Exploring Large Scholarly Networks with Hermes

Gabriel Campero Durand
University of Magdeburg, Germany
campero@ovgu.de

Anusha Janardhana
University of Magdeburg, Germany
anusha.janardhana@ovgu.de

Marcus Pinnecke
University of Magdeburg, Germany
pinnecke@ovgu.de

Yusra Shakeel
University of Magdeburg and METOP GmbH, Germany
shakeel@ovgu.de

Jacob Krüger
Harz University and University of Magdeburg, Germany
jkrueger@hs-harz.de

Thomas Leich
Harz University and METOP GmbH, Germany
tleich@hs-harz.de

Gunter Saake
University of Magdeburg, Germany
saake@ovgu.de

ABSTRACT

Every year, the number of scientific publications increases, adding complexity to the networks of collaborations, citations, and topics, in which papers are embedded. Analyzing these networks with efficient tools is important to help researchers identify relevant works and understand scientific impact. However, available tools face several limitations, indicating that there is still room for improvement. We present Hermes, a prototype for exploring large and heterogeneous scholarly networks. Hermes allows users to seamlessly navigate diverse types of networks within a single graph, spanning hundreds of millions of nodes and relationships. Our prototype achieves reasonable responsiveness on commodity hardware through: *a)* comprehensive indexing, *b)* a careful coupling of a graph database and a search engine, and *c)* incremental processing of temporal queries. In this demonstration, we explain the techniques we adopt and illustrate how to use Hermes for exploring the Microsoft Academic Graph.

1 INTRODUCTION

The number of scientific publications increases every week, creating a large set of data about authors, citations, and collaborations. As a result, it becomes more and more challenging to determine which publications are relevant for a specific research goal [16]. In fact, new terms and fields, such as *big scholarly data* [22] and *science of science* [24], are emerging to cover data management challenges and novel analysis questions, which arise from scholarly information growth.

Scholarly network analysis (SNA): The traditional metrics of science (e.g., the *H-index*), that rely on network-unaware statistics, have been questioned by scholars [23], making a case for improving the analysis of scholarly networks by complementing such metrics with content & network-based analysis.

Types of scholarly networks: At least 7 types of networks are usually considered for SNA [23]; coauthorship/collaboration, citations, co-citations, bibliographical coupling, topics, co-words, and heterogeneous networks.

SNA and heterogeneous networks: Restrictive choices of network type and aggregation entity can limit the generalizability of SNA. To avoid this, researchers recommend to employ heterogeneous networks, and methods capable of extracting value from these networks [23]. One of such methods is FutureRank [15], an approach for relevance ranking based on combining HITS, over a collaboration graph, with PageRank, over a citation graph.

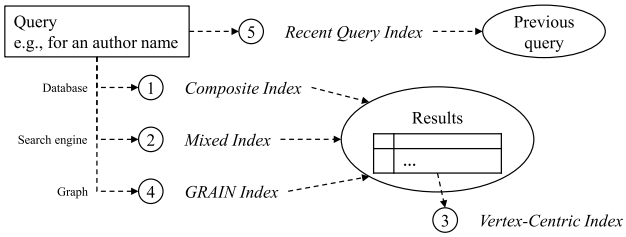

Figure 1: Indexing approaches in Hermes.

Data management for SNA: SNA involves studies at macro (global), meso (community), and micro (individual)-levels of a network. Considering the scale of networks, in addition to the need for managing heterogeneity of entities and analysis (e.g. content & statistical analysis), efficient tool support for SNA poses several data management challenges. Among them, the foremost could reasonably be resource management: to provision for ad-hoc SNA within reasonable response times, while enabling queries across different network scales and representations. Other challenges include data cleaning, provenance, management of analysis results, and integration with external sources.

Existing tools for large scale SNA: Several scientific digital libraries and search engines exist that index large amounts of publications [12], however their services often fall short in several aspects, including consistency, available metrics, and possibilities for ad-hoc SNA [11]. Regarding the latter, for example, Arnetminer[1] [20] internally utilizes several kinds of network analysis, however only local exploration of ego networks is currently offered to its Web users. Among tools supporting SNA, CiteSpace [4] , Pajek [2] , Gephi [1] , igraph, [6] and NetworkX [10] are some of the most popular.

Graph databases and SNA: Although RDF technology has been widely researched for semantic publishing [3], to date there is little research in specialized graph database technologies for SNA. Within our work we consider this research gap.

In this demo paper we describe the first version of Hermes, our proposed tool for SNA based on graph technologies. The core technical insight behind our work at this stage is in exploiting opportunities for close search engine/graph database coupling in SNA tasks.

[1] https://aminer.org/

With our tool we seek to enable users to seamlessly interact with large scale heterogeneous networks, performing ad-hoc SNA at different granularities, either through our APIs, or through native graph/search-engine query languages.

Overall, we hope to encourage the audience to use our tool for multiple purposes. We aim to: 1) help the audience in identifying relevant publications and perfoming SNA tasks; 2) receive feedback to improve Hermes; 3) introduce to the audience the key indexing solutions that improve the performance of Hermes; and 4) employ Hermes for user studies to investigate what could constitute representative SNA workloads. Based on our work, we intend to publish corresponding anonymous datasets of usage patterns as open-source artifacts—thus providing data for workload characterization to the research community. Furthermore, our tool will be freely available.

2 HERMES

In this section we describe the current architecture of our tool, and we present two techniques we currently explore in Hermes for improving its data access.

2.1 Architecture

For indexing and primary storage, Hermes relies on a search engine, i.e., Elasticsearch, and a database, i.e., Cassandra. These two components are integrated through a property graph database, JanusGraph[2]—which can also be configured to other backend indexes and storages. JanusGraph uses the TinkerPop3[3] framework, and the corresponding Gremlin query language. Data access in Hermes is provided via a web-interface with data management supported by Gremlin and Elasticsearch servers.

JanusGraph provides three predefined types of indexes:

- *Composites* are traditional indexes tuned for point-queries. They rely on the primary storage. They cover one or multiple keys of either vertexes or edges of the stored graph.
- *Mixed* indexes are based on the search engine. They enable inexact querying, and relevance boosting.
- *Vertex-centric* indexes are supported by the primary storage. They are included in the same storage space of a vertex. Instead of indexing all entries in the database (global)—as the former two indexes—they only index the set of edges attached to a vertex (local). This allows to traverse faster through edges while filtering on labels and properties.

Apart from these predefined indexes, users of property graph databases can also use the "graph itself as an index". This is a straight-forward approach, which we, for lack of a better term, call GRAIN (graph as an index) [8]. In GRAIN meta-vertexes are added to the modeled data, acting as roots and nodes of a search tree, such that they lead within limited hops to a set of expected entities. A practical enhancement is to denormalize properties of the target vertexes unto the linking edges, such that filtering can be applied on the edges without visiting any vertex. In our system, the GRAIN approach is additionally adopted by using a set of meta-vertexes with temporal expiration. They serve as a cache for previous query results.

Within Hermes, we adopt a comprehensive indexing strategy that exploits all these opportunities. In Figure 1, we show an overview. We remark that the numbering corresponds only to the order in which we introduced each index.

[2]http://janusgraph.org/
[3]http://tinkerpop.apache.org/

Listing 1: Alternative temporal queries with Gremlin.

```
titanGraph.V().
   hasLabel("journal").
   has("retiredAt", P.gte(year0)).
   has("createdAt", P.lte(year0)).count().next();

titanGraph.V().
   hasLabel("journal").
   has("createdAt", year0+i).count().next();
```

2.2 Incremental Processing for Temporal Queries

Analyzing the evolution of publication networks can yield interesting findings. For example, studies on the Microsoft Academic Graph [18] have found the number of publications per year to be following an exponential growth for the last century—doubling every 12 years, with the top 1% of publications continuously accounting for around a quarter of citations each year [7]. Similar analyses with more localized perspectives can have practical value for researchers. For instance, they could help to assess the impact of given conferences over time or to identify high impact research topics.

Several works consider the formalization and evaluation of temporal graph queries—both, in graph databases [17], and in processing engines [13, 14]. Building on studies in this area [9], we design our temporal queries in Hermes by creating a type of meta-vertex, which we call *logger vertexes*. The set of edges in such vertexes provide a relative time-line for when items appear in the world represented by a graph. Analyzing the sequence of edges in a logger vertex allows reconstructing a graph from a certain point in time to another. Through this, it is possible to implement incremental computation of temporal data over a graph model that accumulates several snapshots.

We achieve incremental computation in Hermes by hand-tuned query rewriting. In Listing 1, we provide an example of a Gremlin query for identifying the number of journals on a year-by-year basis throughout a specified period. Instead of adding up the number of items for all years within the interval, we only compute the existing journals for the first year. Then, we keep a rolling total, based on the number of journals created or retired in each successive year.

Recently, authors have considered automated rewriting of multi-snapshot queries for co-scheduling tasks by their common steps—in a style similar to SIMD processing [21]. The authors call this approach *Single Algorithm Multiple Snapshots*. Unlike their work, we do not store separate snapshots, and we do not include automated rewrites in our tool.

In other study, groundwork for *incremental view maintenance* on property graph databases has been proposed, over a formalization of OpenCypher [19]. In contrast, the rewrites for incremental processing illustrated with our current prototype are hand-tuned, and require further standardization.

2.3 Query Rewrites Across Graph and Search Engine Representations

Similar to other graph databases, such as Neo4j[4], JanusGraph can integrate a search engine to support full-text queries. This provides a set of search engine functionalities, which are either

[4]https://neo4j.com/developer/elastic-search/

Listing 2: Rewrite of degree centrality (DC) calculation for Elasticsearch.

```
TermsBuilder termsBuilder = AggregationBuilders.terms(aggregationName).
    field("idOfTargetVertex").size(numberOfVertices);

XContentBuilder contentBuilder;
contentBuilder = JsonXContent.contentBuilder().startObject();
termsBuilder.toXContent(contentBuilder, ToXContent.EMPTY_PARAMS);
contentBuilder.endObject();

SearchRequestBuilder searchRequest= esClient.prepareSearch("index").
    setTypes("edges")).
    setQuery(QueryBuilders.termQuery("edgeLabel","label")).
    setAggregations(contentBuilder);
response = searchRequest.execute().actionGet();
```

offered through the graph API (e.g., launching so-called *indexQueries*), or directly through the search engine APIs. After verifying the potential of *external* access in previous work [8], we seek to leverage this concept in Hermes to improve its functionality by rewriting queries for the search engine APIs.

To make the case for such rewriting, we consider an example degree centrality (DC) calculation in Gremlin that has to be rewritten for the search engine. The DC of vertexes is a measure to understand the actors of a network. It can be defined as the number of edges (either incoming or outgoing) connected to vertexes. The in-DC of a given vertex indicates how *prominent* it is within the network, while the out-DC indicates how *influential* it is. The average vertex DC for a graph is the average of the DCs of its vertexes. In Gremlin, we can express the in-DC computation as a traversal that starts from all vertexes with an incoming edge having a specific label—grouping them by the vertex ID, and the number of edges. Here is an implementation based on recommendations by the TinkerPoP community[5]:

```
g.V().in("label").inV().group().by(__.ID()).by(__.inE(label).count())
```

For the search engine, we can increase the utility of the indexed data by adding to each edge the JanusGraph ID of the connected vertexes. This design decision amounts to a limited use of denormalization, with few chances for inconsistency as the IDs of connected vertexes do not change. Finally, we can construct the average DC calculation as a single Elasticsearch term query that searches for a given value over all edge labels. This query aggregates the count of results based on the IDs of vertexes (either the source or the target vertex). The result is an ordered list of vertexes and number of edges to which they are either the source or the target. We show a prototypical implementation in Listing 2.

For an initial evaluation of this rewrite we use the Pokec dataset, an online social network, and a commodity machine[6]. In Figure 2, we show the results, which indicate a speedup of 150x over the original, by employing the second query approach.[7]

The fundamental reason for the differences in performance is that the queries in fact map to entirely different algorithms. A Gremlin traversal begins by querying a given set of vertexes (either using a composite index or a full-table scan). For each match, Gremlin counts the number of incoming edges with a specific label, groups the results by vertex IDs, and returns these

[5]http://tinkerpop.apache.org/docs/3.2.1-SNAPSHOT/recipes/#degree-centrality
[6]We used an Intel CoreTM i7-2760QM CPU (2.40GHz) processor with 8 cores and 7.7 GiB of memory.
[7]We would like to add the disclaimer that the performance gains we report are specific to the queries and database we selected. Further comparisons are pertinent to assess the benefits of these rewrites for other queries and databases.

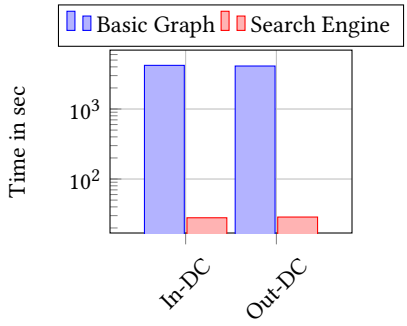

Figure 2: Response times for computing the average degree centrality (DC) on the Pokec dataset.

values. In contrast, the Elasticsearch implementation relies on a simpler representation of the same data: A set of documents that stand for the edges, further indexed with Lucene's inverted indexes. These serve well term-based searches, such as our *entry query*. Another index, covering as terms, the ID of source vertexes, can be matched with the document numbers found by our *entry query*. At last, the resulting collection is aggregated by counting for each term the number of matching edge documents.

In spite of the benefits, there are limits that need to be considered in adopting a search engine for graph tasks in a multi-store context. Namely, the *impedance mismatch* between domains, *inadequate performance optimizations* in the search engine , and *overheads from application-level mapping* of items between the different storage engines.

3 DEMONSTRATION OVERVIEW

Data: Our preloaded dataset is the complete Microsoft Academic Graph [18] as of early 2017, which is a property graph of scholarly publications. It is compiled and made publicly available by *Microsoft*. The graph consists of six entities: Field of study, author, institution, paper, venue, and event. Furthermore, the graph includes six standard relationships: Citations among papers, paper authorship, the venue or journal a paper was published at (or a field of study if no venue is available), the institutional affiliations of authors, and the relationships of events to venues. Overall, the Microsoft Academic Graph comprises around 166 million papers, representing a heterogeneous and large dataset for our tool.

Considering that the audience of our demonstration will be conformed by the database community, we have tagged some subgraphs of interest within the dataset. This includes a graph of papers with transitive reference relationships to the foundational paper of Edgar Codd on relational databases [5]. We call this the *Codd's world* graph, on which we aim to undertake studies in future work.

Demonstration: We will start introducing the audience to the core functionalities of Hermes by searching for authors or papers, as shown in Figure 3. Based on the results, we navigate through some different network representations of the data and explore dependencies between the results. This will enable the audience to use the tool by themselves.

In the next step, we explain some representative scenarios for scholarly network analysis: *Impact evaluation*, *academic recommendation*, or *expert identification*. This selection is based on a recent survey [22]. We then carry out a task from the outlined cases. For every alternative, we first accomplish the task and then walk the user through the series of queries that conform it.

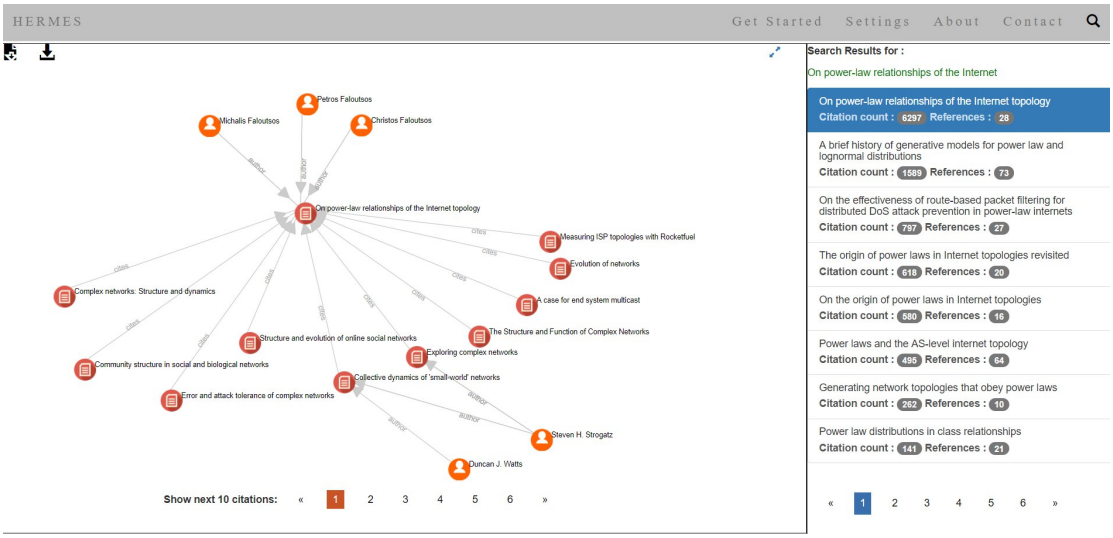

Figure 3: Exploring networks with Hermes.

Following this, we take the user to a Jupyter notebook, where we submit alternative versions (with different optimizations) of a single query. For the queries we observe different execution times in spite of them achieving the same task. This provides insights into the practical, positive impact of the optimizations.

As a takeaway, we expect the user to understand the different indexing and incremental processing techniques, which can be used to build efficient prototypes based on mainstream graph databases. We hope to encourage them to use our tool when searching for literature, due to its supportive features.

4 CONCLUSION

In this paper we introduce Hermes, a tool for exploring and analyzing large scholarly datasets building upon on a graph database and an integrated search engine. While in an early phase, it already comprises a range of functionalities that motivate us to share our work with the community. We seek to encourage others to use our tool, helping us to improve it and, through user studies, contributing to the understanding and characterization of SNA workloads. For future work, we will enhance Hermes with additional analysis options and automated query rewrites. Other target features include support for cross-dataset exploration and for connecting to more data sources (e.g., ORCIDs). Potentially, we can also extend our tool to the purpose of supporting systematic literature reviews and research exploration, addressing the lack of suitable tools in these research areas [11, 16].

ACKNOWLEDGMENTS

We would like to thank the reviewers for their insightful feedback. We thank Jingy Ma for helping us to efficiently load large graph data. This work is supported by DFG grants SA 465/50-1 and LE 3382/2-1, and the DAAD STIBET Matching Funds grant.

REFERENCES

[1] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. 2009. Gephi: an open source software for exploring and manipulating networks. *Icwsm* 8 (2009), 361–362.
[2] Vladimir Batagelj and Andrej Mrvar. 1998. Pajek-program for large network analysis. *Connections* 21, 2 (1998), 47–57.
[3] Peter Boncz. 2013. LDBC: Benchmarks for Graph and RDF Data Management. In *IDEAS*. ACM, 1–2.
[4] Chaomei Chen. 2006. CiteSpace II: Detecting and Visualizing Emerging Trends and Transient Patterns in Scientific Literature. *Journal of the Association for Information Science and Technology* 57, 3 (2006), 359–377.
[5] Edgar F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* 13, 6 (1970), 377–387.
[6] Gabor Csardi and Tamas Nepusz. 2006. The igraph software package for complex network research. *InterJournal, Complex Systems* 1695, 5 (2006), 1–9.
[7] Yuxiao Dong, Hao Ma, Zhihong Shen, and Kuansan Wang. 2017. A Century of Science: Globalization of Scientific Collaborations, Citations, and Innovations. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1437–1446.
[8] Gabriel Campero Durand. 2017. *Best Practices for Developing Graph Database Applications: A Case Study Using Apache Titan.* Master's thesis. University of Magdeburg.
[9] Gabriel Campero Durand, Marcus Pinnecke, David Broneske, and Gunter Saake. 2017. Backlogs and Interval Timestamps: Building Blocks for Supporting Temporal Queries in Graph Databases. In *EDBT/ICDT Workshops*.
[10] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX.* Technical Report. Los Alamos National Laboratory (LANL).
[11] Edgar Hassler, Jeffrey C. Carver, David Hale, and Ahmed Al-Zubidy. 2016. Identification of SLR Tool Needs - Results of a Community Workshop. *Information and Software Technology* 70 (2016), 122–129.
[12] Narayanan Meyyappan, Gobinda G. Chowdhury, and Schubert Foo. 2000. A Review of the Status of 20 Digital Libraries. *Journal of Information Science* 26, 5 (2000), 337–355.
[13] Youshan Miao, Wentao Han, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Enhong Chen, and Wenguang Chen. 2015. Immortalgraph: A System for Storage and Analysis of Temporal Graphs. *ACM Transactions on Storage* 11, 3 (2015), 14.
[14] Vera Zaychik Moffitt and Julia Stoyanovich. 2016. Towards a Distributed Infrastructure for Evolving Graph Analytics. In *WWW*. WWW Steering Committee, 843–848.
[15] Hassan Sayyadi and Lise Getoor. 2009. Futurerank: Ranking Scientific Articles by Predicting Their Future Pagerank. In *SIAM ICDM*. SIAM, 533–544.
[16] Ivonne Schröter, Jacob Krüger, Philipp Ludwig, Marcus Thiel, Andreas Nürnberger, and Thomas Leich. 2017. Identifying Innovative Documents: Quo Vadis?. In *ICEIS*. ScitePress, 653–658.
[17] Konstantinos Semertzidis and Evaggelia Pitoura. 2017. Historical Traversals in Native Graph Databases. In *ADBIS*. Springer, 167–181.
[18] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-june Paul Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. In *WWW*. ACM, 243–246.
[19] Gábor Szárnyas. 2017. Incremental View Maintenance for Property Graph Queries. *arXiv preprint arXiv:1712.04108* (2017).
[20] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD*. ACM, 990–998.
[21] Manuel Then, Timo Kersten, Stephan Günnemann, Alfons Kemper, and Thomas Neumann. 2017. Automatic Algorithm Transformation for Efficient Multi-Snapshot Analytics on Temporal Graphs. In *Proceedings of the VLDB Endowment*, Vol. 10. VLDB Endowment, 877–888.
[22] Feng Xia, Wei Wang, Teshome Megersa Bekele, and Huan Liu. 2017. Big Scholarly Data: A Survey. *IEEE Transactions on Big Data* 3, 1 (2017), 18–35.
[23] Erjia Yan and Ying Ding. 2014. Scholarly Networks Analysis. In *Encyclopedia of Social Network Analysis and Mining*. Springer, 1643–1651.
[24] An Zeng, Zhesi Shen, Jianlin Zhou, Jinshan Wu, Ying Fan, Yougui Wang, and H Eugene Stanley. 2017. The Science of Science: From the Perspective of Complex Systems. *Physics Reports* (2017).