This is a hands-on lab on knowledge graphs[1]*. We will use the GraphDB database for working with knowledge graphs. To prepare for the session, you are recommended to get familiar with the RDF Schema vocabulary specification, SPARQL specification and GraphDB manual.*
In the sequel, we provide the environment setup and then exercises to be solved. Each group must upload the solutions to Learn-SQL (look for the corresponding assignment event). One group member must submit the solutions (check below for a precise enumeration of what you need to submit) and list all group members in such document. Check the assignment deadline and be sure to meet it. It is a strict deadline!

What to deliver?

Upload ONE compressed file with your solutions. The main file of your solution must be a pdf file named as '`[Group]-[MemberSurname]+.pdf`'. The file must be structured according to the sections you will find below and, within each section, specify your solution following the instructions of each section. If you make any assumption not explicit in the statement, add a note in the corresponding section. Some sections ask you to deliver code. In those cases, you must attach a file per section named as follows: '`[Group]-(SUB)SECTION-[MemberSurname]+`'.

Setup Instructions

For section A

For section A, you need to load an excerpt of the DBpedia TBOX and ABOX to GraphDB. We will use DBpedia to get used to GraphDB. Follow these steps:

- Download the Graph DB installer: `https://www.ontotext.com/products/graphdb/graphdb-free/`
  Fill in the form that appears on the right. After submitting the form, you will receive an email at the address you provided with various download links depending on your OS (Windows, MacOS or Linux). Please also check your junk / spam folder in case the email went there.

- Install GraphDB: After downloading the installer, you should run normally and you do not require an administrator account to run it. Follow the steps that appear in the executor without changing any of the configurations and wait for the installation process to finish.

[1] This statement distinguishes the concepts of knowledge graph and ontologies. Refer to the main lectures to know the difference, but, in short, an ontology is a knowledge graph distinguishing the TBOX and ABOX and using inference. Be thoughtful about the wording of the exercise statements and their usage.

- Launch GraphDB: Once the installation is complete, you should start the GraphDB server by launching the `'GraphDB Free'` application from your home / start menu. Once the server is up, a new window in your browser should open automatically with the URL `http://localhost:7200/`.

- Configure settings: Go to the GraphDB application (i.e., the background stand-alone window) and click on the button `'Settings...'` at the top. A new pop-up window should open, and you should introduce the following custom java properties (the first one sets the maximum number of triples that can be loaded via GraphDB Workbench and the heap space used by the JVM):

  `graphdb.workbench.maxUploadSize=40000000000`

  `Xmx=2000m`

  Afterwards, click on the `'Save and restart'` button at the bottom right. The GraphDB server will restart and will open a new browser window again with the same URL as above.

- Load a dump from DBpedia: It includes both the DBpedia TBOX and **some** ABOX instances. Download the file from the following URL `https://www.dropbox.com/scl/fi/vjc0z9bjz8p3b9g3z730z/DBPEDIA2014.zip?rlkey=31ojtegi6sucxu7njbzfk1rfl&st=kkc4h7h6&dl=0` and unzip it. This should generate two files, one with the extension `.owl` and another with the extension `.nt`. In the knowledge graph world, the extension of the file refers to the format used to express the file triples.

- Create a repository: Before importing any data, we need to first create a repository. Go to the GraphDB interface (i.e., to the browser tab) and select `'Setup'` and then `'Repositories'` from the left-hand menu. Afterwards, click on the `'Create new repository'` and apply the following steps:

  Insert a `'Repository ID'` and `'Repository title'` of your own.

  Turn inference on: Under `'Ruleset'` select `'RDFS (Optimized)'`. This tells GraphDB to activate reasoning. The option chosen tells GraphDB to perform reasoning based on an optimized version of the RDFS regime entailment inference rules.

  Note: GraphDB supports inference out of the box (check the theoretical slides for more information). Inference is the derivation of new knowledge from existing knowledge and axioms. It is used for deducing further knowledge based on existing RDF data and a formal set of inference rules. To understand how GraphDB deals with reasoning please check:

  `http://graphdb.ontotext.com/documentation/free/inference.html#inference-in-graphdb`,

`http://graphdb.ontotext.com/documentation/standard/reasoning.html` and `https://graphdb.ontotext.com/documentation/free/rdfs-and-owl-support-optimisations.html`

The first link explains how inference can be customized in GraphDB. The second one explains the reasoning process and how GraphDB materializes it. The third explains the difference between the RDFS inference rules seen in the lectures and what GraphDB calls the optimized version of them.

- Load data into the repository: Go the GraphDB interface and select `'Import'` and then `'RDF'` from the left-hand menu. Afterwards, click on the `'Upload RDF files'` button and choose the two files that you just extracted (uncompress them first!). Wait for uploading the files to be finished, it should take around 3 - 4 minutes to upload the larger file. Once both files are uploaded, select them both and click on the `'Import'` button at the top. A new pop-up will appear and you need to fill in the following configuration parameters:

  Base IRI: enter `http://dbpedia.org/resource/`

  Target graphs: choose `'Named graph'` and enter the following in the text box: `http://localhost:7200/dbpedia/`

  After that, click on the `'Import'` button at the bottom right corner of the pop-up and wait for the import operation to finish. Since the files are large, the import will take around 10 minutes to be done.

- Test: Go to `'SPARQL'` on the left-hand menu and copy the following query into the text box that appears:

  ```
  SELECT DISTINCT ?s WHERE { ?s ?p ?o .  } LIMIT 50
  ```

  Click on the `'Run'` button at the bottom right corner of the text box to execute the query. If the query does not return any results, this means that no data was found in the repository and that the import operation failed. If it does return results, the import was successful and you can proceed with the exercises of the session.

- Turn on autocomplete: Click on `'Setup'` and then `'Autocomplete'` from the left-hand menu. In the page that appears, toggle the button at the top under `'Autocomplete index'` to change it from off to on. A new status that says `'building'` should appear to the right of the toggle. This will activate autocomplete when we are executing queries and will allow us to search for properties or classes that are defined in our knowledge graph. This feature, similar to that in most programming IDEs, will help you massively during the lab. You do not need to wait for the building to finish, you can continue with the exercises of this session and the autocomplete index building will continue in the background.

Important note: all the above steps are meant to facilitate your first steps as an initial walkthrough to GraphDB. Remember though you are responsible to learn the specificities of GraphDB on your own if the above explanations are not enough, so be sure you understand what each step does. Also, you may want to activate other options that you may find interesting.

A Exploring DBpedia

DBpedia

DBpedia[2] *is a crowd-sourced community effort to extract structured content from the information created in various Wikimedia projects. This structured information resembles an open knowledge graph which is available for everyone on the Web. A knowledge graph is a special kind of database which stores knowledge in a machine-readable form and provides means for information to be collected, organised, shared, searched and utilised. Google uses a similar approach to create those knowledge cards during search.*

Tasks

One of the main challenges of a knowledge graph with regard to a traditional (e.g., relational) database is that we may not know, at all, the terminological axioms (i.e., the TBOX) followed by data instances (i.e., the ABOX). Therefore, the first usual step to start working with a knowledge graph is to **explore** its TBOX. During the lectures, you will have to solve a SPARQL exercise that will leave you alone with the burden of exploring DBPedia. There, you will have to write basic SPARQL queries to find classes, properties and, once you identified an interesting one, how to find triples where it participates. All in all, that exercise guides you, for first time, through an exploratory effort over a knowledge graph.

The advantage of using GraphDB, however, is that it offers an advanced visual interface to explore the classes defined in a knowledge graph that saves us from executing a manual exploratory exercise. Indeed, GraphDB provides nice add-ons to explore the knowledge graph in a more efficient way. To access these features, click on `'Explore'` from the left-hand menu and browse the different options. The most relevant one, probably, is `'Class hierarchy'`. Click on it and wait a few seconds for the graph to load. After that, you should get a visual graph that depicts the different classes defined in the DBpedia knowledge graph and the hierarchies between them. For instance, you can see that the biggest class is called `'Agent'`, which has many sub-classes. When clicking on the bubble representing each class, a pop-up panel with metadata about that class appears on the right. This panel contains its semantic bag (i.e., related properties and adjacent classes), the name of the

[2] https://wiki.dbpedia.org

class, its description (obtained through the RDFS property rdfs:label used to provide a free-text description for human consumption) and some of its instances. Note the semantic bag concept (retrieved through the domain-range graph) is similar to the DESCRIBE clause in SPARQL and therefore meets the requiements to *derefer* a URI). Also, realize that the results retrieved by GraphDB through these features **are affected by the inference capabilities selected when creating the repository**.

Spend some minutes exploring DBPedia, both its classes and properties, and browse around and understand all explorative options provided by GraphDB. Once your curiosity is satisfied, go back to the `'SPARQL'` page and execute queries that are related to the classes and properties defined by DBpedia. This exercise is meant to be exploratory and to let you get familiar with GraphDB. There is no delivery associated to this section.

Note: When querying, it is important to note that the semantics of the knowledge graph are determined by different namespaces, which are defined by the "prefix" keyword at the beginning of the query. Thus, it is important to properly understand and use them. Also, be aware of the inference capabilities activated when answering the questions above. Finally, in the SPARQL page, be sure the option "*include inferred data in results*" is ON when using inference.

B Ontology creation

Let's build an ontology. Creating an ontology is not simple and requires training. Thus, in this section we will train your modeling skills as well as introduce you to the required tools to manipulate and create triples. In this assignment we will practice how to create and query your own **ontology**.

B.1 TBOX definition

Define a TBOX for the research publication domain following the idea of Lab 1 (this way, this will also help you to implicitly get a grasp of pros and cons with regard to property graphs). Thus, we want to model the concepts of paper, authorship, publication and review. Specifically, authors write research papers that can be published in the proceedings of a conference or workshop (a conference is a well-established forum while a workshop is typically associated to new trends still being explored), or in a journal. A conference/workshop is organized in terms of editions. Each edition of a conference/workshop is held in a given city (venue) at a specific period of time of a given year. Proceedings are published records which include all the papers presented in an edition of a conference/workshop. Oppositely, journals do not hold joint meeting events and, like a magazine, a journal publishes accepted papers in terms of volumes. There can be various volumes of a journal per year. A paper can be written by many authors, however only one of them acts as corresponding author. A paper can be cited by another paper, meaning their content is related. A paper can be about one

or more topics, specified by means of keywords (e.g., property graph, graph processing, data quality, etc.). A paper must also contain an abstract (i.e., a summary of its content). Finally, we also want to include in the graph the concept of review. When a paper is submitted to a conference/workshop or a journal, the conference chair or the journal editor assigns a set of reviewers (typically three) to each paper. Reviewers are scientists and therefore they are relevant authors (i.e., they have published papers in relevant conferences or journals). Obviously, the author of a certain paper cannot be reviewer of her own paper.

Unfortunately, GraphDB does not provide any feature to create a TBOX / ABOX beyond SPARQL CONSTRUCT. To create the TBOX / ABOX, use an external tool for that purpose. There are several tools to create an ontology: tools providing an interface to create triples, such as Protégé[3] or VocBench[4], or programmatic tools (such as Jena or RDFLib). For this lab, we expect you to create a RDFS ontology using RDFLib (if you would like to explore the option of using OWL talk to the lecturer first).

Note: RDFLib[5] is a community-based powerful Python framework to create knowledge graphs, while Jena is a Java-based framework. Originally, Jena[6] was the tool to go, but in the recent years RDFLib has massively improved and it is much simpler to use.

Tasks:

1. Provide a graphical representation of your TBOX in the main pdf file (follow the principles used in the lectures), the output rdfs (or owl) file **and** additionally the code to generate the TBOX. The additional files must follow the naming notation explained.

B.2 ABOX definition

In this section we want to create an ABOX compliant with the TBOX above created. For this exercise you must reuse the data you built for the research publications domain from the *Assignment of Lab 1* and adapt it to this exercise.

Similar to the property graph lab, it is usual to convert CSV, JSON or even relational data into RDF, RDFS or OWL. The Knowledge Graph community has been very active creating tools to generate triples from any other source. For example, two prominent projects are Any23[7] and Open Refine[8] with its RDF extension[9] (which can directly connect to GraphDB). Nevertheless, it is highly unlikely that simply by using these tools you are able to generate

[3] `https://protege.stanford.edu`
[4] `http://vocbench.uniroma2.it/downloads`
[5] `https://rdflib.readthedocs.io/en/stable/`
[6] `https://jena.apache.org/`
[7] `https://any23.apache.org`
[8] `http://openrefine.org/`
[9] `https://github.com/stkenny/grefine-rdf-extension/wiki`

the required ABOX in real-world projects. In general, you will need RDFLib to create and manipulate ABOX triples.

Bear in mind that a sound ABOX must link the ABOX instances with the TBOX classes via rdf:type. Be aware, however, that depending on the inference regime entailment you consider for your exercise, you may save explicitly creating quite a few of them. Thus, in this lab we will positively evaluate the fact of not creating manually those rdf:type triples that might be generated by inference.

Tasks

1. In the main pdf file, explain the methodology used to define your ABOX from non-semantic data.

2. In the zip file, include your RDFLib code to create the ABOX programmatically and the resulting ABOX file. These files must follow the naming notation explained.

3. Specify the inference regime entailment you are considering. Explain what rdf:type links you saved to explicitly generate thanks to the inference rules.

4. Provide a summary table of your instances. Compute simple statistics about the resulting knowledge graph. For example, the number of classes, the number of properties, number of instances for the main classes and number of triples using the main properties.

B.3 Querying the ontology

Once the ontology has been created, load your TBOX and ABOX into GraphDB. Then, it will be ready to be queried. We are specifically interested in the benefits of having an explicit TBOX defined and enabling reasoning.

Tasks

1. Create two SPARQL queries and provide them in the main pdf file. We will evaluate how interesting the queries are and how interesting is the retrieved information. Show us you know how to squeeze a knowledge graph!

C Knowledge Graph Embeddings

In addition to their reasoning and querying capabilities, which you have already explored in the previous sections of this lab, the information contained in knowledge graphs can be further

utilized in downstream tasks, such as node classification or link prediction. This is typically achieved by transforming the elements of the knowledge graph (i.e., entities and relations) into low-dimensional vector representations, known as Knowledge Graph Embeddings (KGEs), as a vector is the typical expected input for Machine Learning algorithms. These embeddings are usually learned by exploiting the structural and semantic properties together with the topology of the knowledge graph.

In this section, we are going to learn how to create and exploit these embeddings for the graph that you have just created. Before starting, make sure to:

- Have read and understood the KGEs material provided in the campus.
- Have gone through the Collab Notebook tutorial with a toy example and key functionalities of the PyKEEN[10] library, which we are going to use to create the embeddings.

C.1 Importing the data

As a first step, you will have to import the data to Python and create training and test splits that are stratified. As shown in the tutorial, the data format expected is TSV, which you can directly obtain from GraphDB. Make sure to include only the desired data (e.g., you may decide to keep some information out of the KGE and use it for the exploitation task). For the stratification, you can rely on PyKEEN to correctly generate it.

Tasks

1. In the main pdf file, explain, if applicable, which data you selected for the creation of the KGEs.

C.2 Getting familiar with KGEs

As we have seen in class, there exist multiple models to generate KGEs, each with its own underlying assumptions and mechanisms. In this section, you will delve a bit deeper into these models to understand their differences, explore their drawbacks, and gain some mathematical intuition behind them. By understanding their differences and drawbacks, you will be better equipped to select appropriate models for specific tasks or datasets. Additionally, gaining some mathematical intuition behind these models will help you understand why they succeed (or fail) in capturing some of the semantics of KGs. To ground the discussion, in this section we will consider the *TransE*, *TransH*, *TransR*, *RotatE*, *DistMult* and *ComplEx* models.

[10] https://pykeen.readthedocs.io/en/stable/

The most basic model

First of all, we will start generating embeddings with *TransE*, a very simple and interpretable model. Do not worry about obtaining the optimal results, a model with a sensible hyper-parameter configuration will be enough. Once you have your learned embeddings, complete the following tasks:

Tasks

1. Choose a paper from the KG and answer the following questions in the main pdf file:
   - How would you find the most likely embedding vector, according to the TransE model, that would correspond to a paper cited by it?
   - If such a paper existed, how would you find the most likely embedding vector for its author?
2. Obtain the final vector from the previous question and identify the author in your KG whose embedding is closest (in terms of Euclidean distance) to it. Provide the code in the zip file following the naming notation explained.

Improving TransE

As you have observed, this simple method provides good interpretability but has some drawbacks, particularly its ability to properly model more complex relationships. Concretely, you will explore the problems encountered with One-to-Many/Many-to-One/Many-to-Many and symmetric relationships.

Tasks

For the following tasks, give the answer in the main pdf file:

1. Given a KG with the triples *(Author1, writes, Paper1)*, *(Author2, writes, Paper1)* and *(Author1, writes, Paper2)*, describe the optimal solutions that *TransE* would give to the embeddings of the entities.
2. From the models considered in this section, find one that has been created to (at least partially) solve the previous issue and briefly explain how it achieves so.
3. A relation stating that two authors collaborate with each other (e.g., collaboratesWith) is symmetric. With *TransE*, will the model give the same score to the triples *(Author1, collaboratesWith, Author2)* and *(Author2, collaboratesWith, Author1)*? Should they be the same?

4. Describe and sketch (in a simple 2D drawing), why with TransE symmetry is not generally achieved (i.e., unless a trivial and/or degenerate solution is obtained).

5. *RotatE* is a model that has been created to deal, among other things, with symmetry. Again, in a 2D figure, sketch which are the circumstances under which the symmetry *(Author1, relation, Author2)* and *(Author2, relation, Author1)* is achieved.

6. Finally, name (and explain) one of the considered models that allows for symmetry without any consideration.

C.3 Training KGEs

By now, you have seen that there exist multiple KGE models and have explored some of their differences. Now, you will learn the final KGEs before exploring applications with them. Therefore, in this task you should compare the evaluation results of various models (we suggest not more than 4), exploring also the effects of the training parameters, such as *embedding dimension* or *number of negative samples per positive sample*. Finally, choose one KGE model, justifying your decision.

Tasks

1. In the zip file, include the script(s) to generate the different experiments. The file must follow the naming notation explained.

2. In the main pdf file, explain which KGE models and hyperparameters have you chosen, and discuss the results of the experiments. You need to indicate which is the KGE model (and its hyperparameter configuration) that you will use for exploitation.

C.4 Exploiting KGEs

Finally, as a last exercise you will use the generated KGE for a Machine Learning application of your choice. As simple examples, you can perform unsupervised tasks (e.g., clustering of authors) or supervised ones (e.g., classification of papers based on some attribute that you have not included in the KG). Be creative!

Tasks

1. In the main pdf file, explain which task you solved using the KGEs, and provide any relevant information about the results, modeling steps, and other details you consider appropriate.

2. In the zip file, include the scripts from this tasks. The files must follow the naming notation explained.