser-leitungsnetz/stegal/
- https://www.gascade.de/netzinformationen/unser-leitungsnetz/stegal/,https://en.wikipedia.org/wiki/STEGAL, https://de.wikipedia.org/wiki/STEGAL

- https://www.gascade.de/presse/presseinformationen/pressemitt,http://www.friedrich-vorwerk.de/de/aktuell/projekte/neubau-nowal-dn-1000.htmleilung/news/bau-der-nord-west-anbindungsleitung-nowal-startet-anfang-maerz/
- https://www.gasinfocus.com/en/indicator/existing-and-planned-lng-terminals/
- https://nauticor.de/lng-terminal-nynaeshamn
- https://www.gazprom.com/f/posts/86/569604/portovaya_eng.pdf
- https://www.habau.at/de/projekte/erdgasleitung-wag-expansion-3
- https://www.habau.at/de/projekte/erdgasleitung-wag-plus-600-phase-1
- https://www.hydrocarbons-technology.com/projects/swedegas-lng-facility-port-gothenburg/
- https://www.ign.ren.pt/en/armazenamento-subterraneo3
- https://www.ign.ren.pt/en/armazenamento-subterraneo4
- https://www.innogy-gasstorage.cz/en/index/
- https://www.innogy-gasstorage.cz/en/stramberk/
- https://www.n-ergie.de/geschaeftskunden/produkte/erdgas/erdgasspeicher.html
- https://www.ontras.com/fileadmin/user_upload/Dokumente_Download/Publikationen/ONTRAS_Netzpufferanlage_Burggraf_Bernsdorf.pdf
- https://www.osm.pgnig.pl/en
- https://www.rnf.de/hintergrund-die-erm-gasleitung-von-ludwigshafen-nach-karlsruhe-58182/
- https://www.shz.de/lokales/landeszeitung/wissenschaftler-unter-zeitdruck-id5845016.html
- https://www.sourcewatch.org/index.php/Lithuania-Latvia_Interconnection_Gas_Pipeline
- https://www.storengy.com/countries/france/en/nos-sites/beynes.html
- https://www.storengy.com/countries/france/en/nos-sites/cere-la-ronde.html
- https://www.storengy.com/countries/france/en/nos-sites/manosque.html
- https://www.storengy.com/countries/france/en/our-sites/saint-clair-sur-epte.html
- https://www.swedegas.com/Our_services/services/Storage
- https://www.uniper-energy-storage.com/cps/rde/xchg/ust/hs.xsl/3252.htm?rdeLocaleAttr=en
- https://www.uniper-energy-storage.com/cps/rde/xchg/ust/hs.xsl/3437.htm?rdeLocaleAttr=en
- https://www.vng-gasspeicher.de/storage_locations
- https://www.wesernetz.de/netznutzung/bremen/gasnetz-speicheranlagen.php
- https://www.wingas.com/storage-uk-ltd/home.html
- https://www.PLE
- https://www.www.gascade.de/Verdichterstation_Bunde_2016.pdf
- https://www.www.gascade.de/Compressor_station_Eischleben_2016.pdf
- https://www.www.gascade.de/Verdichterstation_Lippe_72dpi060614.pdf
- https://www.www.gascade.de/Anlandestation_Greifswald_Lubmin_2016.pdf
- https://www.www.gascade.de/Verdichterstation_Mallnow_2016.pdf
- https://www.www.gascade.de/Compressor_station_Olbernhau_2016.pdf
- https://www.www.gascade.de/Verdichterstation_Reckrod_2016.pdf

- https://www.www.gascade.de/Verdichterstation_Rehden_2016.pdf
- https://www.www.gascade.de/Compressor_station_Rueckersdorf_2016.pdf
- https://www.www.gascade.de/Verdichterstation_Weisweiler_201609.pdf
- https://www.opal-gastransport.de/Compressor_Station_Radeland_72dpi.pdf
- https://www.opal-gastransport.de/Verdichterstation_Radeland_2016.pdf
- https://www.pipelinesystems.com/: "NETRA compressor station Wardenburg"
- https://www.grtgaz.com: Plan_decennal_2017-2026.pdf
- https://www.streicher.de
- https://www.open-grid-europe.com
- https://www.open-grid-europe.com
- https://www.open-grid-europe.com, Pressinformation, May 2017, "Herbstein compressor station new build project"
- https://www.fluxys.com/tenp/de
- Presseinfo Bayernets, WinGas, 19-Sep-2008
- NWZonline.de, 22-04-2010: "ExxonMobil gibt kräftig Gas" by Tanja Mikulski
- Porzerleben.de: 6-sep-2011, "Open Grid Europe investiert in europäischen Netzverbund"
- Christoph Edler, Bachelorarbeit PR 370005, Technische Universität Wien, "Das österreichische Gasnetz", Juli 2013

9.5 Location name alterations

Location names should be changed into the 26 letters used in the English language.

For names from the individual countries please follow the suggested approach:

- Germany/Austria: *Umlaute* to be replaced with the letter followed by an 'e', e.g.: ü = ue.
- France/Belgium: Omit accent de gues and accent de graphs, e.g.: ó = o.
- Sweden: Please change the last three letters of the Swedish alphabet and replace e.g.: ä = a.
- Poland: Please change any letter, that cannot be found in the English alphabet, knowing that for some letters, that one can only use a single letter instead of the three different letters used in the Polish alphabet, e.g.: z = z.
- Spain/Portugal: Please change any letter, that cannot be found in the English alphabet, e.g.: ñ = n.
- Greece: Please do not use Greek letters. Please try to write the Greek words with Latin letters.
- Denmark: Please change any letter that contains non-English letters, e.g.: "å" with "aa".
- Slovakia, Czech Republic, Hungary, Rumania, Latvia, Lithuania, Estonia, Bulgaria, Slovenia, Croatia: PLEASE use your common sense, based on the examples from the other countries above.

9.6 Country name abbreviations

For convenience we provide a short list of names and two-digit codes (see Table 9.3) for the probably most important countries associated with the European Transmission Grid.

Table 9.3: Country codes

Country name	Country code	Country name	Country code
Albania	(AL)	Kosovo	(XK)
Armenia	(AM)	Latvia	(LV)
Austria	(AT)	Liechtenstein	(LI)
Azerbaijan	(AZ)	Lithuania	(LT)
Belarus	(BY)	Luxembourg	(LU)
Belgium	(BE)	Malta	(MT)
Bosnia and Herzegovina	(BA)	Moldova	(MD)
Bulgaria	(BG)	Montenegro	(ME)
Croatia	(HR)	Netherlands	(NL)
Cyprus	(CY)	Norway	(NO)
Czech	(CZ)	Poland	(PL)
Denmark	(DK)	Portugal	(PT)
Estonia	(EE)	Romania	(RO)
Finland	(FI)	Serbia	(RS)
France	(FR)	Slovakia	(SK)
Georgia	(GE)	Slovenia	(SI)
Germany	(DE)	Spain	(ES)
Greece	(EL)	Sweden	(SE)
Hungary	(HU)	Switzerland	(CH)
Iceland	(IS)	Turkey	(TR)
Ireland and Northern Ireland	(IE)	Belarus	(UA)
Italy	(IT)	Great Britain	(GB)
Russia Federation	(RU)	Europe	(EU)

9.7 Code for generation of combined non-OSM data set

Below the cod that was shown in the chapter describing the generation of a gas network data set has been collated, and will be listed as an entire unit below.

```
# Importing modules
import Code.M_Internet          as M_Internet
import Code.M_GIE               as M_GIE
import Code.M_CSV               as M_CSV
import Code.M_Visuell           as M_Visuell
import Code.M_LoadAdditional    as M_LoadAdditional
import Code.JoinNetz            as JoinNetz
import Code.M_Shape             as M_Shape
import os


DataLocationType    = 'Data_Raw_CSV'
# options are any component label in list
CompNames2Join      = ['LNGs','Storages']
CompNames2Test      = ['Compressors', 'LNGs','Storages', 'PipeSegments']
```

(continues on next page)

(continued from previous page)

```
# options are any non-OSM data source
sourceList          = ['InterNet',  'GIE']


# String giving the relative path name where to save CSV gas network data set files␣
↪of raw data set
dir_Data_Raw_Merged         = 'Ausgabe/GeneratedNetz/02_NonOSM/Data_Merged/'
dir_Data_Raw_Component      = 'Ausgabe/GeneratedNetz/02_NonOSM/Data_Component/'

# String giving the relative path name where to save CSV gas network data set files␣
↪of final data set
dir_Data_Final              = 'Ausgabe/GeneratedNetz/02_NonOSM/Data_Final/'
StatsInputDirName           = 'Ausgabe/GeneratedNetz/02_NonOSM/SetupFiles'
DataStatsOutput             = 'Ausgabe/GeneratedNetz/02_NonOSM/StatsData'
SettingsInputFileName       = 'Ausgabe/GeneratedNetz/02_NonOSM/StatsData'

# Loading data from CSV files, and storing in data dictionary "DataDict"
sourceList          = ['InterNet',  'GIE']
DataLocationType    = 'Data_Raw_CSV'
DataDict            = {}
    Netz_InterNet       = M_CSV.read('Ausgabe/InternetDaten/Data_Raw/')
    Netz_GIE            = M_CSV.read('Ausgabe/GIE/Data_Raw/')

DataDict['InterNet'] = Netz_InterNet
DataDict['GIE']      = Netz_GIE

# Joining data into a single network
CompNames2Join    = ['LNGs', 'Storages', 'Nodes']
ThresValues       = [20, 30, 30]
Netz_Merged       = JoinNetz.join(DataDict, 'Scen_1', CompNames2Join, ThresValues)

# Cleaning of the network
Netz_Merged.cleanUpNodes(skipNodes = True)

# Writing joined network to CSV file
M_CSV.write(dir_Data_Raw_Merged, Netz_Merged)


# Generation of histogram plots
DataStatsOutput    = 'Ausgabe/GeneratedNetz/02_NonOSM/StatsData'
StatsInputDirName  = 'Ausgabe/GeneratedNetz/02_NonOSM/SetupFiles/Method_Testing'
M_Stats.gen_DataHists(Netz_Merged, StatsInputDirName  = StatsInputDirName,␣
↪DataStatsOutput    = DataStatsOutput)


# Generation of the heuristic parameters
CompNames2Test      = ['LNGs', 'Compressors', 'Storages', 'PipeSegments']
M_Stats.gen_StatsParam(Netz_Merged, CompNames = CompNames2Test, SavePlots = True,␣
↪MaxCombDepth = 1,
                       StatsInputDirName  = 'Ausgabe/GeneratedNetz/02_NonOSM/
↪SetupFiles/Method_Testing',
                       DataStatsOutput    = 'Ausgabe/GeneratedNetz/02_NonOSM/
↪StatsData')

################################################
# Here comes the manual work by the user to select the heuristic methods within the␣
↪component setup files.
```

(continues on next page)

(continued from previous page)

```
#####################################################


for compName in CompNames2Test:
    SettingsFileName    = ('Ausgabe/GeneratedNetz/02_NonOSM/SetupFiles/Value_
↪Generation/RetSummary_' + compName + '.csv')
    SimSettings         = M_CSV.read_CSV_raw(SettingsFileName)

    Netz_Merged         = M_Stats.pop_Attribs(Netz_Merged, CompNames = [compName],␣
↪SimSettings = SimSettings)

Netz_Component = Netz_Merged

# writing constant values into attributes
Netz_Component.make_Attrib([], '',    'end_year',   'const', 2050)
Netz_Component.make_Attrib([], '',    'start_year', 'const', 1983)

    Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_1', 'const',␣
↪1, 'fill')
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_2', 'const', 1,
↪'fill')
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_3', 'const', 1,
↪'fill')
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_4', 'const', 1,
↪'fill')
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_5', 'const', 1,
↪'fill')
Netz_Component.make_Attrib(['Compressors'], '', 'turbine_fuel_isGas_6', 'const', 1,
↪'fill')

# Converting a gas component data set to a gas network data set
# Simplifying network
Netz_Component.cleanUpNodes(compNames = [], skipNodes = True)
Netz_Final  = M_Shape.moveComp2Pipe(Netz_Component.copy2(), 'Compressors',
↪'PipeSegments', maxDistance = 100.0)
Netz_Final  = M_Shape.moveComp2Pipe(Netz_Final.copy2(), 'Storages',
↪'PipeSegments', maxDistance = 100.0)
Netz_Final  = M_Shape.moveComp2Pipe(Netz_Final.copy2(), 'ConnectionPoints',
↪'PipeSegments', maxDistance = 100.0)
Netz_Final  = M_Shape.moveComp2Pipe(Netz_Final.copy2(), 'EntryPoints',
↪'PipeSegments', maxDistance = 100.0)
Netz_Final  = M_Shape.moveComp2Pipe(Netz_Final.copy2(), 'BorderPoints',
↪'PipeSegments', maxDistance = 100.0)
Netz_Final  = M_Shape.moveComp2Pipe(Netz_Final.copy2(), 'LNGs',
↪'PipeSegments', maxDistance = 100.0)

# Cleaning up network
Netz_Final.Nodes  = Netz_Final.add_Nodes([], [])
Netz_Final.Nodes  = M_Shape.reduceElement(Netz_Final.Nodes, reduceType = 'LatLong')

Netz_Final.remove_unConnectedComponents()
Netz_Final.remove_unUsedNodes()

Netz_Final.Nodes  = Netz_Final.add_Nodes([], [])

# Writing final network to CSV
M_CSV.write(dir_Data_Final, Netz_Final)
```

(continues on next page)

(continued from previous page)

```
# Plotting the final gas network data set
M_Visuell.quickplot(Netz_Final, figureNum = figureNum, LegendStr = '', LegendStyle =
↪'Str(Num)', countrycode = 'EU', Cursor = Cursor, Save = False)
```

9.8 Acknowledgement

The SciGRID_gas project is led by Dr. Wided Medjroubi.

We acknowledge the contribution of Dr. Ontje Luensdorf and Jan Dasenbrock from the DLR Institute of Networked Energy System to the SciGRID_gas project.