Universität Leipzig

MAG Import

Documentation

Johannes Leupold, Philip Fritzsche, Timo Adameit

July 31, 2017

Contents

1 Introduction

Microsoft Academic Graph (MAG) is a large scale publication dataset. It contains information about scientific articles published in both journals and conferences paired with information about their authors and affiliations. This project aims to import Microsoft Academic Graph into Gradoop, the open source graph processing framework created by the Database Group at Universtät Leipzig. Therefore, the MAG model must be converted to make it compatible with the extended property graph model (EPGM) used in Gradoop. We are trying to extract the schema graph from the MAG dataset to match the schema with the one extracted from DBLP Graph, another academic publication dataset. The goal of this schema matching is creating a consolidated graph containing both the information from MAG and DBLP to combine advantages of both datasets. For instance, MAG has affiliation information and connects publications to a field of study hierarchy, while DBLP contains information about publication types and enables the creation of co-author relations.

The following sections contain information on the MAG dataset itself and the considerations in creating the parser and the importer. Finally, we will provide our schema matching of both datasets.

2 Microsoft Academic Graph

The Microsoft Academic Graph[1] "is a heterogeneous graph containing scientific publication records, citation relationships between those publications, as well as authors, institutions, journals, conferences, and fields of study". [2] It is used to support user experience in Bing, Cortana and Microsoft Academic. It can be accessed through the Microsoft Cognitive Services Academic Knowledge API. [2]

For our project, we are using an older dump of the MAG, created in February 2016. The parser will still be applicable to a current dump of the graph, because the schema hasn't changed since. The only problem in obtaining a current dump is that it is nowhere to be found. Since at the moment the graph can only be accessed through the Academic Knowledge API, it is very difficult to create such dumps.

2.1 Dimensions Of The Dataset

The MAG dataset contains a big number of publications and other information. Herrmannova and Knoth write [1]

> " It is also the largest publicly available dataset of citation data. As such, it is an important resource for scholarly communications research."

[1] `https://www.microsoft.com/en-us/research/project/microsoft-academic-graph`

Papers	126,909,021
Authors	114,698,044
Institutions	19,843
Journals	23,404
Conferences	1,283
Conference Instances	50,202
Fields Of Study	50,266

Table 1: Number of nodes in the MAG dataset, grouped by entity type, as of September 2016

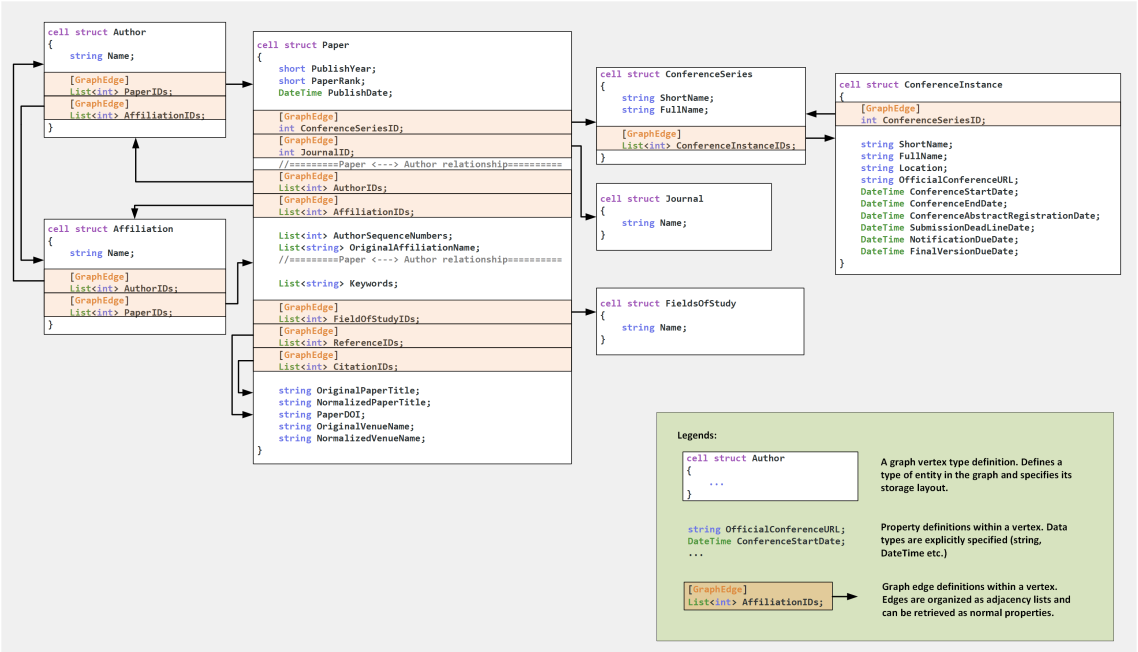


Figure 1: The current schema of the MAG dataset.

In their analysis of September 2016, they present their findings about the dimensions of the MAG (see Table 1).

The current schema of the MAG can be viewed in Figure 1.

2.2 Primitive Types

Different types of primitives (such as Node, Edge etc.) are found in the MAG. To successfully parse the graph, we need to analyse the primitive types and find a way to represent them using Gradoop primitives (`Vertex`, `Edge`, `Properties`). MAG contains four different types of primitives.

2.2.1 Node

A node represents an entity in the graph, for instance a paper. It has properties, which may contain foreign keys referencing other entity types. In the EPGM, a node can be represented as a `Vertex`.

Files containing node primitives:

- `Affiliations.txt`
- `Authors.txt`
- `ConferenceInstances.txt`
- `Conferences.txt`
- `FieldsOfStudy.txt`
- `Journals.txt`
- `Papers.txt`

2.2.2 Edge

An edge creates a connection between exactly two nodes and encodes a relationship between the entities represented by these nodes. The edge may have properties. The EPGM representation of an edge would be `Edge`.

Files containing edge primitives:

- `FieldOfStudyHierarchy.txt`
- `PaperKeywords.txt`
- `PaperReferences.txt`

2.2.3 Hyperedge

A hyperedge is a connection between three or more nodes to encode polyvalent relationships between these entities. The MAG contains only one hyperedge representing the three-side relationship between authors, papers and affiliations, expressing "The author writes the paper while working at the affiliation". In the EPGM, a hyperedge can be represented by one `Vertex` representing the relationship and multiple `Edge`s to the involved nodes or by multiple `Edge`s which can be clearly assigned to each other.

Files containing hyperedge primitives:

- `PaperAuthorAffiliations.txt`

2.2.4 Multi Attribute

A multi attribute is an additional property for a node. Other than basic properties it can have multiple values for the same entity. Therefore, it can't be stored in the same file due to the TSV (tab separated values) format. The EPGM equivalent of a multi attribute should be single property.

Files containing multi attribute primitives:

- `PaperUrls.txt`

3 Parsing

The MAG parser aims to create an internal representation of the graph independent from further processing, e.g. the import into Gradoop. Therefore, we pursue a separation of concerns approach. We created a callback interface to allow the user of our project to specify further processing themselves. The parser takes an implementation of this interface as a constructor argument and passes all parsed objects to this `ElementProcessor` for further consideration. The processor could be our own Gradoop importer or a simple statistics tool, for example.

The input files get processed line-wise, while the lines are splitted at tabs, like you would a TSV parser expect to do.

All primitives contained in the MAG are abstracted into the class `MagObject`. Each `MagObject` has a `TableSchema` containing a description of the objects structure (property names and types). The schema also contains the object type, a representation of the primitive type described in Section 2.2. Possible values of this values are defined as `enum ObjectType {NODE, EDGE, EDGE_3, MULTI_ATTRIBUTE}`. For each input file, there is a `TableSchema` which is named after the node or edge type. In the callback implementation all of these information can be accessed.

There are multiple property types, some of which are basic types, others have special meanings for hyperedges. The basic types are:

- `ID`: The unique key for an entity. Only used on nodes.
- `ATTRIBUTE`: A generic property. Can be used on all primitive types.
- `KEY`: A foreign key, therefore contains the ID of an entity. This field type is used on nodes as a foreign key and on edges and hyperedges as link partners for the edge.
- `IGNORE`: The field is not needed (redundant information) and is ignored in parsing. Can be used on all primitive types.

Some field types are created solely for the purpose of parsing hyperedges.

- `ATTRIBUTE_1`: When the hyperedge is split into two edges, this field gets used as `ATTRIBUTE` only for the first of those edges.
- `KEY_1`: This field is a link partner only for the first edge created. It is attached as an `ATTRIBUTE` to the second edge to connect both edges.

- KEY_2: The field is a link partner only for the second edge created. In the creation of the first edge, it is ignored.

All needed TableSchemas are bundled into one InputSchema for easier access. The parsing order is defined by primitive type. EDGEs are processed first, with EDGE_3 following. After that, we process all NODEs and at last the MULTI_ATTRIBUTES. The parsing order inside one of those groups is non-deterministic.

4 Import Into Gradoop

The Gradoop importer is an implementation of the MAG parser ElementProcessor callback. It takes care of all primitive types and creates the graph as JSON output in the Gradoop format once parsing is finished. The parsed MagObjects are processed regardless of schema name, only the object type is taken into consideration. The corresponding package contains two classes: ImportMain, which takes care of the full import process, and GradoopElementProcessor which creates the EPGM elements from the parsed MsagObjects. The following section covers the process of creating EPGM elements from the different object types.

4.1 Primitive Representation

4.1.1 NODE

The transformation of a NODE to the corresponding EPGM Vertex isn't very complicated. The processor creates an InputVertex and uses the schema name of the NODE as its label. All fields marked ATTRIBUTE are joined into a single Properties object which is assigned to the vertex. The ID field of the node is used as vertex ID. All vertices are kept in an ArrayList for later creation of datasets.

The KEY fields are extracted from the node using the helper function getForeignKeys. Each foreign key is transformed into an ImportEdge, going from the just created node to the referenced node. The edge is labeled [schema name]|[target schema name]. The edge id is [ID]|[target ID].

Papers' properties are additionally inserted into a HashMap, using their ID as the key for later addition of the URLs multi attribute.

4.1.2 EDGE

An EDGE gets directly transformed into an InputEdge. The edge gets labeled with the schema name and uses the ID [source ID]|[target ID]. The ATTRIBUTE fields are extracted and converted into a Properties object which is attached to the ImportEdge. All edges are kept in an ArrayList.

4.1.3 EDGE_3

For the processing of EDGE_3, the hyperedges are split into two single edges, one connecting papers and authors and the second connecting authors to affiliations. The second one is annotated with the paper ID to maintain the relationship between both edges.

Here, the additional field types are taken into consideration. A modified helper function, convertAttributesEdge3, is used to extract Properties from the MsagObject. It pays attention to the context (first or second run) and only returns the correct attributes. A similar process is applied to the link partners: the function getForeignKeysEdge3 returns only those references, needed in the edge currently created.

Both edges are transformed into ImportEdges and labeled like normal edges, but with _1 and _2 suffixes.

4.1.4 MULTI_ATTRIBUTE

A MULTI_ATTRIBUTE must be added to the Vertex it is associated with. The association is determined using the single KEY field of the attribute. After that, the associated paper's properties are loaded from the previously created map. The property is added to a List<PropertyValue> which is stored in the Properties object.

4.2 Graph Creation

After parsing is finished, the resulting lists of vertices and edges are retrieved from GradoopElementProcessor and converted into Flink DataSets. The datasets are used to create a LogicalGraph which is then written to a JSONDataSink.

5 Schema Extraction

The schema extraction is achieved using the Gradoop Grouping operator. The grouping reads JSON input files and writes DOT output files. The schema graph maintains the attributes used in the full graph representation. After the extraction, the schema graph can be visualized using the graphviz library.

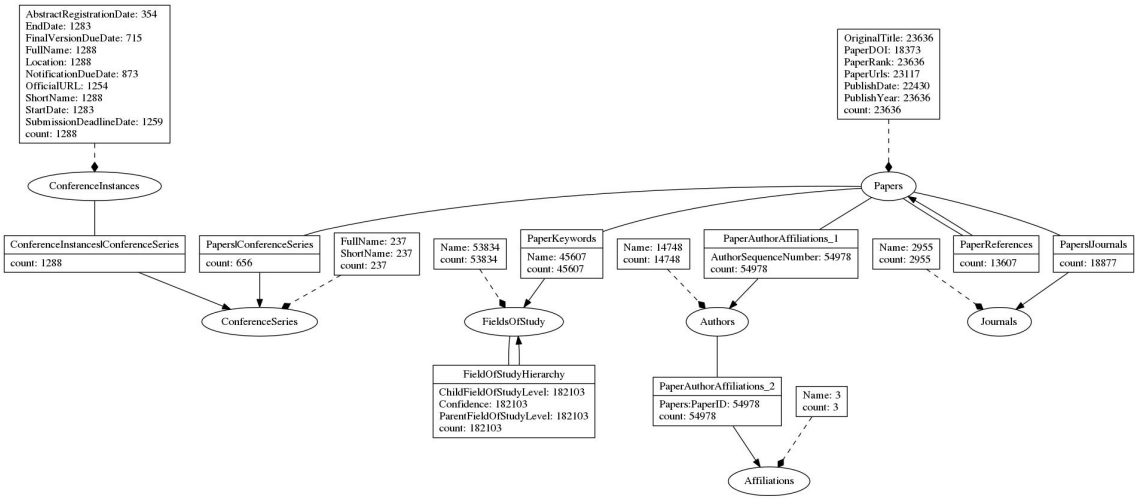
5.1 General Procedure

The schema graph extraction is done in multiple steps. These steps are necessary, since the grouping operator in gradoop drops all attributes when creating the grouped graph. The following steps are performed to create a grouped graph containing all attributes that are used for the specific node or edge type at least once.

1. Read the input graph from the specified input path, using a `JSONDataSource`.
2. Perform a `Transformation` on the input graph, transforming all vertex attributes and edge attribute to a single attribute named `attributes`, containing a map, mapping all attributes to 1. The transformation helper class to achieve this is called `JoinAttributes`.
3. Group the resulting graph, using vertex and edge labels as grouping criteria. While doing so, use an `PropertyValueAggregator`, which counts the occurrences attribute names on every instance of a given node or edge type and counts the occurrences of every attribute. The aggregator `MapSumAggregator` provides this functionality.
4. Transform the resulting schema graph again, splitting the previously created map, now containing the attribute counts, contained in the property `attributes`, back to single attributes with their count as value. The class `SplitAttributes` does so.
5. Write the schema graph to the specified output path, using a rewritten version of the `DOTDataSink` which supports writing of attributes for both, vertices and edges.

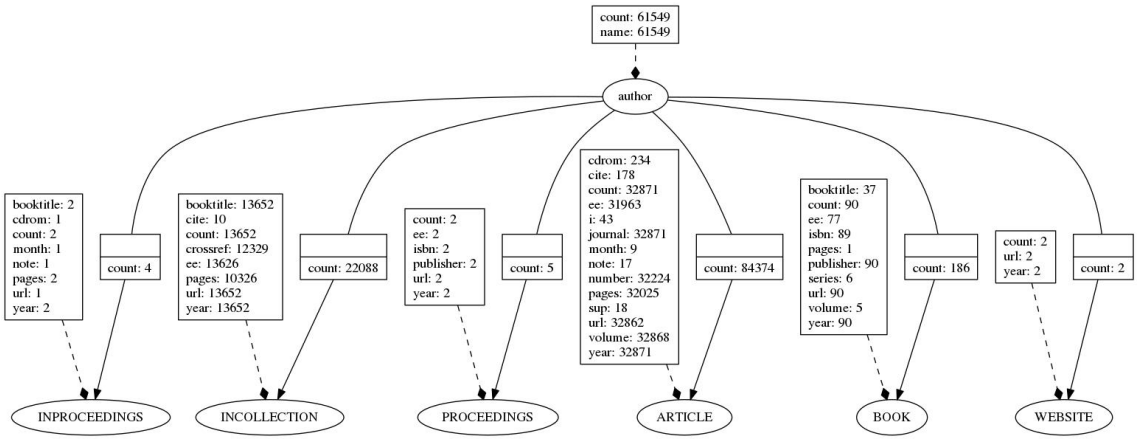
5.2 Microsoft Academic Graph

The following schema graph was created from a representative subgraph of MAG. The subgraph was created using all papers written at Universität Leipzig.



5.3 DBLP Graph

The following figure shows the schema graph of the DBLP graph. It was created from a small subset of the DBLP graph, thus not containing all publication types.



6 Schema Matching

The matching of MAG and DBLP Graph is not an easy task to accomplish. Both graphs differ greatly in means of structure. For instance, DBLP graph contains multiple types of publications, while MAG only has one supertype named *Paper*. The attribute sets of the DBLP publications and MAG papers are also not compatible regarding their structure. The only trivial matching between MAG and DBLP can be done in merging the *Author* nodes. Both have exactly the same structure and attributes (only `name`). Every other node from the DBLP graph must be examined separately.

On another note, the schema graph of DBLP seems to have an issue: The title attributes are missing. The matching of two publications without a title attribute can be difficult if not impossible under some circumstances. After examining the DBLP dataset and the Gradoop JSON files, we conclude that the DBLP parser doesn't even add this attribute - despite it being present in the dataset. For the following considerations we will assume, that the title attribute is present on all DBLP publications.

In the following paragraphs, we will work on a matching for the different publication types of DBLP. First of all, a obvious matching has to be done: the `title` attribute of DBLP publications should be matched to the `OriginalTitle` attribute of the *Paper* node of MAG, while `year` can be matched to `PublishYear`. The `url` attribute can be matched to *Paper*'s `PaperURLs`. Using this matching, we can already identify publications which are present in both graphs. Furthermore, all DBLP publication nodes need to be matched to the MAG *Paper* node, or maybe a new combined *Publication* node.

ARTICLE The `cite` attribute of the publication type could in some way be matched to the *PaperReferences* edge of MAG. The attribute has to be dereferenced in order to find the edge. The `journal` attribute can be matched to MAG by following the *Papers*/*Journals* edge and mapping it to the *Journal*'s `name`. `year` combined with `month` could be used to match to `PublishDate`. Last, the `ee` attribute, containing a DOI URL can be matched to `PaperDOI` in MAG. All remaining attributes of MAG and DBLP are additional information and can't be matched to each other.

BOOK The usage of the `booktitle` attribute on books, which also have a `title` attribute is not exactly clear. But if present, `booktitle` could be matched to `OriginalTitle` in MAG. The `ee` identifier can, after transformation, be mapped to `PaperDOI`.

INCOLLECTION The `booktitle` attribute could match to `OriginalTitle` of the *Paper* node if not `title` already matches. The `crossref` attribute can be matched to the *PaperReference* edge like described above. `ee` is matched to `PaperDOI` like with the preceeding publication types.

INPROCEEDINGS `booktitle` should be treated like with *INCOLLECTION*, or, if the proceedings carry the name of the conference, it can be used to match to *ConferenceSeries*' `FullName` by following the *Papers*/*ConferenceSeries* edge. `year` combined with `month` could be used to match to `PublishDate`.

PROCEEDINGS Besides being matched to `OriginalTitle`, the `title` attribute of *PROCEEDINGS* could be matched to *ConferenceSeries*' `FullName` or even to *ConferenceInstances*' `FullName`. The `ee` identifier can, after transformation, be mapped to `PaperDOI`.

WEBSITE *WEBSITE* does not provide any further attributes to be matched.

All remaining node and edge types of MAG do not have an equivalent part in DBLP. They can be considered additional information that can be directly tranferred into a merged graph.

References

[1] Drahomira Herrmannova and Petr Knoth. An analysis of the microsoft academic graph. *D-Lib Magazine*, 22(9/10), 2016.

[2] Microsoft Research. Microsoft academic graph. `https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/`, July 2017.