B. Mitschang et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2017),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 629

The Big Picture: Understanding large-scale graphs using Graph Grouping with GRADOOP

Martin Junghanns[1] André Petermann[1] Niklas Teichmann[2] Erhard Rahm[1]

Abstract: Graph grouping supports data analysts in decision making based on the characteristics of large-scale, heterogeneous networks containing millions or even billions of vertices and edges. We demonstrate graph grouping with GRADOOP, a scalable system supporting declarative programs composed from multiple graph operations. Using social network data, we highlight the analytical capabilities enabled by graph grouping in combination with other graph operators. The resulting graphs are visualized and visitors are invited to either modify existing or write new analytical programs. GRADOOP is implemented on top of Apache Flink, a state-of-the-art distributed dataflow framework, and thus allows us to scale graph analytical programs across multiple machines. In the demonstration, programs can either be executed locally or remotely on our research cluster.

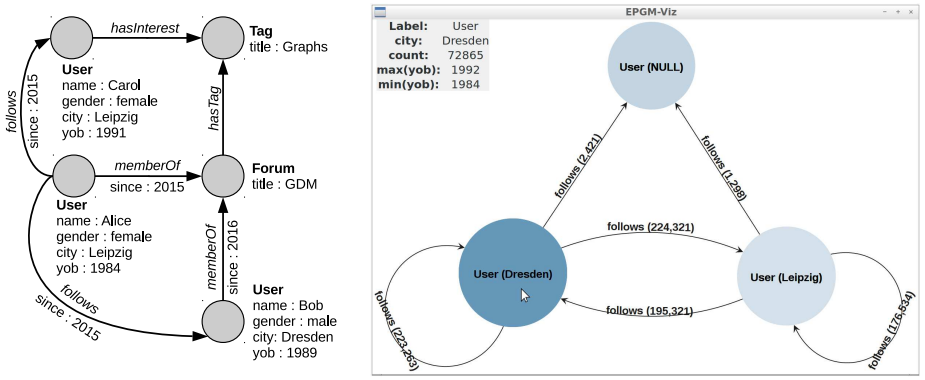
Keywords: Graph Analytics, Graph Algorithms, Distributed Computing, Dataflow systems

1 Introduction

Graphs are an intuitive way to model, analyze and visualize complex relationships among real-world data objects. The flexibility of graph data models and the variety of existing graph algorithms made graph analytics attractive to different domains, e.g., to analyze the link structure of the world wide web [BP98], users of a social network [Ne10], protein interaction in biological networks [Pa11] or business process executions in enterprise data [Pe14]. In these domains, graphs are often heterogeneous in terms of the objects they represent. For example, vertices of a social network may represent users and forums while edges may express friendships or memberships. Further on, vertices and edges may have associated properties to describe the respective object, e.g., a user's name or the date a user became member of a forum.

The property graph model [RN10, An12] is an established approach to model heterogeneous networks. Figure 1(a) shows a property graph that represents a simple social network containing multiple types of vertices (e.g., *User* and *Forum*) and edges (e.g., *follows* and *memberOf*). Vertices as well as edges are further described by properties in the form of key-value pairs (e.g., *name : Alice* or *since : 2015*). However, while small graphs are an intuitive way to visualize connected information, with vertex and edge numbers increasing up to millions or billions, it becomes almost impossible to understand the encoded information by mere visual inspection. One way to reduce complexity is the grouping of vertices and edges to so-called *super vertices* and *super edges* of a *summary graph* supporting the analyst in extracting and understanding the underlying information [THP08, Ch08, Ju17].

[1] University of Leipzig, Database Group & ScaDS Dresden/Leipzig,
[junghanns,petermann,rahm]@informatik.uni-leipzig.de
[2] University of Leipzig, Database Group & ScaDS Dresden/Leipzig, teichmann@studserv.uni-leipzig.de



(a) Example of a Social Network

(b) Summary graph in EPGM-Viz; Properties are shown for the selected vertex

Fig. 1: (a) shows an example social network; (b) shows the summary graph of a large social network grouped by the user's city including aggregate values expressing the oldest and youngest user's age per city and the number of edges among cities.

Graph grouping allows structural summarization and attribute aggregation by user-defined vertex and edge properties. In Figure 1(b), the users of a social network are grouped by their *city* property, i.e., each super vertex in the summary graph represents all users that live in the same city. As the property graph model is schemaless, users may not necessarily provide certain properties. Such users are grouped within a dedicated *NULL* vertex. Super edges represent mutual relationships among the grouped vertices of the input graph. For example, an edge between Dresden and Leipzig in the summary graph represents all edges of that same type among users from Dresden and Leipzig. Besides the structural summarization, graph grouping allows for attribute aggregation on super vertices and super edges. In our example, each super vertex and super edge stores the number of elements it represents. Super vertices additionally store the minimum and maximum year of birth among the users.

In the demonstration, we present the graph grouping operator [Ju17] of Gradoop [Ju16], an open-source graph analytical system that implements the so-called Extended Property Graph Model and that supports declarative operations on single property graphs as well as collections of these. For example, given a social network similar to Figure 1(a), the summary graph of Figure 1(b) can be easily declared by the following script:

```
summaryGraph = socialNetwork
  .subgraph(
    (vertex -> vertex[:label] == 'User'),
    (edge -> edge[:label] == 'follows'))
  .groupBy(
    [:label, 'city'], [COUNT(), MIN('yob'), MAX('yob')],
    [:label], [COUNT()])
```

We will demonstrate how Gradoop and its operators can be used to compute expressive summary graphs. Since Gradoop is implemented on top of Apache Flink [Ca15], a distributed state-of-the-art dataflow framework, demo programs can be executed either locally or on our research cluster without modifying the program. In the following section, we provide a brief overview about Gradoop and the Extended Property Graph Model. In Section 3, we describe our demonstration scenario in more detail.

2 Graph Grouping with GRADOOP

GRADOOP is a system for declarative graph analytics supporting the combination of multiple graph operators and algorithms in a single program. Graph data is represented within the Extended Property Graph Model (EPGM) [Ju16], which is based on the property graph model [RN10], i.e., on directed multigraphs supporting identifiers, labels and named attributes (properties) for vertices as well as edges. As an extension, the EPGM supports the concept of *logical graphs*, which are logical partitions of a base graph. Thus, it is possible to analyze single graphs as well as collections of these. Logical graphs also support labels and properties, for example, to represent communities in a social network and to store their number of users. Furthermore, logical graphs and collections are input and output of EPGM operators which enables the composition of complex analytical programs. GRADOOP already provides operator implementations for graph pattern matching, subgraph extraction, graph transformation, set operations on multiple graphs as well as property-based aggregation and selection [Ju16]. Graph analytical programs are declared using a Java API representing the domain specific language GrALa (Graph Analytical Language). Below the user-facing API, graph operators and algorithms are mapped to the programming abstractions provided by Apache Flink and thus, their execution can be scaled out across a cluster of machines. The source code of GRADOOP is available online under GPL license[4].

Graph grouping extends the set of available graph operators in GRADOOP. The operator takes a single logical graph as input and computes a new logical graph, which we call a *summary graph*. The operator signature for graph grouping in GrALa is defined as follows:

```
LogicalGraph.groupBy(
  vertexGroupingKey[], vertexAggregateFunction[],
  edgeGroupingKey[], edgeAggregateFunction[]) : LogicalGraph
```

While the first argument is a list of vertex grouping keys, the second argument refers to a list of user-defined or system-provided vertex aggregate functions. Analogously, the third and fourth argument are used to define edge grouping keys and edge aggregate functions. In our introductory example, the operator is parameterized using the symbol `:label` and the property key `city` as vertex grouping keys. We use system-provided aggregate functions to count the elements inside each group and to determine minimum and maximum year of birth. The resulting super vertices adopt the label, the grouping property (e.g., *city : Dresden*) and the results of the aggregate functions (e.g., *count : 72,865*). Edges are implicitly grouped by the super vertices of their incident vertices and explicitly by their label and counted. In the example, one can also see the composition of subgraph extraction and graph grouping. First, a subgraph containing solely vertices of type *User* and edges of type *follows* is extracted from the social network and forwarded to the grouping operator. The resulting summary graph can be either used as input for another operator (e.g., pattern matching), stored in a data sink or visualized.

[4] `http://www.gradoop.com`

3 Demonstration Description

In our demonstration, we show how Gradoop can be used to compute summary graphs from social network data. We provide three example programs to cover distinct aspects of graph grouping: (1) subgraph grouping analogous to our example, (2) type-dependent grouping to declare grouping keys on individual labels and (3) graph grouping along dimensional hierarchies for Graph OLAP scenarios [Ch08]. Example (2) and (3) require the graph transformation operator to pre-process the input graph before the actual grouping. Visitors are also invited to modify our example programs or to write new ones including further operators (e.g., pattern matching) and plug-in algorithms (e.g., community detection).

The programs are presented and developed using the Gradoop Java API and executed either locally on the demonstration laptop or remotely on our research cluster. We provide real-world graphs and artificial social network data with up to 10 billion edges generated by the LDBC data generator [Er15]. Gradoop provides multiple ways to visualize the resulting summary graphs. In example (1), we will use our graph visualization EPGM-Viz[5] (Figure 1(b)), in example (2), we will use the DOT output and GraphViz[6], and in example (3), we will utilize the Neo4j graph database to visualize the results[7].

4 Acknowledgments

This work is partially funded by the German Federal Ministry of Education and Research under project ScaDS Dresden/Leipzig (BMBF 01IS14014B).

References

[An12] Angles, R.: A Comparison of Current Graph Database Models. In: Proc. ICDEW. 2012.

[BP98] Brin, Sergey; Page, Lawrence: The Anatomy of a Large-scale Hypertextual Web Search Engine. In: Proc. WWW. 1998.

[Ca15] Carbone, P. et al.: Apache Flink™: Stream and Batch Processing in a Single Engine. IEEE Data Eng. Bull., 38(4), 2015.

[Ch08] Chen, C.; Yan, X.; Zhu, F.; Han, J.; Yu, P. S.: Graph OLAP: Towards online analytical processing on graphs. In: Proc. ICDM. 2008.

[Er15] Erling, O. et al.: The LDBC Social Network Benchmark. In: Proc. SIGMOD. 2015.

[Ju16] Junghanns, M.; Petermann, A.; Teichmann N.; Gómez K.; Rahm E.: Analyzing Extended Property Graphs with Apache Flink. In: Proc. SIGMOD NDA Workshop. 2016.

[Ju17] Junghanns, M.; Petermann, A.; Rahm E.: Distributed Graph Grouping with Gradoop. In: Proc. BTW. 2017.

[Ne10] Newman, M.: Networks: An Introduction. 2010.

[Pa11] Pavlopoulos, G. A. et al.: Using graph theory to analyze biological networks. BioData Mining, 4(1), 2011.

[Pe14] Petermann, A.; Junghanns, M.; Müller, R.; Rahm, E.: BIIIG: Enabling business intelligence with integrated instance graphs. In: Proc. ICDE Workshops. 2014.

[RN10] Rodriguez, M. A.; Neubauer, P.: Constructions from Dots and Lines. arXiv, 2010.

[THP08] Tian, Y.; Hankins, R. A.; Patel, J. M.: Efficient Aggregation for Graph Summarization. In: Proc. SIGMOD. 2008.

[5] https://github.com/dbs-leipzig/EPGM-Viz

[6] http://www.graphviz.org/

[7] https://github.com/s1ck/flink-neo4j