

## KG4CDO: A Knowledge Based Framework for Objects Models Synthesis

[KG4CDO website](#)

### Contributors

Correspondence to:

- [Igor Kulikov \(i.a.kulikov@gmail.com\)](mailto:i.a.kulikov@gmail.com)
- [Nataly Zhukova \(nazhukova@mail.ru\)](mailto:nazhukova@mail.ru)
- [Man Tianxing \(mantx@jlu.edu.cn\)](mailto:mantx@jlu.edu.cn)

### Paper

#### Synthesis of multilevel knowledge graphs: Methods and technologies for dynamic networks

Tianxing Man, Alexander Vodyaho, Dmitry I. Ignatov, Igor Kulikov, Nataly Zhukova  
Engineering Applications of Artificial Intelligence, 2023. Connected research.

If you find this repository useful, please cite our paper and corresponding framework:

```
@article{Man_Vodyaho_Ignatov_Kulikov_Zhukova_2023,  
  title={Synthesis of multilevel knowledge graphs: Methods and technologies for dynamic networks},  
  volume={123},  
  url={http://dx.doi.org/10.1016/j.engappai.2023.106244},  
  DOI={10.1016/j.engappai.2023.106244},  
  journal={Engineering Applications of Artificial Intelligence},  
  publisher={Elsevier BV},  
  author={Man, Tianxing and Vodyaho, Alexander and Ignatov, Dmitry I. and Kulikov, Igor and Zhukova, Nataly},  
  year={2023},  
  month=aug,  
  language={en}}
```

### Use case for models building

This section considers a use case of the construction of the telecommunication network (TN) model as a structurally complex dynamic object. The diagram of building and keeping up-to-date the telecommunication network model in the form of a knowledge graph is presented in Fig. 1.

The diagram shows the telecommunication network, from which the initial data are received. The data also comes from the existing information systems. The requirements to the resulting models are defined by end users. TN models are built based on network data using inductive and deductive synthesis.

The main steps in building a TN model as a KG:

1. Obtaining and preparing a TN dataset that includes existing private graph models, statistical and operational data received from the telecommunication network. Two scenarios of working with the source data are supported:

- a. Real private graph models are provided by the existing information systems, operational and statistical data come from the TNs.
- b. Synthetic private graph models with a given number of levels and elements, operational and statistical data for building the TN model are generated.

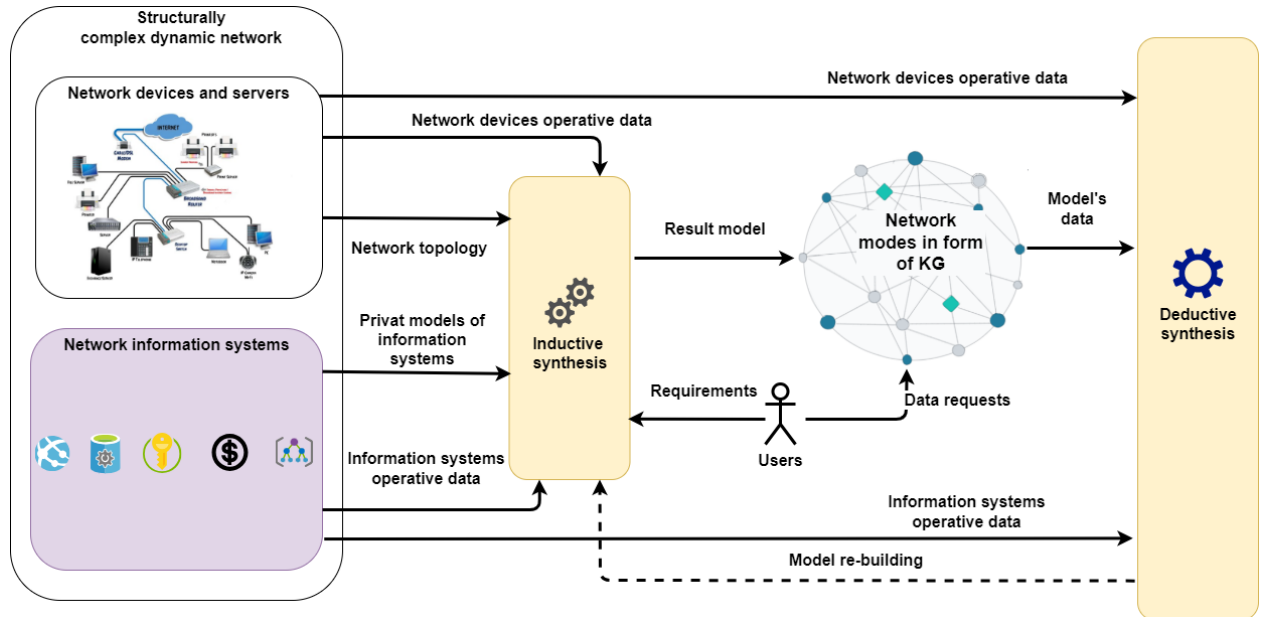


Fig. 1 Diagram of building and keeping up-to-date of the telecommunication network model in the form of knowledge graph

2. Inductive synthesis of a TN model in the format of a knowledge graph with evaluation of synthesis execution time. During synthesis, it is possible to set the following parameters: synthesis of a single-level model/synthesis of a multilevel model and number of elements in the synthesized model. For multilevel models, the number of levels in the synthesized model and distribution of model elements by levels are also defined.
3. Deductive synthesis of a TN model in the format of a knowledge graph with evaluation of synthesis execution time. Supported synthesis parameters: synthesis of a single-level model / synthesis of a multilevel model; number of the level at which the proof of model existence should be performed; deductive synthesis algorithm. The framework supports two algorithms of deductive synthesis: the basic algorithm of deductive synthesis in which the facts received from the TN are processed in one data processing cycle and the results of their processing are used to draw a conclusion about the state of the target model, and a modified algorithm which works in the mode of a sequential processing of each incoming fact (the algorithm is intended for use in systems operating in near real-time mode).
4. Measuring the time of loading the generated inductive model in RDF/XML and Turtle formats into an RDF data repository and determining the number of triplets in the knowledge graph.

Multiple models with different structures can be built and then compared in terms of performance of synthesis methods and time of SPARQL queries execution to an RDF repository which hosts the TN model in the form of KG.

## Framework structure

The component diagram of the developed framework is shown in Fig. 2.

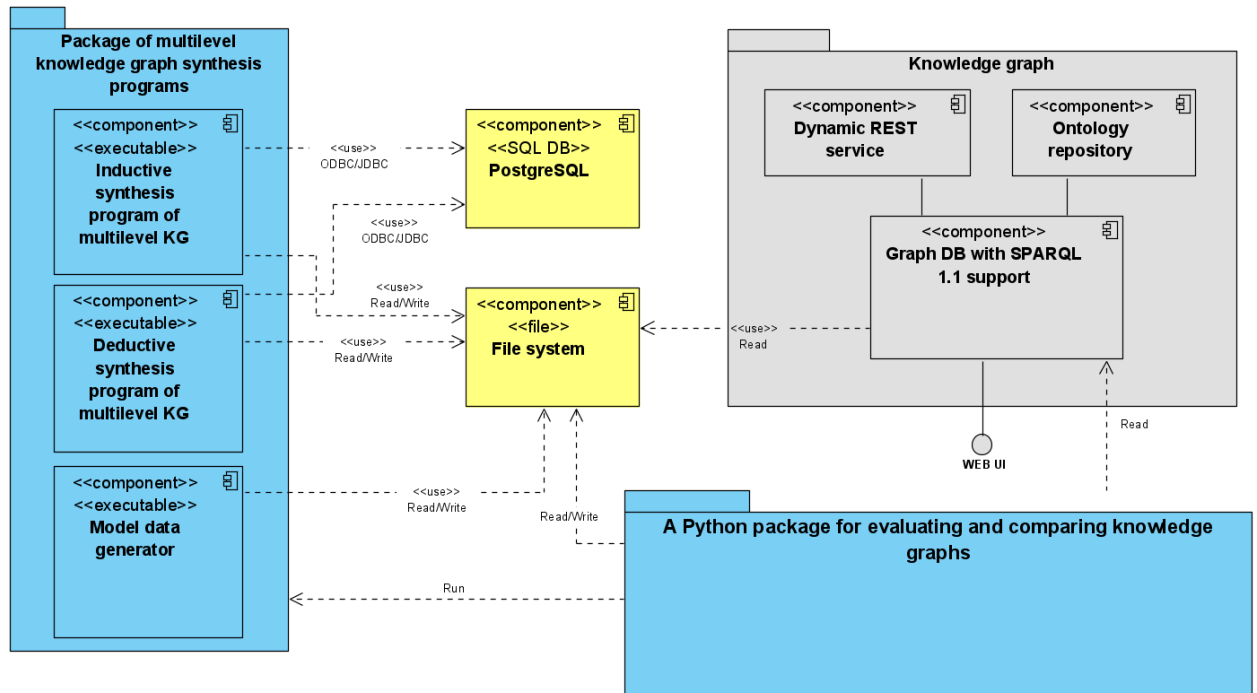


Fig. 2 Component diagram of the developed framework

The framework consists of the following components:

1. A package of multilevel knowledge graph synthesis programs, which includes:  
**Inductive synthesis program of multilevel KG** – Ind\\_synthesis\_4\_(SQL).py.  
**Input data:** Private graph models of SCDO provided by information systems in CSV format (Test\_model.csv, columns: MODEL\_TYPE (type of private graph model, e.g. access rights system model, network topology model, billing model, etc.), NODE\_TYPE (element type of the private graph model, e.g. network devices and links for the network topology model or user accounts and tariffs for the billing model, etc.), ID (identifier of the element within the private graph model), NAME (name of the element of the private graph model), PARENT\_ID (identifier of the parent element of the private graph model), LEVEL\_NUM (level number within the private graph model)); rules for linking private graph models in CSV format (Links\_rules.csv, columns: SRC\_MODEL (name of the type of the first linked model), SRC\_ID (element identifier of the first linked model), SRC\_NAME (element name of the first linked model), DIST\_MODEL (name of the type of the second linked model), DIST\_ID (element identifier of the second linked model), DIST\_NAME (element name of the second linked model), RULE (name of the linking rule)).

**Output:** The model of the CSDO in the form of a knowledge graph in RDF/XML format and files Model\_base.npz, Model\_base\_links.npz containing the inductive model and Model\_base\_req.npz, Model\_base\_links\_req.npz containing the inductive model, that takes into account the requirements of users in the format of Numpy data arrays, the runtime of inductive synthesis.

**Deductive synthesis program of multilevel KG for the base algorithm** – Deductive\_synthesis\_(SQL).py.

**Input data:** Telecommunication network model in the form of knowledge graph in RDF/XML format and Numpy array; set of facts to be processed in CSV format (Facts.csv, columns: MODEL\_TYPE (private graph model type), NODE\_TYPE (private graph model element type), FACT\_ID (identifier of the fact within the private graph model), NAME (fact name), PARENT\_ID (identifier of the parent element of the private graph model), LEVEL\_NUM (level number within the private graph model)). **Output:** Level number at which the model is proved, -1 if the model cannot be proved, deductive synthesis runtime.

**Deductive synthesis program of multilevel KG for the modified algorithm** – Deductive\_synthesis\_modified(SQL).py.

**Input data:** Telecommunication network model in the form of knowledge graph in RDF/XML format and Numpy array; single fact to be processed in CSV format (Facts.csv, columns: MODEL\_TYPE (private graph model type), NODE\_TYPE (private graph model element type), FACT\_ID (identifier of the fact within the private graph model), NAME (fact name), PARENT\_ID (identifier of the parent element of the private graph model), LEVEL\_NUM (level number within the private graph model)); number of the level at which the synthesis time needs to be evaluated.

**Output:** Time to process a single fact, time to perform deductive synthesis at a given level.

**A model data generator that contains the following procedures:**

**Private model generator of SCDS component models for building a multilevel model** – Model\_creation\_hierarchy.py. The generator has the following restrictions: the number of linked models is 2, the number of levels of linked models is 3, 4, 5. The structure of generated models is presented in the table below.

Model #1		Model #2	
Level 1	1	Level 1	1
Level 2	L2	Level 2	L2
Level 3	L3	Level 3	L3
Level 4	L4	Level 4	L4
Level 5	L5	Level 5	L5
Level 6: objects	M1	Level 6: objects	M2

**Input data:** Number of lower level elements of linked model #1 (M1), number of lower level elements of linked model #2 (M2), number of elements at level 2 of linked models (L2), number of elements at level 3 of linked models (L3), number of elements at level 4 of linked models (L4), number of elements at level 5 of linked models (L5).

**Output:** Private graph models in CSV format (Test\_model.csv, columns: MODEL\_TYPE (private graph model type), NODE\_TYPE (private graph model element type), ID (element identifier within the private graph model), NAME (private graph model element name), PARENT\_ID (parent element identifier of the private graph model), LEVEL\_NUM (level number within the private graph model)); set of facts for processing in CSV format (Facts.csv, columns: MODEL\_TYPE (private graph model type), NODE\_TYPE (private graph model element type), FACT\_ID (identifier of the fact within the private graph model), NAME (fact name), PARENT\_ID (identifier of the parent element of the private graph model), LEVEL\_NUM (level number within the private graph model)).

***Private model generator of SSDS component for building a one-level model – Model\_creation\_one-level.py.***

**Input data:** Number of elements of model \#1 to be linked, number of elements of model \#2 to be linked.

**Output:** Private graph models in CSV format (Test\_model.csv, columns: MODEL\_TYPE (private graph model type), NODE\_TYPE (private graph model element type), ID (element identifier within the private graph model), NAME (private graph model element name), PARENT\_ID = None and LEVEL\_NUM = 0 for all the elements of one-level model); set of facts for processing in CSV format (Facts.csv, columns: MODEL\_TYPE (private graph model type), NODE\_TYPE (private graph model element type), FACT\_ID (identifier of the fact within the private graph model), NAME (fact name), PARENT\_ID = None and LEVEL\_NUM = 0 for all the elements of one-level model).

***A set of CSDO test model generators in the format of multilevel knowledge graphs for analyzing SPARQL query execution time:***

- Load\_comp\_KG(hierarhy\_3\_2-2)\_generate\_script.py is a script that generates a three-level knowledge graph consisting of two private graph models linked at the second level.
- Load\_comp\_KG(hierarhy\_3\_3-3)\_generate\_script.py is a script that generates a three-level knowledge graph consisting of two private graph models linked at the third level.
- Load\_comp\_KG(hierarhy\_4\_2-2)\_generate\_script.py is a script that generates a four-level knowledge graph consisting of two private graph models linked at the second level.
- Load\_comp\_KG(hierarhy\_5\_2-2)\_generate\_script.py is a script that generates a five-level knowledge graph consisting of two private graph models linked at the second level.

**Input data:** The number of elements of the linked private models at each level.

**Output:** The CSDO model in the form of a knowledge graph in RDF/XML format.

**Generators of the test CSDO model in the format of one-level knowledge graph for analyzing the speed of SPARQL queries execution** – Load\_comp\_KG(linear)\_generate\_script.py.

**Input data:** The number of elements of the private models to be linked.

**Output:** The CSDO model in the form of a knowledge graph in RDF/XML format.

2. Knowledge graph component including:

- A graph DBMS with SPARQL 1.1 support;
- Ontology repository;
- Dynamic REST service for organizing interaction with external systems.

3. Relational DBMS PostgreSQL.

4. File system for storing input and output data in file formats.

The file system is used in the following cases:

- Inductive and Deductive Synthesis programs perform reading of private graph models and operational data about TNs from files placed in the File System;
  - Inductive synthesis program generates TN models in RDF/XML format and places them as files in the File System;
  - Inductive and Deductive Synthesis programs save their logs in the File System;
  - Inductive and Deductive Synthesis programs use PostgreSQL relational DBMS to implement operations on data tables and store intermediate results;
  - Loading models and operational data into the graph data store is performed from RDF/XML files hosted in the File System.
5. A Python package for evaluating and comparing knowledge graphs which also contains a set of SPARQL request is built based on model structure and is aimed to test different request types. A typical set of requests is provided:
- Select an element of the CSDO model by its identifier.
  - Select a list of logically related elements of the CSDO model (taking into account hierarchical and single-level connections).
  - Select a list of logically related elements of the CSDO model using additional filtering.
  - Select a list of logically related elements of the CSDO model using additional filtering and grouping.
  - Search for a list of elements by a substring.

### Example of use

The framework was used to evaluate the performance of inductive and deductive synthesis algorithms and to evaluate the query execution time to the generated knowledge graphs when building models of complex objects with different structure and size. A detailed description of the conducted study can be found in the [article](#). Below the main steps that should be executed to evaluate the performance of inductive and deductive synthesis algorithms on the example of building five-levels knowledge graph are enumerated:

1. Generate private graph models.

To obtain a multilevel knowledge graph, the script **Model\_creation\_hierarchy.py** should be used. Input parameters are set as script variables: num\_items\_1, num\_items\_2 – numbers of elements on low levels for private models. The higher levels of provided private models are described in the configuration file: Test\_template\_1.csv:

```
Billing,Billing_Node,1,Billing_core_node,,0
Billing,Billing_Node,2,Tariff_1,1,1
Billing,Billing_Node,3,Tariff_2,1,1
Billing,Billing_Node,4,Tariff_3,1,1
Billing,Billing_Node,5,Tariff_4,1,1
Billing,Billing_Node,6,Tariff_5,1,1
Billing,Tier,7,Tier_1,1,1
Billing,Tier,8,Tier_2,1,1
Billing,Tier,9,Tier_3,1,1
Billing,Tier,10,Tier_4,1,1
Billing,Tier,11,Tier_5,1,1
PPV,PPV_Channel,1,PPV_calaloge_Core,,0
PPV,PPV_Channel,5,Sport,1,1
PPV,PPV_Channel,6,Children,1,1
PPV,PPV_Channel,7,Famaly,1,1
PPV,PPV_Channel,8,Documentary,1,1
PPV,PPV_Channel,9,Sport_events,5,2
PPV,PPV_Channel,10,Sport_interview,5,2
PPV,PPV_Channel,11,Sport_news,5,2
PPV,PPV_Channel,12,Famaly_shows,7,2
PPV,PPV_Channel,13,Famaly_films,7,2
PPV,PPV_Channel,14,Famaly_sport,7,2
PPV,PPV_Channel,15,Children_cartoons,6,2
PPV,PPV_Channel,16,Children_sport,6,2
PPV,PPV_Channel,17,Children_films,6,2
PPV,PPV_Channel,18,Documentary_films,8,2
Entitlements,Entitlements_Node,1,Entitlements_core_node,,0
Entitlements,Billing_Node,2,Tariff_1,1,1
Entitlements,Billing_Node,3,Tariff_2,1,1
Entitlements,Billing_Node,4,Tariff_3,1,1
Entitlements,Billing_Node,5,Tariff_4,1,1
Entitlements,Billing_Node,6,Tariff_5,1,1
Entitlements,Tier,13,Tier_1,1,1
Entitlements,Tier,14,Tier_2,1,1
Entitlements,Tier,15,Tier_3,1,1
Entitlements,Tier,16,Tier_4,1,1
Entitlements,Tier,17,Tier_5,1,1
Entitlements,PPV_Channel,7,Sport,2:13,2
Entitlements,PPV_Channel,8,Documentary,2:13,2
Entitlements,PPV_Channel,9,Children,3:14,2
Entitlements,PPV_Channel,10,Famaly,3:14,2
Entitlements,PPV_Channel,11,Sport,4:15,2
Entitlements,PPV_Channel,12,Children,4:15,2
Entitlements,PPV_Channel,18,Famaly,4:15,2
Entitlements,PPV_Channel,19,Documentary,4:15,2
Entitlements,PPV_Channel,20,Sport_interview,5:16,2
Entitlements,PPV_Channel,21,Sport_news,5:16,2
Entitlements,PPV_Channel,22,Sport_events,6:17,2
Entitlements,PPV_Channel,23,Famaly_shows,6:17,2
Services,Service,1,Services_core_node,,0
Services,Service,2,PPV,1,1
Services,Service,3,Watch_TV,1,1
Services,Service,4,VOD,1,1
Services,Service,5,Time_shift,1,1
Services,Service,6,DVR,1,1
User,User_Group,1,User_core_node,,0
User,User_Group,2,Group_1,1,1
User,User_Group,3,Group_2,1,1
User,User_Group,4,Group_3,1,1
```

User,User\_Group,5,Group\_4,1,1  
User,User\_Group,6,Grope\_5,1,1

The file includes the following private models templates: Billing model, PPV events schedule model, Entitlements model, Services model end User groups model. Each of the private model is hierarchical an their structure on upper levels are static and defined in the Test\_template\_1.csv. The lower level items are added dynamically using the rule: Billing model, Entitlements model, Services model end User groups model are enriched by random items which are limited by max\_items\_1 variable and PPV events schedule model is limited by max\_items\_2. Each of the private models is hierarchical.

It should be noted that for the purposes of forming a symmetric structure of the KG at the upper levels, the data generator provides an equal number of elements at levels 2–5 for linked private graph models.

Example of Input parameters set:

```
def create_model(): 1 usage  kulikovia *  
    num_items_1 = 100000  
    num_items_2 = 100000
```

Example of output:

```
≡ Test_facts.csv  
≡ Test_model.csv
```

To obtain a one-level knowledge graph it is necessary to use the script: **Model\_creation\_one-level.py**. Input parameters are set as script variables: num\_items\_1, num\_items\_2 – numbers of elements on low levels for privet models. The higher levels of provided private models are empty due to one-level model has no hierarchy. Pay attention that **Model\_creation\_one-level.py** also generate rules for linking models for inductive synthesis algorithm (Links\_rules-one-level-TEST.csv).

Example of Input parameters set:

```
def create_model(): 1 usage  kulikovia *  
    num_items_1 = 100000  
    num_items_2 = 100000
```

Example of output:

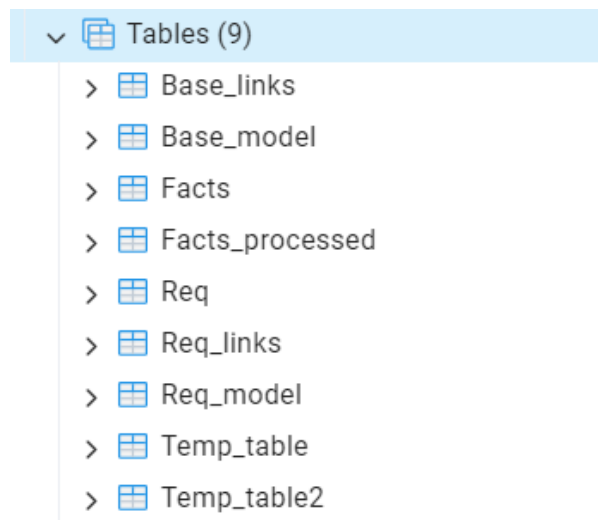
```
≡ Test_facts.csv  
≡ Test_model.csv
```

```
≡ Links_rules-one-level-TEST.csv
```

2. Perform inductive synthesis and estimate its execution time.



To perform inductive synthesis it is required to use the script **Ind\_synthesis\_4\_(SQL).py**, Python interpreter version 3 or higher, PostgreSQL version 13 or higher. It is necessary to create a database and to run the SQL script **DB\_create\_script.sql** in the PostgreSQL console. The following DB scheme must be created:



Please ensure that you DB connection parameters are valid. The our example is below:

```
#Open DB connection
con = psycopg2.connect(
    database="Synthesis_mod_AI",
    user="postgres",
    password="admin",
    host="127.0.0.1",
    port="5433"
)
```

Input data in the format of CSV files should be located in the same folder with the Python script. Output data is placed to the folder defined in the work\_dir variable:

```
#Set up working directory
work_dir = "C:\\1\\"
```

The execution time of synthesis steps is displayed in the console of the Python interpreter.

Example of input data:

```
#Set up model type
model_type = 1 #1 - hierarchical model; 0 - one-level model
```

≡ Test\_model.csv

≡ Links\_rules.csv

or

≡ Links\_rules-one-level-TEST.csv

depending on the model type (hierarchical or one-level).

Example of output:

## Console data:

Connected to DB  
Table Base\_model truncated  
Table Req\_model truncated  
Table Base\_links truncated  
Req truncated  
Table Req\_links truncated  
Temp\_table truncated  
Temp\_table2 truncated  
Started  
SRC model imported. Time: 923 ms.  
Base model created. Time: 0 ms.  
SRC model added. Time: 3097 ms.  
Indexes normalized. Time: 145 ms.  
Base links created. Time: 447 ms.  
Req dic entered  
Req object created. Time: 2927 ms.  
Req object: The JOIN #1 executed  
Part#1 finished  
ID normalisation started  
Reqs applied. Time: 2104 ms.  
Req model links created. Time: 507 ms.  
Models saved. Time: 9282 ms.  
Total time spent: 19432 ms.

## Output files:



Base.csv

- inductive model in CSV format;



Base\_rdf\_1.xml

- inductive model in RDF/XML format;



Req.csv

- inductive model with users requirements in CSV format;



Req\_rdf\_1.xml

- inductive model with users requirements and operative data in RDF/XML format;



Model\_base.npz



Model\_base\_links.npz



Model\_base\_links\_req.npz



Model\_base\_req.npz


- All the models in format of Numpy arrays for using directly by Deductive synthesis algorithms.


3. Perform deductive synthesis and estimate its execution time (for basic and modified algorithms).


To perform deductive synthesis, it is necessary to use the scripts **Deductive\_synthesis\_(SQL).py** and **Deductive\_synthesis\_modified(SQL).py**, which require Python interpreter version 3 or higher to run. Input data in CSV file format and arrays in the format provided by the Python module NP, used for working with arrays, must be located in the same folder with the Python script; the number of the level at which the proof occurs is specified via the Python variable of the script `max_level`. The synthesis time is displayed in the console of the Python interpreter.


Deductive synthesis analyses a set of facts about the model. Initially, we generate the full set of positive facts about all the model elements together with private model and place the data to `Test_facts.csv` file. This file should be placed to the working directory defined in the `work_dir` variable. Please pay attention that provided set of facts provides deductive synthesis in the 0 level. To change the level of improvement it is necessary to delete manually facts on chosen levels in the `Test_facts.csv` file.


Example of input data for **Deductive\_synthesis\_(SQL).py**:


 `Model_base.npz`

 `Model_base_links.npz`

 `Model_base_links_req.npz`

 `Model_base_req.npz`

 `Test_facts.csv`


 `Req_model_1.csv`


Example of output for **Deductive\_synthesis\_(SQL).py**:


Console data:


```
2024-10-27 21:05:43.853914 - Deductive synthesis is Started
2024-10-27 21:05:44.628946 - Reference model has been imported
2024-10-27 21:05:47.814100 - Set of facts has been imported
2024-10-27 21:05:50.102797 - The rules readed
2024-10-27 21:05:50.102797 - The rules applied
2024-10-27 21:05:50.102797 - The rules red
2024-10-27 21:05:50.102797 - The deductive model proving started
2024-10-27 21:05:50.159390 - The model Max_level= 0
2024-10-27 21:05:52.200438 - The model is proved. Complexity: 400000 Level number: 0 Len model:400000
2024-10-27 21:05:52.200438 - Exec. time: 4386ms.
```


Example of input data for **Deductive\_synthesis\_modified(SQL).py**:

 Model\_base.npz


 Model\_base\_links.npz

 Model\_base\_links\_req.npz

 Model\_base\_req.npz

 Test\_one\_fact.csv

- this file should be made manually and consists only of header and one chosen fact.

 Req\_model\_1.csv

Example of output for **Deductive\_synthesis\_modified(SQL).py**:

Console data:

```
2024-10-27 21:27:17.623687 - Deductive synthesis is Started
2024-10-27 21:27:18.444923 - Reference model has been imported
2024-10-27 21:27:20.011175 - Set of facts has been imported
2024-10-27 21:27:20.011175 - Facts import time: 1566.251953125ms.
2024-10-27 21:27:21.436311 - The rules readed
2024-10-27 21:27:21.436311 - Rules reading time: 1425.1357421875ms.
2024-10-27 21:27:24.561038 - The current factes have been processed
2024-10-27 21:27:24.561038 - Facts processing time: 3124.727783203125ms.
2024-10-27 21:27:24.627123 - The model Max_level= 0
2024-10-27 21:27:24.849862 - The model is proved. Complexity: 0 Level number: 0 Len model:400000
2024-10-27 21:27:24.849862 - Exec. time for Level: 0 - 288.823486328125ms.
```

#### 4. Analyze the execution time of SPARQL queries to CSDO models.

In order to analyze the execution time of SPARQL queries to the CSDO models in the form of a knowledge graph with different parameters, it is necessary to generate such models using a set of generators:

- Load\_comp\_KG(hierarhy\_3\_2-2)\_generate\_script.py,
- Load\_comp\_KG(hierarhy\_3\_3-3)\_generate\_script.py,
- Load\_comp\_KG(hierarhy\_4\_2-2)\_generate\_script.py,
- Load\_comp\_KG(hierarhy\_5\_2-2)\_generate\_script.py,
- Load\_comp\_KG(linear)\_generate\_script.py.

The Python interpreter version 3 or higher is required to run the scripts. Model parameters are set via the following script variables: max\_level\_2, max\_level\_3, max\_level\_4, max\_level\_5, max\_objects\_1 and max\_objects\_2. The resulting models in RDF/XML format are loaded into the RDF data repository using standard data storage methods.

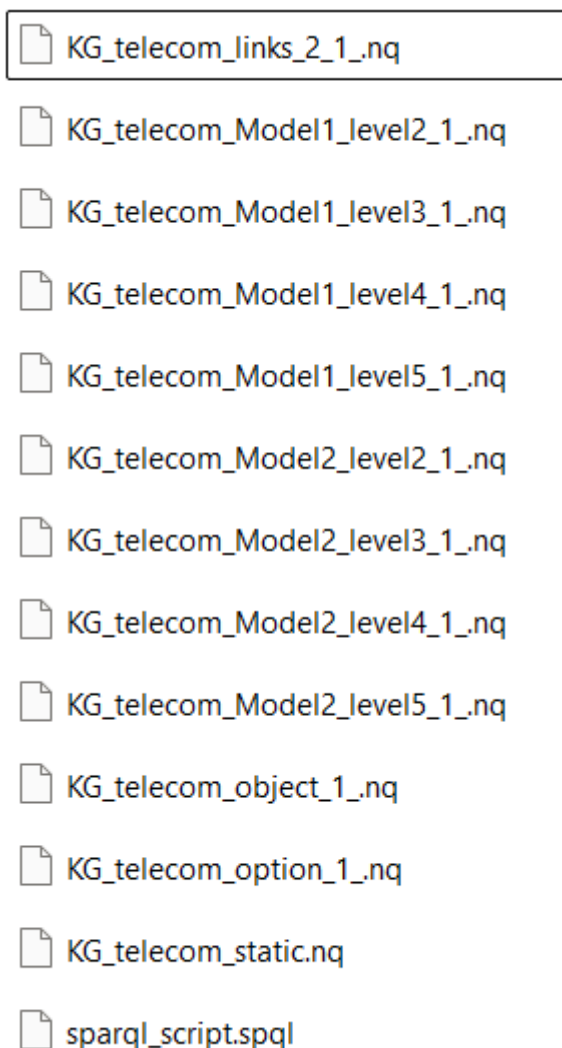
All the script have similar sets of parameters and we provide an example for Load\_comp\_KG(hierarhy\_5\_2-2)\_generate\_script.py.

Example of input data:

```
Max_Level_2 = 1000
Max_Level_3 = 1000
Max_Level_4 = 1000
Max_Level_5 = 1000
Max_Objects_1 = 1000 #Model 1 lowest level
Max_Objects_2 = 1000 #Model 1 lowest level
Max_Step_1 = 100000 #Maximum number of model elements in each rdf file
Max_Step_2 = 50000 #Maximum number of model elements in each rdf file

#Constants
SPARQL_path = "C:/Blazegraph/1" #The path is used for UPDATE RDF DB creation script
Model_path = "Hierarchy_model/" #The path for output files
```

Example of output:



To load the RDF/XML data to RDF datastore it is necessary to run sparql\_script.spql in Update console of the RDF datastore. The authors have used [Blazegraph](#). Queries were executed

from the standard Web application Blazegraph which provides a screen display of the execution time of each SPARQL query.