The VLDB Journal
https://doi.org/10.1007/s00778-019-00548-x

SPECIAL ISSUE PAPER

The ubiquity of large graphs and surprising challenges of graph processing: extended survey

Siddhartha Sahu[1] · Amine Mhedhbi[1] · Semih Salihoglu[1] · Jimmy Lin[1] · M. Tamer Özsu[1]

Received: 21 January 2019 / Revised: 9 May 2019 / Accepted: 13 June 2019

Abstract
Graph processing is becoming increasingly prevalent across many application domains. In spite of this prevalence, there is little research about how graphs are actually used in practice. We performed an extensive study that consisted of an online survey of 89 users, a review of the mailing lists, source repositories, and white papers of a large suite of graph software products, and in-person interviews with 6 users and 2 developers of these products. Our online survey aimed at understanding: (i) the types of graphs users have; (ii) the graph computations users run; (iii) the types of graph software users use; and (iv) the major challenges users face when processing their graphs. We describe the participants' responses to our questions highlighting common patterns and challenges. Based on our interviews and survey of the rest of our sources, we were able to answer some new questions that were raised by participants' responses to our online survey and understand the specific applications that use graph data and software. Our study revealed surprising facts about graph processing in practice. In particular, real-world graphs represent a very diverse range of entities and are often very large, scalability and visualization are undeniably the most pressing challenges faced by participants, and data integration, recommendations, and fraud detection are very popular applications supported by existing graph software. We hope these findings can guide future research.

Keywords User survey · Graph processing · Graph databases · RDF systems

1 Introduction

Graph data representing connected entities and their relationships appear in many application domains, most naturally in social networks, the Web, the Semantic Web, road maps, communication networks, biology, and finance, just to name a few examples. There has been a noticeable increase in the prevalence of work on graph processing both in research and in practice, evidenced by the surge in the number of different commercial and research software for managing and processing graphs. Examples include graph database systems [13,20,26,49,65,73,90], RDF engines [52,96], linear algebra software [17,63], visualization software [25,29], query languages [41,72,78], and distributed graph processing systems [30,34,40]. In the academic literature, a large number of publications that study numerous topics related to graph processing regularly appear across a wide spectrum of research venues.

Despite their prevalence, there is little research on how graph data are actually used in practice and the major challenges facing users of graph data, both in industry and in research. In April 2017, we conducted an online survey across 89 users of 22 different software products, with the goal of answering 4 high-level questions:

(i) What types of graph data do users have?
(ii) What computations do users run on their graphs?
(iii) Which software do users use to perform their computations?

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00778-019-00548-x) contains supplementary material, which is available to authorized users.

✉ Siddhartha Sahu
s3sahu@uwaterloo.ca

Amine Mhedhbi
amine.mhedhbi@uwaterloo.ca

Semih Salihoglu
semih.salihoglu@uwaterloo.ca

Jimmy Lin
jimmylin@uwaterloo.ca

M. Tamer Özsu
tamer.ozsu@uwaterloo.ca

[1] University of Waterloo, Waterloo, Canada

Published online: 29 June 2019

(iv) What are the major challenges users face when processing their graph data?

Our major findings are as follows:

- *Variety* Graphs in practice represent a very wide variety of entities, many of which are not naturally thought of as vertices and edges. Most surprisingly, traditional enterprise data comprised of products, orders, and transactions, which are typically seen as the perfect fit for relational systems, appear to be a very common form of data represented in participants' graphs.
- *Ubiquity of very large graphs* Many graphs in practice are very large, often containing over a billion edges. These large graphs represent a very wide range of entities and belong to organizations at all scales from very small enterprises to very large ones. This refutes the sometimes heard assumption that large graphs are a problem for only a few large organizations such as Google, Facebook, and Twitter.
- *Challenge of scalability* Scalability is unequivocally the most pressing challenge faced by participants. The ability to process very large graphs efficiently seems to be the biggest limitation of existing software.
- *Visualization* Visualization is a very popular and central task in participants' graph processing pipelines. After scalability, participants indicated visualization as their second most pressing challenge, tied with challenges in graph query languages.
- *Prevalence of RDBMSes* Relational databases still play an important role in managing and processing graphs.

Our survey also highlights other interesting facts, such as the prevalence of machine learning on graph data, e.g., for clustering vertices, predicting links, and finding influential vertices.

We further reviewed user feedback in the mailing lists, bug reports, and feature requests in the source code repositories of 22 software products between January and September of 2017 with two goals: (i) to answer several new questions that the participants' responses raised and (ii) to identify more specific challenges in different classes of graph technologies than the ones we could identify in participants' responses. For some of the questions in our online survey, we also compared the graph data, computations, and software used by the participants with those studied in academic publications. For this, we reviewed 252 papers from 3 different year's proceedings of 7 conferences across different academic venues.

Different database technologies and research topics are often motivated with a small set of common applications, informally referred to as "killer" applications of the technology. For example, object-oriented database systems are associated with computer-aided design and manufacturing, and XML is associated with the Web. An often-asked question in the context of graphs is: What is the killer application of graph software products? The wide variety of graphs and industry fields mentioned by our online survey participants hinted that we cannot pinpoint a small set of such applications. To better understand the applications supported by graphs, we reviewed the white papers posted on the Web sites of 8 graph software products. We also interviewed 6 users and 2 developers of graph processing systems. Our reviews and interviews corroborated our findings that graphs have a very wide range of applications but also highlighted several common applications, primarily in data integration, recommendations, and fraud detection, as well as several new applications we had not identified in our online survey. Our interviews also give more details than our online survey about the actual graphs used by enterprises and how they are used in applications.

In addition to discussing the insights we gained through our study, we discuss several directions about the future of graph processing. We hope our study can inform research about real use cases and important problems in graph processing.

2 Methodology of online survey, mailing lists, source repositories, and academic publications

In this section, we first describe the format of our survey and then how we recruited the participants. Next we describe the demographic information of the participants, including the organizations they come from and their roles in their organizations. Then we describe our methodology of reviewing academic publications. Then we describe our methodology for reviewing the user feedback in the mailing lists, bug reports, and feature requests in the source code repositories of the software products. We end this section with a discussion of our methodology, which we believe other researchers can easily reproduce to study the uses of other technology, and some lessons we learned from our experience of performing a user study. We review our methodology of reviewing white papers and our interviews in Sects. 4.1 and 5.1, respectively.

2.1 Online survey format and participants

2.1.1 Format

The survey was in the format of an online form. All of the questions were optional, and participants could skip any number of questions. There were 2 types of questions:

(i) *Multiple-choice* There were 3 types of multiple-choice questions: (a) yes–no questions; (b) questions that

allowed only a single choice as a response; and (c) questions that allowed multiple choices as a response. The participants could use an *Other* option when their answers required further explanation or did not match any of the provided choices. We randomized the order of choices in questions about the computations participants run and the challenges they face.

(ii) *Short-answer* For these questions, the participants entered their responses in a text box.

There were 34 questions grouped into six categories: (i) demographic questions; (ii) graph datasets; (iii) graph and machine learning computations; (iv) graph software; (v) major challenges; and (vi) workload breakdown.

2.1.2 Participant recruitment

We prepared a list of 22 popular software products for processing graphs (see Table 1) that had public user mailing lists covering 6 types of technologies: graph database systems, RDF engines, distributed graph processing systems (DGPSes), graph libraries to run and compose graph algorithms, visualization software, and graph query languages.[1] Our goal was to be as comprehensive as possible in recruiting participants from the users of different graph technologies. However, we acknowledge that this list is incomplete and does not cover all of the graph software used in practice.

We conducted the survey in April 2017, and used 4 methods to recruit participants from the users of these 22 software products:

- *Mailing lists* We posted the survey to the user mailing lists of the software in our list.
- *Private emails* Five mailing lists, (i) Neo4j; (ii) OrientDB; (iii) ArangoDB; (iv) JanusGraph; and (v) NetworkX, allowed us to send private emails to the users. We sent private emails to 171 users who were active on these mailing lists between February and April of 2017.
- *Slack channels* Two of the software products on our list, Neo4j and Cayley, had Slack channels for their users. We posted the survey to these channels.
- *Twitter* A week after posting our survey to the mailing lists and Slack channels and sending private emails, we posted a tweet with a link to our survey to 7 of the 22 software products that had an official Twitter account. Only Neo4j retweeted our tweet.

Participants that we recruited through different methods shared the same online link and we could not tell the number of participants recruited from each method. In particular, we suspect that there were more users from graph database systems mainly because their mailing lists contained more active users, as given in Table 1. Moreover, 4 of the 5 mailing lists that allowed us to send private emails and the Slack and Twitter channels belonged to graph database systems. We note that after posting the survey on Twitter, we received 12 responses.

In the end, there were 89 participants. Below, we give an overview of the organizations these participants work in and the role of the participants in their organizations.

Field of organizations We asked the participants which field they work in. Participants could select multiple options. Table 2 shows the 12 choices and participants' responses. In the table, "*R*" and "*P*" indicate researchers and practitioners (defined momentarily), respectively. In addition to the given choices, using the *Other* option, participants indicated 5 other fields: education, energy market, games and entertainment, investigations and audits, and grassland man-

Table 1 Software products used for recruiting participants and the count of active users in their mailing list in February–April 2017

Technology	Software	#Users	
Graph database system	ArangoDB [13]	40	238
	Caley [20]	14	
	DGraph [26]	33	
	JanusGraph [49]	32	
	Neo4j [65]	69	
	OrientDB [73]	45	
	Sparksee [90]	5	
RDF engine	Apache Jena [52]	87	110
	Virtuoso [96]	23	
Distributed graph processing engine	Apache Flink (Gelly) [30]	24	39
	Apache Giraph [34]	8	
	Apache Spark (GraphX) [40]	7	
Query language	Gremlin [41]	82	82
Graph library	Graph for Scala [35]	4	97
	GraphStream [37]	8	
	Graphtool [38]	28	
	NetworKit [68]	10	
	NetworkX [69]	27	
	SNAP [84]	20	
Graph visualization	Cytoscape [25]	93	116
	Elasticsearch X-Pack Graph [29]	23	
Graph representation	Conceptual graphs [23]	6	6

Last column is the total count for each technology

[1] The linear algebra software we considered, e.g., BLAS [17] and MATLAB [63], either did not have a public mailing list or their lists were inactive.

Table 2 Participants' fields of work

Field	Total	*R*	*P*
Information and technology	48	12	36
Research in academia	31	31	0
Finance	12	2	10
Research in industry laboratory	11	11	0
Government	7	3	4
Health care	5	3	2
Defense and space	4	3	1
Pharmaceutical	3	0	3
Retail and e-commerce	3	0	3
Transportation	2	0	2
Telecommunications	1	1	0
Insurance	0	0	0
Other	5	2	3

agement. In total, participants indicated 17 different fields, demonstrating that graphs are being used in a wide variety of fields. Throughout the survey, we group the participants into 2 categories:

- *Researchers* are the 36 participants who indicated at least one of their fields as research in academia or research in an industry laboratory. Some of these participants further selected other choices as their fields, the most popular of which were information and technology, government, defense and space, and health care.
- *Practitioners* are the remaining 53 participants who did not select research in academia or an industry laboratory. The top two fields of practitioners were information and technology and finance, indicated by 36 and 10 people, respectively.

In the remainder of this paper, we will explicitly indicate when the responses of the researchers and practitioners to our survey questions differ significantly. In the absence of an explicit comparison, readers can assume that both groups' responses were similar.

Size of organizations Table 3 shows the sizes of the organizations that the participants work in, which ranged from very small organizations with less than 10 employees to very large ones with more than 10,000 employees.

Role at work We asked the participants their roles in their organizations and gave them the following 4 choices: (i) researcher; (ii) engineer; (iii) manager; and (iv) data analyst. Participants could select multiple options. The top 4 roles were engineers, selected by 54, researchers, selected by 48, data analysts, selected by 18, and managers, selected by 16. The other roles participants indicated were architect, devops, and student.

Table 3 Size of the participants' organizations

Size	Total	*R*	*P*
1–10	27	17	10
10–100	23	6	17
100–1000	14	4	10
1000–10,000	6	4	2
$> 10,000$	15	4	11

Table 4 Academic conferences and surveyed years

Conference	Years reviewed
VLDB	2014 [48], 2017 [18], 2018 [8]
KDD	2015 [54], 2017 [55], 2018 [56]
SOCC	2015 [85], 2017 [86], 2018 [87]
OSDI/SOSP	2016 [57], 2017 [88], 2018 [12]
ICML	2016 [15], 2017 [75], 2018 [28]
SC	2016 [80], 2017 [81], 2018 [82]

2.2 Review of academic publications

In order to compare the graph data, computations, and software academics work on with those that our participants indicated, we surveyed papers in the proceedings of 3 different years of the 7 academic conferences shown in Table 4.[2] Our goal in choosing these conferences was to select a variety of venues where papers on graph processing are published. Specifically, our list consists of venues in databases, data mining, machine learning, operating systems, high-performance computing, and cloud computing. For each paper in these proceedings, we first selected the ones that directly studied a graph computation or were developing graph processing software. We omitted papers that were not primarily focused on graph processing, even if they used a graph algorithm as a subroutine to solve a problem. For example, we omitted a paper studying a string matching algorithm that uses a graph algorithm as a subroutine. In the end, we selected 252 papers.

For each of the 252 papers, we identified: (i) the graph datasets used in experiments; (ii) the graph and machine learning computations that appeared in the paper; and (iii) the graph software used in the paper. In our survey, we asked users questions about which graph and machine learning computations they perform. The choices we provided in these questions came from the computations we identified in these

[2] For each conference, we initially surveyed one year selected between 2014 and 2016 and later extended the survey to include the years 2017 and 2018. Note that OSDI and SOSP are held in alternating years.

publications (see Sects. 3.2.1 and 3.2.2 and Appendices A and B for details).

2.3 Review of emails and code repositories

To answer some questions that participants' responses raised and to identify more specific challenges users face than the ones we identified from participants' responses, we reviewed emails in the mailing lists of the 22 software products between January and September of 2017. In addition, 20 of these 22 software products had open-source code repositories. We reviewed the bug reports and feature requests (*issues* henceforth) in these repositories between January and September of 2017. We also reviewed the repositories of 2 popular graph visualization tools: Gephi [33] and Graphviz [39]. For emails and issues before January 2017, we performed a targeted keyword search to find more instances of the challenges we identified in the January–September 2017 review.

In total, we reviewed over 6000 emails and issues. The overwhelming majority of the emails and issues were routine engineering tasks, such as users asking how to write a query or developers asking for integration with another software. The number of emails and issues that were useful for identifying challenges was 299 in total. We review these challenges in Sect. 3.4.2. Table 22 in "Appendix" shows the exact number of emails and issues we reviewed for each product and the number of commits in its code repository to give readers a sense of how active these repositories are.

2.4 Note on methodology

We end this section with two points about our methodology and a brief summary of the lessons we learned from performing a user study.

Biases Performing a survey study brings up challenging methodological questions, such as what is a principled way of recruiting participants, picking a list of choices for graph queries, or reviewing academic publications? Our guiding principle when addressing these questions was to be as broad as possible and to avoid ad hoc decisions. For example, the initial 22 graph software products we found were the products that had open-source mailing lists of a much longer list of all products we were aware of.

Similarly, when asking about the different graph queries and computations, instead of an ad-host list of choices, we gathered a list from academic publications. However, the choices we made inevitably introduced biases.

We acknowledge these biases when we present our findings in later sections. In particular, the numbers we report should not be interpreted statistically. Our goal was not to understand any statistical property, e.g., the average number of edges, about the graphs used in practice, or users of graphs. Despite these biases, we found overwhelming evidence for some of our observations, which we believe give insights into how graphs are used in practice.

Abundance of Public Information Having direct access to actual users from industry is a known challenge for researchers in academia. There is, however, an abundance of public information in mailing lists, forums, vendor Web sites, open-source code repositories, social media, question-and-answer Web sites, and elsewhere, which can be used to survey actual users and arrange in-person interviews. In this paper, we essentially reviewed this public information and used it to contact actual users. Our work is not the first but the most extensive that we know of in terms of the public sources it reviews. We believe our methodology can easily be repeated by researchers in academia to study how other types of data or technology is used in practice.

Lessons from the survey methodology We highlight three lessons from our experience of applying our methodology.

- *Lesson 1* Many users are willing to share information. Especially for our online survey, we did not expect to recruit 89 participants prior to sending the survey out.
- *Lesson 2* Avoid making assumptions about participants' answers in the survey. For instance, as we discuss in Sect. 3.1.2, we assumed few users would have edge graphs with more than 1 billion edges, so capped the choices of a question about graph sizes at 1 billion. This resulted in losing important information about how much larger the graphs are beyond 1 B.
- *Lesson 3* Users have different and often non-technical languages than researchers to explain their technology. For instance, our interview with a biologist using an RDF engine heavily involved terms such as tissues, processes, angiogenesis, molecules, and chemical reactions. This required spending considerable time during the interview on terminology, which put a hurdle on focusing on technical topics on graph processing. We learned to thoroughly study our interviewee's products and application domains prior to an interview.

3 Online survey

In this section, we describe the questions we asked in the survey and report the responses of the participants. We also report the results of our review of academic publications. Throughout the section, we highlight the results that we found particularly interesting or surprising.

3.1 Graph datasets

In this section, we describe the properties of the graph datasets that the participants work with.

Table 5 Real-world entities represented by the participants' graphs and studied in publications

Category	Human	RDF	Scientific	Non-human	NH-P	NH-B	NH-W	NH-G	NH-D	NH-I	NH-K
Total	45	23	15	60	13	11	4	7	5	9	11
R	18	11	9	22	1	6	2	4	1	7	6
P	27	12	6	38	12	5	2	3	4	2	5
A	165	20	45	169	7	28	77	33	0	17	11

Legend for non-human entities: *products* (NH-P), *business and financial data* (NH-B), *Web data* (NH-W), *geographic maps* (NH-G), *digital data* (NH-D), *infrastructure networks* (NH-I), *knowledge and textual data* (NH-K)

3.1.1 Real-world entities represented

We asked the participants about the real-world entities that their graphs represent. We provided them with 4 choices and the participants could select multiple of them.

(i) *Humans*: e.g., employees, customers, and their interactions.
(ii) *Non-human entities*: e.g., products, transactions, or Web pages.
(iii) *RDF or Semantic Web*.
(iv) *Scientific*: e.g., chemical molecules or biological proteins.

For the participants who selected non-human entities, we followed up with a short-answer question asking them to describe what these are. Participants indicated 52 different kinds of non-human entities, which we group into 7 broad categories.[3] We indicate the acronyms we use in our tables for each category in parentheses:

(i) *Products* (NH-P): e.g., products, orders, and transactions.
(ii) *Business and Financial Data* (NH-B): e.g., business assets, funds, or bitcoin transfers.
(iii) *World Wide Web Data* (NH-W).
(iv) *Geographic Maps* (NH-G): e.g., roads, bicycle sharing stations, or scenic spots.
(v) *Digital Data* (NH-D): e.g., files and folders or videos and captions.
(vi) *Infrastructure Networks* (NH-I): e.g., oil wells and pipes or wireless sensor networks.
(vii) *Knowledge and Textual Data* (NH-K): e.g., keywords, lexicon terms, words, and definitions.

Table 5 shows the responses. In the table, the number of academic publications that use each type of graph is listed in the *A* row. We highlight two interesting observations:

- *Variety* Real graphs capture a very wide variety of entities. Readers may be familiar with entities such as social connections, infrastructure networks, and geographic maps. However, many other entities in the participants' graphs may be less natural to think of as graphs. These include malware samples and their relationships, videos and captions, or scenic spots, among others. This lends credence to the cliché that graphs are everywhere.
- *Product graphs* Products, orders, and transactions were the most popular non-human entities represented in practitioners' graphs, indicated by 12 practitioners. This contrasts with their relative unpopularity among researchers and academics. Only 1 researcher used product graphs, and after digital data graphs, product graphs were the second least popular graphs used in academic papers. Such product–order–transaction data are traditionally the classic example of enterprise data that perfectly fits the relational data model. It is interesting that enterprises represent similar product data as graphs, possibly because they find value in analyzing connections in such data.

We also note that we expected scientific graphs to be used mainly by researchers. Surprisingly, scientific graphs are prevalent among practitioners as well.

3.1.2 Size

We asked the participants the number of vertices, number of edges, and total uncompressed size of their graphs. They could select multiple options. Table 6a–c shows the responses. As shown in the tables, graphs of every size, from very small ones with less than 10 K edges to very large ones with more than 1 B edges, are prevalent across both researchers and practitioners. We make one interesting observation:

- *The ubiquity of very large graphs* A significant number of participants work with very large graphs. Specifically, 20 participants (8 researchers and 12 practitioners) indicated using graphs with more than a billion edges. Moreover, the 20 participants with graphs with more than one billion edges are from organizations with different scales, rang-

[3] Six entities that the participants mentioned did not fall under any of our 7 categories, which we list for completeness: call records, computers, cars, houses, time slots, and specialties.

Table 6 Sizes of the participants' graphs

Vertices	Total	*R*	*P*
(a) *Number of vertices*			
< 10 K	22	11	11
10–100 K	22	9	13
100 K–1 M	19	7	12
1–10 M	17	6	11
10–100 M	20	10	10
> 100 M	27	10	17
Edges	Total	*R*	*P*
(b) *Number of edges*			
< 10 K	23	11	12
10–100 K	22	9	13
100 K–1 M	13	3	10
1–10 M	9	5	4
10–100 M	21	8	13
100 M–1 B	21	8	13
> 1 B	20	8	12
Size	Total	*R*	*P*
(c) *Total uncompressed bytes*			
< 100 MB	23	12	11
100 MB–1 GB	19	9	10
1–10 GB	25	9	16
10–100 GB	17	5	12
100 GB–1 TB	20	8	12
> 1 TB	17	5	12

Table 7 Sizes of organization using graphs with > 1B edges

Size	1–10	10–100	100–1000	> 10, 000
#	4	4	7	4

ing from very small to very large, as shown in Table 7. This refutes the common assumption that only very large organizations—such as Google [62], Facebook [21], and Twitter [83] that have Web and social network data—have very large graphs. Finally, we note that these large graphs represent a variety of entities, including social, scientific, RDF, product, and digital data,[4] indicating that very large graphs appear in a wide range of domains.

[4] Some participants selected multiple graph sizes and multiple entities, so we cannot perform a direct match of which graph size corresponds to which entity. The entities we list here are taken from the participants who selected a single graph size and entity, so we can directly match the size of the graph to the entity.

Table 8 Topology and stored data types of the participants' graphs

Topology	Total	*R*	*P*
(a) *Directed versus undirected*			
Only directed	63	23	40
Only undirected	11	6	5
Both	15	7	8
Topology	Total	*R*	*P*
(b) *Simple versus multigraphs*			
Only simple graphs	26	9	17
Only multigraphs	50	20	30
Both	13	7	6

Type	Vertices			Edges		
	Total	*R*	*P*	Total	*R*	*P*
(c) *Data types stored on vertices and edges*						
String	79	31	48	66	24	42
Numeric	63	23	40	59	23	36
Date/time stamp	56	19	37	49	18	31
Binary	15	8	7	8	4	4

One thing that is not clear from our survey is how much larger the participants' graphs are beyond the maximum limits we inquired about (100 million vertices, 1 billion edges, and 1 TB uncompressed data). In order to answer this question, we categorized the graph sizes mentioned in the user emails we reviewed that were beyond these sizes. Focusing on the number of edges, we found 42 users with 1–10 B-edge graphs, 17 with 10–100 B-edge graphs, and 7 users processing graphs over 100 B edges. Two participants also clarified through an email exchange that their graphs contained 4 B and 30 B edges. As in our survey results, these large graphs represented a wide range of entities, such as product–order–transaction data, or entities from agriculture and finance. Table 20 in "Appendix" shows the exact distribution of sizes we identified. As we discuss in Sect. 5, several of the applications described in our applications also contained graphs in the 10–100 B-edge and over 100 B-edge scale.

3.1.3 Other questions on graph datasets

Topology We asked the participants whether their graphs were: (i) *directed or undirected* and (ii) *simple graphs or multigraphs*. We clarified that multigraphs are those with possibly multiple edges between two vertices, while simple graphs do not allow multiple edges between two vertices. Table 8a, b shows the responses.

Types of data stored on vertices and edges We asked the participants whether they stored data on the vertices and edges of their graphs. All participants except 3 indicated that they

Table 9 Frequency of changes

Frequency	Total	*R*	*P*
Static	40	21	19
Dynamic	55	22	33
Streaming	18	9	9

do. We asked the types of data they store and gave them 4 choices: (i) string; (ii) numeric; (iii) date or time stamp; and (iv) binary. Table 8c shows participants' responses. Five participants also indicated storing JSON, lists, and geographic coordinates using the *Other* option.

Dynamism We asked the participants how frequently the vertices and edges of their graphs change, i.e., are added, deleted, or updated. We provided 3 choices with the following explanations: (i) *static*: there are no or very infrequent changes; (ii) *dynamic*: there are frequent changes, and all changes are stored permanently; and (iii) *streaming*: there are very frequent changes and the participants' software discards some of the graph after some time. Table 9 shows the responses. Surprisingly 18 participants (9 researchers and 9 practitioners) indicated having streaming graphs.

3.2 Computations

In this section, we describe the computations that the participants perform on their graphs.

3.2.1 Graph computations

Our goal in this question was to understand what types of graph queries and computations, not including machine learning computations, participants perform on their graphs. We asked a multiple-choice question that contained as choices a list of queries and computations followed by a short-answer question that asked for computations that may not have appeared in the first question as a choice. In the multiple-choice question, instead of asking for a set of ad hoc queries and computations, we selected a list of graph queries and computations that appeared in the publications of the 6 conferences we reviewed (recall Sect. 2.2), using our best judgment to categorize similar computations under the same name. We describe our detailed methodology in "Appendix A."

Table 10 shows the 13 choices we provided in the multiple-choice question, the responses we got, and the number of academic publications that use or study each computation. As shown in the table, all of the 13 computations are used by both researchers and practitioners. Except for two computations, the popularity of these computations is similar among participants' responses and academic publications. The exceptions are neighborhood and reachability queries, which are, respectively, used by 51 and 27 participants, but studied, respectively, in 10 and 8 publications. Finding connected components appears to be a very popular and fundamental graph computation—it is the most popular graph computation overall and also among practitioners. We suspect it is a common preprocessing or cleaning step, e.g., to remove singleton vertices, across many tasks.

A total of 13 participants answered our follow-up short-answer question on other graph queries and computations they run. Example answers include queries to create schemas and graphs, custom bioinformatic algorithms, and finding k-cores in a weighted graph.

3.2.2 Machine learning computations

We next asked participants what kind of machine learning computations they perform on their graphs. Similar to the previous question, these questions were formulated to iden-

Table 10 Graph computations performed by the participants and studied in publications

Computation	Total	*R*	*P*	*A*
Finding connected components	55	18	37	31
Neighborhood queries (e.g., finding 2-degree neighbors of a vertex)	51	19	32	9
Finding short/shortest paths	43	18	25	28
Subgraph matching (e.g., finding all diamond patterns, SPARQL)	33	14	19	52
Ranking and centrality scores (e.g., PageRank, Betweenness Centrality)	32	17	15	45
Aggregations (e.g., counting the number of triangles)	30	10	20	24
Reachability queries (e.g., checking if u is reachable from v)	27	7	20	8
Graph partitioning	25	13	12	12
Node similarity (e.g., SimRank)	18	7	11	11
Finding frequent or densest subgraphs	11	7	4	4
Computing minimum spanning tree	9	5	4	4
Graph coloring	7	3	4	8
Diameter estimation	5	2	3	2

Table 11 Machine learning computations and problems performed by the participants and studied in publications

Computation	Total	*R*	*P*	*A*
(a) *Machine learning computations*				
Clustering	42	22	20	22
Classification	28	10	18	34
Regression (linear/logistic)	11	5	6	2
Graphical model inference	10	5	5	5
Collaborative filtering	9	4	5	5
Stochastic gradient descent	4	2	2	9
Alternating least squares	0	0	0	1
(b) *Problems solved by machine learning algorithms*				
Community detection	31	15	16	15
Recommendation system	26	10	16	5
Link prediction	25	10	15	11
Influence maximization	14	5	9	6

tify the machine learning computations that appeared in the academic publications we reviewed. We describe our detailed methodology in "Appendix B." We asked the following 2 questions:

- *Which machine learning computations do you run on your graphs?* The choices were: clustering, classification, regression (linear or logistic), graphical model inference, collaborative filtering, stochastic gradient descent, and alternating least squares.
- *Which problems that are commonly solved with machine learning do you solve using graphs?* The choices were: community detection, recommendation system, link prediction, and influence maximization.[5]

Table 11a and b shows the responses and the number of academic publications that use or study each computation. It is clear that machine learning is used very widely in graph processing. Specifically, 61 participants indicated that they either perform a machine learning computation or solve a problem using machine learning on their graphs. Clustering is the most popular computation performed, while community detection is the most popular problem solved using machine learning. None of the participants selected alternating least squares as a computation they perform.

[5] In the publications, link prediction referred to problems that predict a missing edge in a graph or data on an existing edge. Influence maximization referred to finding influential vertices in a graph, e.g., those that can bring more vertices to the graph. We did not provide detailed explanations about the problems to the participants.

Table 12 Graph traversals performed by the participants

Traversal	Total	*R*	*P*
Breadth-first search or variant	19	5	14
Depth-first search or variant	12	4	8
Both	22	8	14
Neither	20	11	9

3.2.3 Other questions on computations

Streaming Computations: We asked the participants if they performed incremental or streaming computations on their graphs: 32 participants (16 researchers and 16 practitioners) indicated that they do. We followed up with a question asking them to describe the incremental or streaming computations that they perform. A total of 4 participants indicated computing graph or vertex-level statistics and aggregations. A total of 3 participants indicated incremental or streaming computation of the following algorithms: approximate connected components, k-core, and hill climbing. For completeness, we list the other computations participants mentioned: computing node or community properties, calculating approximate answers to simple queries, incremental materialization, incremental enhancement of the knowledge graph, and scheduling.

We note that the 22 software products in Table 1 have limited or no support for incremental and streaming computations. We further note that we did not find any user in our further reviews of other sources or interviews that performed continuous computation on a very dynamic stream of edges or nodes.

Traversals We asked the participants which fundamental traversals, breadth-first search or depth-first search, they use in their algorithms. Table 12 shows the responses. Participants commonly use both kinds of traversals.

3.3 Graph software

We next review the properties of the different graph software that the participants use.

3.3.1 Software types

Software for Querying and Performing Computations We asked the participants which types of graph software they use to query and perform computations on their graphs. The choices included 5 types of software from Table 1 as well as distributed data processing systems (DDPSes), such as Apache Hadoop and Spark, relational database management systems (RDBMSes), and linear algebra libraries and software, such as BLAS and MATLAB. Table 13 shows the exact

Table 13 Software for graph queries and computations

Software	Total	*R*	*P*	*A*
Graph database system (e.g., Neo4j, OrientDB, TitanDB)	59	20	39	6
Apache Hadoop, Spark, Pig, Hive	29	11	18	10
Apache Tinkerpop (Gremlin)	23	9	14	1
Relational database management system (e.g., MySQL, PostgreSQL)	21	6	15	7
RDF engine (e.g., Jena, Virtuoso)	16	8	8	12
Distributed graph processing systems (e.g., Giraph, GraphX)	14	8	6	36
Linear algebra library/software (e.g., MATLAB, Maple, BLAS)	8	6	2	6
In-memory graph processing library (e.g., SNAP, GraphStream)	7	5	2	4

choices and responses: 84 participants answered this question and each selected 2 or more types of software. We highlight 3 interesting observations:

- *Popularity of Graph Database Systems*: The most popular choice was graph database systems. We suspect this is partly due to their increasing popularity and partly due to the inherent bias in the participants we recruited—as explained in Sect. 2.1.2, more of them came from users of graph database systems. We did not ask the participants which specific graph database system they used.
- *Popularity of RDBMSes*: 21 participants (6 researchers and 15 practitioners) chose RDBMSes. We consider this number high given that we did not recruit participants from the mailing lists of any RDBMS. Interestingly, 16 of these 20 participants also indicated using graph database systems. From our survey, we cannot answer what the participants used RDBMSes for. It is possible that they use an RDBMS as the main transactional storage and a graph database system for graph-specific tasks such as traversals. This was the case in the applications described to us in our interviews (see Sect. 5).
- *Unpopularity of DGPSes*: Only 6 practitioners indicated using a DGPS, such as Giraph, GraphX, and Gelly. This contrasts with DGPSes' popularity among academics, where they are the most popular systems, studied by 36 publications. One can consider graph database systems as RDBMSes that are specialized for graphs and DGPSes as DDPSes that are specialized for graphs. In light of this analogy, we note that there is an opposite trend in the usage of these groups of systems. While more participants indicated using graph database systems than RDBMSes, significantly more participants indicated using DDPSes than DGPSes.

Software for non-querying tasks We asked the participants which types of graph software, possibly an in-house one, they use for tasks other than querying graphs. Table 14 shows the choices and the responses. We highlight one interesting observation:

Table 14 Software used for non-querying tasks

Software	Total	*R*	*P*	*A*
Graph visualization	55	22	33	15
Build/extract/transform	14	8	6	0
Graph cleaning	5	1	4	2
Synthetic graph generator	4	3	1	49
Specialized debugger	2	0	2	0

Table 15 Architectures of the software used by participants

Architecture	Total	*R*	*P*
Single machine serial	31	17	14
Single machine parallel	35	21	14
Distributed	45	17	28

- *Importance of Visualization:* Visualization software is, by a large margin, the most popular type of software participants use among the 5 choices. This clearly shows that graph visualization is a very common and important task. As we discuss in Sect. 3.4, participants also indicated visualization as one of their most important challenges when processing graphs.

3.3.2 Other questions on software

Software Architectures: We asked the participants the architectures of the software products they use for processing graphs. The choices were single machine serial, single machine parallel, and distributed. Table 15 shows the responses. Distributed products were the most popular choice and users' selections highly correlated with the size of graphs they have. For example, 29 of the 45 participants that selected distributed architecture had graphs over 100 M edges.

Data Storage in Multiple Formats: We asked the participants whether or not they store a single graph in multiple formats: 33 participants answered yes and the most popular multiple-

Table 16 Graph processing challenges faced by participants

Challenge	Total	*R*	*P*
Scalability (i.e., software that can process larger graphs)	45	20	25
Visualization	39	17	22
Query languages/programming APIs	39	18	21
Faster graph or machine learning algorithms	35	19	16
Usability (i.e., easier to configure and use)	25	10	15
Benchmarks	22	12	10
More general-purpose graph software (e.g., that can process off-line, online, and streaming computations)	20	9	11
Graph cleaning	17	7	10
Debugging and testing	10	2	8

format combination was a relational database format and a graph database format. "Appendix C" provides the detailed responses.

3.4 Practical challenges

In this section, we first discuss the challenges in graph processing that the participants identified, followed by a discussion of the challenges that we identified through our review of user emails and code repositories of different types of graph technologies.

3.4.1 Challenges identified from survey

We asked the participants 2 questions about the challenges they face when processing their graphs. First, we asked them to indicate their top 3 challenges out of 10 choices we provided. Table 16 shows the choices and the participants' responses. Second, we asked them to state their biggest challenge in a short-answer question. Three major challenges stand out unequivocally from the responses:

- *Scalability*: The ability to process large graphs is the most pressing challenge participants face. Scalability was the most popular choice in the first question for both researchers and practitioners. Moreover, it was the most popular answer in the second question where 13 participants reiterated that scalability is their biggest challenge. The specific scalability challenges that the participants mentioned include inefficiencies in loading, updating, and performing computations, such as traversals, on large graphs.
- *Visualization*: Perhaps more surprisingly, graph visualization emerges as one of the top 3 graph processing challenges, as indicated by 39 participants in the first question and 1 participant in the short-answer question. This is consistent with the participants indicating visualization as the most popular non-query task they perform on their graphs, as discussed in Sect. 3.3.1.
- *Query Languages and APIs*: Query languages and APIs present another common graph processing challenge, as indicated by 39 participants in the first question and 5 participants in the short-answer question. The specific challenges mentioned in the short-answer responses include expressibility of query languages, compliance with standards, and integration of APIs with existing systems. For instance, one participant found current graph query languages to have poor support for debugging queries and another participant indicated their difficulty in finding software that complies fully with SPARQL standards.

3.4.2 Challenges identified from review

To go beyond the survey and to understand more specific challenges users face or new functionalities users want, we studied the user emails and code repositories of different classes of software. Below, we categorize the challenges we found on visualization in graph database systems, RDF engines, DGPSes, and graph libraries, separately under *Visualization*. We also list the challenges we found in graph database systems and RDF engines related to query languages separately under *Query Languages*. The exact counts of emails and issues we found for each challenge is in Table 21 in "Appendix."

Graph database systems and RDF engines

- *High-Degree Vertices*: Users want the ability to process very high-degree vertices in a special way. One common request is to skip finding paths that go over such vertices either for efficient querying or because users do not find such paths interesting.
- *Hyperedges*: Hyperedges are edges between more than 2 vertices, e.g., a family relationship between three individuals. In graph database systems and RDF engines, there is no native way to represent hyperedges. The user discussions include suggestions to simulate hyperedges, such as having a "hyperedge vertex" and linking the vertices in the hyperedge to this mock vertex.
- *Versioning and Historical Analysis*: Users want the ability to store the history of the changes made to the vertices and edges and query over the different versions of the graph. These requests are made in systems that do not support versioning and the discussions are on how to add versioning support at the application layer.
- *Schema and Constraints*: Users want the ability to define schemas over their graphs, analogous to DTD and XSD

schemas for XML data [27], usually as a means to define constraints over their data. Examples include enforcing that the graph is acyclic or that some vertices always have a certain property.
- *Triggers*: Users ask for trigger-like capabilities in their graph database systems. Examples include automatically adding a particular property to vertices during insertion or creating a backup of a vertex or an edge in the filesystem during updates. We note that some systems do support limited trigger functionality, such as OrientDB's *hooks* or Neo4j's *TransactionEventHandler* API.

Graph visualization software:

- *Customizability:* One common challenge is to have the ability to customize the layout and design of the rendered graph, such as the shape or color of the vertices and edges.
- *Layout*: Another common challenge is drawing graphs with certain structures on the screen according to a specific layout users had in mind. The most common example is drawing *hierarchical* graphs, i.e., those in which some vertices are drawn on top of other vertices in an organizational hierarchy. Other examples include the drawing of star graphs, planar graphs, or a specialized tree layout, such as a *phylogenetic tree* [60].
- *Dynamic Graph Visualization:* Several users want support for or have challenges in animating the additions, deletions, and updates in a dynamic graph that is changing over time.

Users also have challenges in rendering large graphs with thousands or even millions of vertices and edges.

Query languages One of the most popular discussions in user emails of graph database systems and RDF engines was writing different queries in the query language of the software. In almost every case, there was a way of satisfying the users' needs. Below we list 2 such types of queries that could be interesting to researchers.

- *Subqueries*: Many users have challenges in the expression or performance of subqueries, i.e., using a query as part of another query. The challenges vary across different systems. Some users want the ability to embed SQL as a subquery in SPARQL. Other users want the results of a subquery to be a graph that can further be queried,[6] or to use a subquery as a predicate in another query.
- *Querying across Multiple Graphs*: A common request in graph database systems and RDF engines is to construct queries that span multiple graphs, such as using the results of a traversal in one graph to start traversals in another. This is analogous to querying over multiple tables by joins in RDBMSes.[7]

Profiling and debugging slow queries and using indices correctly to speed up queries are other common topics among users.

DGPSes and graph libraries

- *Off-the-Shelf Algorithms:* The most common request we found in DGPSes and graph libraries is the addition of a new algorithm that users could use off the shelf. All of these software products provide lower-level programming APIs using which users can compose graph algorithms. A small number of users want enhancements to these APIs as well. From our review, it appears that users of these software products find more value in directly using an already implemented algorithm than implementing the algorithms themselves.
- *Graph Generators:* All of the DGPSes and graph libraries in our list have modules to generate synthetic graphs. Our review revealed that users find these graph generators useful, e.g., for testing algorithms. A common request was the ability to generate different kinds of synthetic graphs, such as k-regular graphs or random directed power-law graphs.
- *GPU Support:* Several users, both in DGPSes and in graph libraries, want support for running graph algorithms on GPUs.

In every DGPS we reviewed, a common challenge is users' computations running out of cluster memory or having problems when using disk. We also note that except for Gelly, every DGPS and every graph library either have a visualization component or users have requests to add one, showing the importance of visualization across users of a range of different graph technologies.

3.5 Workload breakdown

We asked the participants how many hours per week they spend on 6 graph processing tasks and provided them with 3 choices: (i) less than 5 hours; (ii) 5 to 10 hours; and (iii) more than 10 hours. Table 17 shows the choices and ranks the tasks in terms of the number of participants that selected more than 10 hours first, then 5 to 10 hours, and then less than 5 hours. According to this ranking, the participants spend the most time in analytics and testing and the least time on ETL and cleaning.

[6] This feature is called *composition* and is supported in SPARQL but not in the languages of some graph database systems.

[7] This functionality is supported in RDF engines but not supported in some graph database systems.

Table 17 Time spent by the participants on different tasks

Task	0–5 h	5–10 h	> 10 h
Analytics	30	18	23
Testing	40	12	20
Debugging	37	18	15
Maintenance	46	14	13
ETL	44	14	10
Cleaning	52	10	6

4 Applications from white papers

4.1 Methodology

In order to understand the popular application areas and fields using graph software, we surveyed the white papers of software vendors. White papers are documents that software vendors provide, often for marketing purposes, to give information about the use cases of their products. In our case, we consider white papers to be any document found on a software vendor's official Web site categorized as a white paper, a use case, a case study, or a scenario. From the initial software products in Table 1, only four graph database systems, specifically ArangoDB, Neo4j, OrientDB and Sparksee, had white papers. To extend our review, we add the white papers of four RDF engines that were not in our initial list: AllegroGraph [6], AnzoGraph [11], GraphDB by Ontotext [70], and Stardog [91]. We note that we only found white papers for graph database systems and RDF engines.

For each white paper, we selected the ones that describe an application using the product, e.g., music recommendation or money laundering detection. We omitted white papers that did not describe a specific application. For example, we omitted white papers specific to the software architecture of a product. In the end, we reviewed 89 white papers.

4.2 Applications

We labeled each white paper with a high-level application category and the field of industry of the customer in the case study. Table 18 shows the different applications, the fields of industry in which the application was covered, and the number of white papers from graph databases and RDF systems that discussed the project. We found a total of 12 applications described in the white papers of graph databases and 5 applications in the white papers of RDF systems. As seen in the table, there is an overlap of the applications across both types of systems.

The three most popular applications were as follows:

- *Data Integration*: 44 white papers discussed primarily some data integration task that constructed a central, highly heterogenous graph from multiple sources. Data integration was also referred by some white papers as *master data management* or *knowledge graph creation*. This category does not contain the white papers that described primarily another main business application but performed data integration as an initial step.
  Data integration white papers briefly mentioned a variety of other applications that would be supported by the integrated data, such as enterprise search. Many of these 44 white papers, as well as many white papers that discussed a data integration initial step, emphasized that customers found data integration easier in the semi-structured graph models than structured relational tables.
- *Personalization and Recommendations*: The second most popular application was the use of a graph-based application data to personalize user interactions and provide better recommendations for the customers of a business. For example, one white paper described an e-commerce Web site that created a graph representation of the behavior of online shoppers and the interactions between customers and products to help make new product recommendations [14]. Another example was a personalized course curriculum service based on a hierarchical course topic relationships, represented as a graph, and the individual progress of each student [2].
  Many white papers avoided technical terms, but the applications described seemed to read the neighborhoods of users, represented as nodes in the underlying graph, to retrieve useful signals to make a recommendation.
- *Fraud and Threat Detection*: The third most popular application was the detection of fraud and threats in various businesses. For example, one white paper described the use of graphs to detect financial fraud in banks by looking for rings in the graph formed after linking bank accounts, personal details, and financial transactions [66]. Another application detected and prevented cyber crimes by monitoring for anomalous patterns in network traffic [1] represented as a graph.
  As a key benefit of using graphs, white papers highlighted that, compared to their equivalent SQL formulations, fraud patterns were easier to express as subgraph queries. Several white papers also mentioned that relational systems were not efficient enough to support these queries.

5 Applications from interviews

5.1 Methodology

White papers give an overview of the important applications using different software but often contain very high-level and non-technical marketing language. To understand some of the applications using graphs in more depth, we invited the

Table 18 Application areas and example uses of graphs in various fields described in graph software white papers

Application	Example	Fields	GDB	RDF	Total
Data integration	Building an ontology by integrating multiple heterogeneous biomedical data sources	Aerospace, art and culture and heritage, education, entertainment, finance, food and cooking, government, health and life sciences, intelligence and law enforcement, IT, journalism, marketing, retail, social media, toys and figurines	23	21	44
Personalization and recommendation	Recommending products on an e-commerce platform	Entertainment, finance, health and life sciences, hospitality and travel, IT, manufacturing, marketing, media, music, retail, social media, telecommunication	19	5	24
Fraud and threat detection	Detect cybercrime by searching for anomalous patterns	Finance, government, insurance, media, retail	9	1	10
Risk analysis and compliance	Risk reporting by banks to comply with government regulations	Finance, health and life sciences, IT, supply chain and logistics	2	3	5
Identity and access management	Monitor direct and indirect owners of businesses for financial analysis	Insurance, IT, telecommunication	4	0	4
Infrastructure management and monitoring	Manage cascading failures by tracking server interdependencies	Intelligence and law enforcement, IT	3	0	3
Delivery and logistics	Routing and tracking delivery parcels	Retail, supply chain and logistics	2	0	2
Social network analysis	Find the most viral users with maximum reach to other users	Social media	2	0	2
Other applications	Natural language question answering, call graph analysis, code analysis, drug discovery, traffic route recommendation	IT, telecommunication, traffic management	3	2	5

participants of our online survey for an in-person interview. Thirty-three participants had provided us with their email addresses and 4 of them agreed. To extend our interviews, we reached out to several of our contacts in major software companies and graph vendors. We did 4 additional in-person interviews: 2 developers and 2 users of graph processing software in major enterprises.

The occupations of our interviewees were as follows:

- Two IT consultants to several large enterprises on graph technologies.
- A developer of graph processing systems at Alibaba.
- A developer of graph processing systems at Siemens.
- A principal scientist at Amazon working on knowledge graphs.
- Engineers from a contact management company called FullContact [32], an electric utility company called State Grid [92], and a start-up called OpenBEL [71] that develops data publishing tools for biologists.

We lead the interviews with an open-ended question asking the interviewee to walk us through a concrete business application that uses graph data. The developers explained the applications of their customers. We asked questions about the details of the graph data, the graph computations they run, and the graph processing software they use in their applications. In addition, we asked three extra questions to each interviewee: (i) Where do you use graph visualization? (ii) Do you do streaming computations on your graphs? (iii) Do you have machine learning computations that use your graphs?

5.2 Overall observations

We make four observations:

- None of the applications that used graphs representing transactional business data used a graph database or an RDF store as the main system of record. In each case, a relational system was the main system of record and the transactional data were replicated to a graph software for the application to use. This gives a sense of where graph databases and RDF systems are in the IT ecosystem of our interviewees' enterprises. The developer from Alibaba

mentioned data replication as an important challenge for their internal customers.
- Interviewees mentioned visualizing graphs in data exploration, debugging, query formulation, and as a presentation tool within the enterprise, for example to show a manager the benefits of modeling an application data as a graph.
- Our interviewees were not aware of any continuous streaming computation performed on their graphs. Several interviewees mentioned processing highly dynamic graphs and buffering a window of several hours or days of these graphs. However, the computations in those applications were batch computations. For example, in one case, 3 days of business data would be copied over into a graph software to search for subgraph patterns.
- The machine learning applications that the interviewees discussed used graphs to extract features about nodes in a graph that were representing a business entity, such as a product or a customer. The feature extractions involved aggregating data from several hop neighborhoods of nodes and in one case through a recursive path query. These features would be used in vector representations of nodes and used by a machine learning application to do a prediction. We describe such a use case in Sect. 5.3.2.

We next describe some of the applications from our interviews in detail. We discuss several other applications in "Appendix E." Overall there were similarities between the applications described by our interviewees and those from the white papers, but we also discovered some new applications. We cover one such new application called *contingency analysis* in Sect. 5.6. Some interviewees modified or omitted detailed information about their applications, datasets, queries, software, or challenges. For example, they modified what the vertices and edges actually are or gave the approximate sizes of their graphs. We report the applications as described by the interviewees.

5.3 Recommendations

5.3.1 Keyword recommendations on Alibaba's E-commerce Web site

When customers enter keywords to the search text box on Alibaba's e-commerce Web site, several keywords that are related to the search are recommended. These recommended keywords often aim to increase the diversity of products the user sees on the site. Internally, these recommendations are made by an application that uses a very large knowledge graph and performs parallel traversals that find and rank paths in this knowledge graph. There are two interesting aspects of this application:

(i) Strict latency: The recommendation needs to be done in several milliseconds. None of the other applications we encountered during our interviews required such strict latencies for the computations they had to perform.
(ii) Size and generation of the graph: The underlying graph is primarily automatically generated from other data sources and was one of the largest graphs we encountered during our interviews.

We describe the graph, the computation performed on the graph, and the software that stores the graph and performs the computation.

Knowledge Graph:[8] The graph contains three types of nodes:

(i) *Products:* A subset, approximately 10 million, of products sold on Alibaba.
(ii) *Product Categories:* Includes categories such as "shoes," "winter jackets," "TVs," or "electronics." There are approximately 10 thousand of categories.
(iii) *Concepts:* This is an umbrella term to refer to a very large number of concrete and abstract real-world entities. Examples include "football," "sports," "China," "young male," or "born in 1980s." A small part of the concepts are manually curated inside Alibaba and some are obtained from the Chinese edition of Wikipedia. However, majority of them are previous search keywords that users have used. These are extracted from the search logs. As we describe momentarily, these are also the keywords that the application recommends to users. There are approximately 100 million of these.

There are two main edge types:

(i) *Product–category edges:* In most cases, each product belongs to exactly one category. So there are approximately 10 million of these.
(ii) *Concept–product and concept–category edges:* There are edges between concepts and products and between concepts and categories, indicating a relation between a concept and a product or a category. The edges are automatically generated through several techniques, such as an analysis of the logs that contain keywords used by users, their clicks, and purchases or using natural language processing on reviews. To each generated edge, a weight is assigned to indicate the strength of the connec-

[8] Note that the use of term "knowledge graph" vs other terms such as "property graph" or simply "graph" is slightly arbitrary. We found our interviewees referring to any data stored in RDF stores as knowledge graphs. We also found that interviewees referred to graphs that represent abstract things, e.g., keyword topics or concepts, also as knowledge graphs even if they were not stored in an RDF system.

tion between the concept and the product or category. There are over 100 billion of these edges.

Recommendation Computation: The recommendation computation happens as follows: Each user is tagged with a subset of the concepts indicating their known properties, such as "male" vs "female," "IT professional" vs "accountant," or "born in 1980s." There are several dozen such tagged concepts. From each tagged concept, a roughly 4-hop breadth-first search traversal is performed to find new concepts. Each path from a tagged concept to each newly found concept c is given a weight, based on the weights of the edges on the path, and aggregated to give a relevance weight to c. All new concepts are finally ranked, and a subset of them are returned to the users. This entire computation has to happen in 4ms.

Software: The graph is stored in an in-house distributed graph database. The database stores the structure and properties on the nodes and edges in a distributed key-value store. So, both the neighbors and weights are stored as key-value pairs and the 4-hop neighborhoods of nodes often need to be fetched from different machines. Part of the graph is kept in memory and part is kept in SSDs. The database supports the Gremlin language [41] and the property graph model. The traversals are written in the Gremlin traversal API and resulting paths are aggregated in custom code written outside of Gremlin.

5.3.2 Configuration recommendation: Siemens' automation systems

The context of our next application, also described briefly here [58], is the configuration of industrial automation systems, a project by Siemens. Such systems are comprised of a combination of mechanical, hydraulic, and electric devices with complex constraints between individual component and a multitude of choices—for example, the number of input and output ports, line voltage, budget vs. premium options, etc. A user, which might be an end customer or a sales representative, incrementally builds a plan that fulfills all the required functionalities, selecting from a product catalog. While some configuration information is explicitly captured in terms of product features, there is also a large amount of tacit or implicit knowledge. One solution that Siemens has been exploring is the use of recommendation techniques to aid in the selection of components.

The approach combines product information as well as past user behavior, modeled as a knowledge graph and stored in an RDF system. Product information comes from a domain ontology and captures semantic relations such as "has line voltage," "is of type," and "contains," as well as product hierarchies, e.g., "S7-1500" is a subtype of "S7," which in turn is a specialization of a "control system." Information of past behavior comes from historical solutions, i.e., automation solutions that have been previously configured.

The novelty of this project comes from the combination of these two sources of evidence. When asked as to why a knowledge graph as opposed to relational tables for storing and integrating these heterogeneous sources, our interviewee responded with two main reasons: first, the flexibility of the data model and second, a knowledge graph is closer to how users conceptualize the data. Both of these points were echoed by the white papers and other interviewees.

Although the graph database is an important component of the overall solution, its role is little more than a repository of features. The actual recommendation algorithm is based on tensor factorization: the rows and columns correspond to entities (tens of thousands), while each slice corresponds to one relation, e.g., "contains." Given the entities in a partial solution, the system's task is to recommend the most likely item to complete the solution (using previously configured solutions as ground truth). That is, the tensor is materialized from the graph database and used to train a model (written in TensorFlow in this case). Thus, while this is perhaps an example of a machine learning application on graph databases in a technical sense, the integration is rather shallow.

5.4 Fraud and threat detection

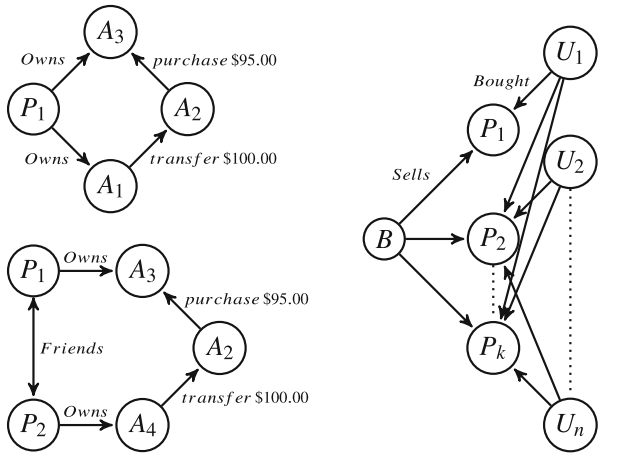
Four applications described in our interview with Alibaba and one application described by one consultant to a large financial institution were related to fraud and threat detection. There were two commonalities between these applications:

(i) *Searching a subgraph pattern:* Each application was based on finding some subgraph pattern, e.g., a bipartite, star, or cycle, in a very large transaction graph.
(ii) *Use of graph visualization:* In each case, the detected pattern was merely a signal of a potential fraud that triggered an inspection by other systems or a human for further investigation. Manual human investigations involved using a graph visualization software and exploring the neighborhoods of the emerged pattern. The consultant also mentioned using visualization for discovering the pattern to search for in the first place.

We briefly review some of these applications and when possible discuss the patterns searched.

5.4.1 Fake transactions on Alibaba.com

This application detects fake transactions initiated by businesses that sell products on Alibaba's e-commerce platform to increase their ranking on the platform. There are two broad patterns the application searches for: (i) cycle patterns shown in Fig. 1a and (ii) a near-complete bipartite clique shown



(a) Cycle patterns. **(b)** Bipartite transaction pattern.

Fig. 1 Patterns for detecting fake transactions

in Fig. 1b. These patterns are detected by different applications on different graphs. We describe the patterns, the input graphs on which the patterns are searched, the software that searches the patterns, and several challenges our interviewee mentioned.

Cycle patterns

Pattern: The top pattern in Fig. 1a is searching for an evidence that there is a business owner `P1` who is paying a fake buyer `P2` to buy products from `P1`. In particular, the pattern searches for transaction where `P1` transfers some amount of money from her account `A1` to an account `A2`, belonging to the fake buyer, which transfers a similar amount of money back to an account `A3` that is also owned by `P1`. In a slightly more advanced version of the pattern, also discussed in a recent publication [76], a friend of `P1` sends some amount of money to the fake buyer to initiate the fake purchase. Our interviewee noted that these cycle patterns are simplified versions of multiple other fraud patterns searched by other internal and external customers of Alibaba on other transaction graphs. Detecting such cyclic patterns in fraud-related applications also appeared in 7 use cases in the white papers.

Input Graph: These cyclic patterns are continuously searched for in a graph that contains data about the financial accounts of the businesses on Alibaba, as well some social connection information, e.g., contact information of Alibaba customers or other available social information.

Software: The current software stack is a bit complex, but briefly involves the following steps: (i) a stream of financial transaction edges are buffered for a period of time, roughly 10 seconds; (ii) the necessary neighborhood of those edges are extracted from a distributed in-house graph database; and (iii) the pattern is searched on the extracted graph in an single-node special solution. The extracted graph is several millions of edges and vertices. The applications latency is roughly 30 seconds.

Bipartite patterns

Pattern: Detecting fraud through cycle patterns is difficult because often money transfers do not go through Alibaba's systems. A more effective way is to find near-complete (and not necessarily fully complete) bipartite graph of products and customers on a graph extracted from the actual transactions on the Alibaba platform.

Such patterns are signals that a large number of fake customers buy the same set of products, say owned by the same business. This activity is similar to click farming [22] to increase ad revenue of Web sites. Part of Alibaba's fraud detection system searches for multiple large instances of the pattern, where the pattern can contain hundreds of products and customers.

Input Graph: In a simplified form, the application runs on a graph that contains businesses, products, and customers as separate nodes and `sells` edges between businesses and products, `purchased` edges between products and customers. These patterns are very complex and detecting them on a large window of purchase transactions is very challenging, so the application limits the window to several days of transactions. This generates an input graph with a few hundred million nodes, and several billion edges.

Software: The application runs off-line and uses a single-node custom-built in-memory graph processing software.

Challenges: One challenge is to detect the nodes that are part of a pattern without enumerating the instances of these patterns. Enumeration of patterns that have a high degree of symmetry is expensive because across two matches of the pattern, there can be a large overlap of the nodes. As a simple example, consider searching for a (10, 10) complete bipartite pattern and the input graph contains a (20, 10) complete bipartite pattern. There will be $\binom{20}{10}$ many instances of the smaller pattern inside the larger pattern, even if there are only 30 different nodes across these matches. A second challenge is scalability. The application would like to search the patterns ideally across hundreds of billions of edges, by generating the graph from a much larger time period of transactions.

Other Patterns: We omit a detailed description, but the interviewee described two other use cases:

- The first application detects gambling activity on Alipay, which is an online payment platform. The input graph contains customers' Alipay accounts as nodes and *Alipay*

groups, a service to allow groups of people to exchange money among themselves. The edges are `membership` edges between customers and groups. For each gambling game, a set of gamblers start and join a new group. Similar to the fake transactions application, the pattern in its simplified form forms a near-complete bipartite graph of accounts and groups.

- Another application monitors attacks and threats on Alibaba's cloud network and traffic graph, which contains information about the hosts, e.g., IP addresses, ports, domain names, and the traffic between the hosts. The application searches a star pattern consisting of a single node with several labeled edges, some with regex patterns, to match address and host name patterns. The graph is highly dynamic, and the application searches patterns on a snapshot that contains only a few days' data, which contains over 100 billion edges.

5.4.2 Application at a large financial institution

Our consultant interviewee described a fraud detection application for a large financial institution that had customer accounts and different transactions between accounts. The searched pattern was quite complex and was not described in detail. Broadly it involves searching for connected accounts over very long paths in the graph, where the nodes that are close to the center of these paths have an unusually high number of transactions, i.e., edges, and amounts of transactions. Interestingly, when asked how they observed that this pattern is a signal of fraud, the interviewee said that initially he manually searched for fraudulent patterns. Specifically, he visualized large chunks of the graph, sometimes containing several thousands of nodes, on a visualization software, and eyeballed known fraudsters' activities. He noticed this pattern as part of this visualization activity.

5.5 Question answering with personal assistant products of Alibaba and Amazon

Alibaba and Amazon both produce voice-controlled personal assistant products, *AliGenie* [5] and *Alexa* [4], respectively, that can be accessed from different devices, such as smart speakers or mobile phones for several services. One of these services is to answer factual questions asked by human users through speech. Our interviewees from both companies described similar applications that use a knowledge graph to answer these questions.

The questions asked by users can be highly varied and require knowledge from public information, corporate information, or user-specific information, e.g., about the movies the user has seen. Our interviewees both described similar applications that use a knowledge graph to answer questions.

In both cases, the interviewees could not provide the details of these graphs but briefly mentioned that the graph used to answer these questions include many internal and external sources. In Alibaba's case, the Chinese edition of Wikipedia, information from Alibaba Music, information about the businesses that sell products on Alibaba were mentioned. Both interviewees mentioned the challenges of integrating such numerous and diverse internal and external sources.

The high-level steps of both applications were very similar and consisted of components that perform voice recognition and natural language processing to understand the important entities used in the query. For example, in the question "What are the movies that Tom Hanks played in 2018?", "Tom Hanks" would be identified as the main entity. Then all nodes that are referred to as Tom Hanks are identified from the graph and a local search is made around these nodes to find nodes that are of type movies and have date information. The details of these searches were not described, but in both cases many nodes will be matched, returned, and ranked before an answer is produced. Both interviewees mentioned doing semantic inference using ontologies, e.g., to infer that the word "played" is semantically related to "acted in," to extend the search or rank the results.

Interviewees provided few details about the software on which the knowledge graph is stored and the search is performed. In the case of Amazon, the graph was stored in RDF format and indexed in an in-house software (not an RDF system). In the case of Alibaba, although each edge (or fact) in the graph was extracted as an RDF triple, the graph was then stored in an in-house graph database that supports the property graph model.

5.6 Contingency analysis of power failures at State Grid

Contingency analysis is a preemptive analysis done on an electric power grid to check the severity of different possible failures.

Our interviewee from StateGrid described a contingency analysis system designed for the grid in one Chinese province. In contrast to other applications which often used one large graph, this application, logically, uses a very large number of small graphs. Interestingly, these graphs are very similar to each other and the application repeats the same computation on each graph in parallel. We describe the input graph, the computation, and the software used by the application.

Input Graph: The application has a *base graph* that represents the components of the power grid using the abstract *bus–branch* standardized model [42]:

- Vertices correspond to buses that represent electrical nodes, which can include power system elements like substations, loads, and generators. Operational parameters such as bus id, load power, voltage magnitude, voltage angle, self-impedance, and power injection are stored as vertex attributes. There are approximately 2.5K vertices.
- Edges correspond to branches that represent electrical paths for current flows, such as transmission lines and transformers. Operational parameters such as power flow, line impedance, and transformer turns ratio are stored as edge attributes. There are approximately 3K edges.

This is a dynamic graph, and the attributes on the edges are changing every few seconds and the topology changes, e.g., a new node is added or removed, every few minutes.

Computation: To determine how the failure of a component affects the flow of power in the grid, the application generates a few thousand logical *derived graphs* from the base graph. Each derived graph modifies the base graph slightly, say by removing a single edge, to simulate a potential failure. For each derived graph G', the application formulates some power equations. We do not provide the details of these equations, but an overview can be found in reference [98]. In a simplified form, readers can think of these as equations of the form $Ax = B$, where A and B are power-related matrices, each row of which represents information about the neighborhood of a vertex in the derived graph. These equations are solved in parallel using matrix factorization. We note that there is a significant potential to reuse the computation results across the derived graphs, as the graphs are very similar. The solution x's are analyzed to assign severity values to each derived graph and an alert is raised for abnormally high severity values, indicating the system has found a potential failure case, which could have severe outcomes.

Software used: The base graph is stored in TigerGraph [93]. Derived graphs are logical and not explicitly stored, but their corresponding matrices A and B are read from TigerGraph in parallel and moved to a custom code that solves the power flow linear algebra equations. The overall latency of the application is several seconds. Although this is not done currently, some equations can also be directly solved using iterative vertex-centric computations on the base graph directly [98].

6 Related work

To the best of our knowledge, our survey is the first study that has been conducted across users and of a wide spectrum of graph technologies and various public information about these technologies to understand graph datasets, computations, and software that is in use, the business applications that use graphs and the challenges users face.

Several surveys in the literature have conducted user studies to compare the effectiveness of different techniques used to perform a particular graph processing task, primarily in visualization [19,46] and query languages [50,74,77]. Additionally, several software vendors have conducted surveys of their users to understand how their software is used to process graphs. Some of these surveys are publicly available [31,67,89]. However, these surveys are limited to studying a specific software product.

There are also numerous surveys in the literature studying different topics related to graph processing. Examples include surveys on query languages for graph database systems and RDF engines [10,44,47], graph algorithms [3,45,53,97], graph processing systems [16,61], and visualization [24,95]. These surveys do not study how users use the technologies in practice.

7 Conclusion and future work

Managing and processing graph data are prevalent across a wide range of fields in research and industry. We surveyed 89 users, interviewed 8 users, and reviewed user emails, code repositories, and white papers of a large suite of software products. The participants' responses and our review provide useful insights into the types of graphs users have, the software and computations users use, the business applications they develop, and the major challenges users face when processing their graphs. We hope that these insights and in particular the challenges we highlight will help guide research on graph processing.

We conclude with two final remarks. First, we found product–order–transaction graphs to be the most popular type of graph. Workloads that process these product data appear in popular SQL benchmarks, such as TPC-C [94], and are well studied in research on relational systems. However, several existing graph benchmarks, such as LDBC [59] and Graph500 [36], do not yet provide workloads and data to process product graphs. One such benchmark is the WatDiv benchmark [7] that generates RDF triples containing information about products and purchases. Developing similar benchmarks and popularizing their use would be highly beneficial to the research community. Such benchmarks are great facilitators of research.

Second, query languages and APIs emerged as one of the top challenges in our survey and certainly the most popular discussion topic in emails and code repositories. These challenges can be partly mitigated by a collaborative effort to standardize the query languages of different graph software that satisfy users' needs. One such successful effort is the adoption of SPARQL as a standard for querying RDF

data. Similar efforts are ongoing for developing standard query languages and JDBC-like interfaces [51] for property graphs, such as the Gremlin language [79] and the efforts to combine openCypher [72], PGQL [78], and G-CORE [9] to create GSQL [43]. There is also ongoing effort to develop a standard set of linear algebra operations for expressing graph algorithms [64].

Acknowledgements We are grateful to Chen Zou for helping us in using online survey tools and drafting an early version of this survey. We are also grateful to Nafisa Anzum, Jeremy Chen, Pranjal Gupta, Chathura Kankanamge, and Shahid Khaliq for their valuable comments on the survey and help in categorizing the academic publications, user emails, and issues. We would like to thank our online survey participants and our interviewees: Brad Bebee, Scott Brave, Jordan Crombie, Luna Dong, William Hayes, Thomas Hubauer, Peter Lawrence, Stephen Ruppert, Chen Yuan, with a special thanks to Zhengping Qian for the many hours of follow-up discussions after our interview. Finally, we would like to thank the anonymous reviewers for their valuable comments. This research was partially supported by multiple Discovery Grants from the Natural Sciences and Engineering Council (NSERC) of Canada.

A Choices of graph computations

One way to ask this question is to include a short-answer question that asks "What queries and graph computations do you perform on your graphs?". However, the terms graph queries and computations are very general and we thought this version of the question could be under-specified. We also knew that participants respond less to short-answer questions, so instead we first asked a multiple-choice question followed by a short-answer question for computations that may not have appeared in the first question as a choice.

In a multiple-choice question, it is very challenging to provide a list of graph queries and computations from which participants can select, as there is no consensus on what constitutes a graph computation, let alone a reasonable taxonomy of graph computations. We decided to select a list of graph queries and computations that appeared in the publications of six conferences, as described in Sect. 2.2. We use the term graph computation here to refer to a query, a problem, or an algorithm.

For each of the 90 papers, we identified each graph computation, if (i) it was directly studied in the paper; or (ii) for papers describing a software, it was used to evaluate the software. We used our best judgment to categorize the computations that were variants of each other or appeared as different names under a single category. For example, we identified motif finding, subgraph finding, and subgraph matching as *subgraph matching*. When reviewing papers studying linear algebra operations, e.g., a matrix–vector multiplication, for solving a graph problem such as BFS traversal, we identified the graph problem and not the linear algebra operation as a computation.

Finally, for each identified and categorized computation, we counted the number of papers that study it and selected the ones that appeared in at least 2 papers. In the end, we provided the participants with the 13 choices that are shown in Table 10.

B Choices of machine learning computations

Similar to graph computation, machine learning computation is a very general term. Instead of providing a list of ad hoc computations as choices, we reviewed each machine learning computation that appeared in the 90 graph papers we had selected. Specifically, the list of machine learning computations we identified included the following: (i) *high-level classes of machine learning techniques*, such as clustering, classification, and regression; (ii) *specific algorithms and techniques*, such as stochastic gradient descent and alternating least squares that can be used as part of multiple higher-level techniques; and (iii) *problems* that are commonly solved using a machine learning technique, such as community detection, link prediction, and recommendations. We then selected the computations, i.e., high-level techniques, specific techniques, or problems, that appeared in at least 2 papers. As in the graph computations question, we used our best judgment to identify and categorize similar computations under the same name.

C Storage in multiple formats

We asked the 33 participants who said that they store their data in multiple formats, which formats they use as a short-answer question. Out of the 33 participants, 25 responded. Their responses contained explicit data storage formats as well as the internal formats of different software. Table 19 shows the number of responses we received for the main formats. A relational database and a graph database format combination was the most popular combination. Other combinations varied significantly, examples of which include HBase and Hive, GraphML and CSV, and XML and triple-store.

D Other tables from the survey

Table 20 shows the sizes of graphs we found in user emails and issues. Table 21 shows the number of emails and issues we identified for each specific challenge we discussed in Sect. 3.4.2. Table 22 shows the total number of emails and issues we reviewed for each software product from January

Table 19 Data storage formats

Data storage format	#
Graph databases	10
Relational databases	8
RDF store	5
NoSQL store (Key-value, HBase)	5
XML/JSON	4
JGF/GML/GraphML	4
CSV/text files	3
Elasticsearch	3
Binary	2

Table 20 Graph sizes in user emails and issues

Vertices	#
(a) *Number of vertices*	
100 M–1 B	10
1 B-10 B	17
10 B–100 B	1
> 100 B	2
Edges	**#**
(b) *Number of edges*	
1 B–10 B	42
10 B–100 B	17
100 B–500 B	6
> 500 B	1

Table 21 Challenges found in user emails and issues

Challenge	#
Graph DBs and RDF engines	
High-degree vertices	24
Hyperedges	18
Triggers	18
Versioning and historical analysis	14
Schema and constraints	10
Visualization software	
Layout	31
Customizability	30
Large graph visualization	8
Dynamic graph visualization	4
Query languages	
Subqueries	7
Querying across multiple graphs	6
DGPS and graph libraries	
Off-the-shelf algorithms	41
Graph generators	7
GPU support	3

to September of 2017. The table also shows the number of commits in the code repositories of each software product during the same period.

E Other applications from interviews

Large-Scale Data Integration for Analysis of Turbines: Our interviewee from Siemens described an application that Siemens engineers use to analyze different properties of gas turbines Siemens produces using a knowledge graph. The application emphasized the advantage of using graphs to integrate different sources of corporate data, in this case mainly about where turbines are installed, measurements from the installed turbines' sensors, and information about maintenance activity on the turbines. The knowledge graph is stored in an RDF engine and engineers ask queries, such as "What is the mean time failure of turbines with coating loss?" through a visual interface where they navigate the different types of nodes in the knowledge graph and express aggregations. The visually expressed queries get translated to SPARQL queries.

Contact Deduplication: One of our interviews was with two engineers from a company called FullContact that manages contact information about individuals by integrating public and manually curated information, which is sold to other businesses. An over 10 B-edge and 4 B-vertex graph models this contact information as follows: nodes represent different pieces of information, such as addresses, phone numbers, and edges between nodes indicate the likelihood that the information belongs to the same individual. The company uses GraphX to run a connected component-like algorithm to finding the contacts that are likely to belong to the same individual.

Other applications using knowledge graphs: One of our interviewees was a consultant to a chemical company specializing in agricultural chemicals. The company has an over 30 billion-edge knowledge graph on pesticides, seeds, chemicals that are stored in a commercial RDF system. This graph is used by many applications, such as, to track the evolution of seeds, to power internal wiki pages and tools used by analysts. Another interviewee was the founder of a start-up that works on tools that can be used by biologists to publish biological knowledge. Our interviewee described examples of knowledge graphs that model the cellular activity in the context of different species' different tissues. Triples included facts about which genes transcribe which protein and which proteins' presence decreases other proteins' presence, etc. [71]. The example applications were similar broadly to our interviewee from the chemical company and power wikis and Web site which biologists use to analyze these interactions.

Table 22 Number of reviewed emails and issues, and the code commits in the repositories of each software product

Technology	Software	#Emails	#Issues	#Commits
Graph database system	ArrangoDB	140	466	5264
	Caley	50	57	151
	DGraph	175	558	760
	JanusGraph	225	308	411
	Neo4j	286	243	4467
	OrientDB	169	668	918
	Sparksee	8	NA	NA
RDF engine	Apache Jena	307	126	471
	Virtuoso	72	61	179
Distributed graph processing engine	Apache Flink (Gelly)	34	68	48
	Apache Giraph	19	34	23
	Apache Spark (GraphX)	23	28	11
Query language	Gremlin	409	206	1285
Graph library	Graph for Scala	10	12	18
	GraphStream	18	26	7
	Graphtool	121	66	172
	NetworKit	37	30	236
	NetworkX	78	148	171
	SNAP	57	17	34
Graph visualization	Cytoscape	388	264	8
	Elasticsearch X-Pack Graph	50	38	NA
	Gephi	NA	147	10
	Graphviz	NA	58	277
Graph representation	Conceptual graphs	30	NA	NA

References

1. Cyber Threat Intelligence. https://bitnine.net/agensgraph-usecase-cyber-threat-intelligence-en
2. Personalized Education Service. https://bitnine.net/agensgraph-usecase-personalized-education-service-en
3. Aggarwal, C.C., Wang, H.: Graph Data Management and Mining: A Survey of Algorithms and Applications, pp. 13–68. Springer, Berlin (2010)
4. Alexa. https://en.wikipedia.org/wiki/Amazon_Alexa
5. AliGenie. https://en.wikipedia.org/wiki/AliGenie
6. AllegroGraph. https://franz.com/agraph/allegrograph
7. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified Stress Testing of RDF Data Management Systems. In: ISWC (2014)
8. Amer-Yahia, S., Pei, J. (eds.): PVLDB, Volume 11 (2017–2018). http://www.vldb.org/pvldb/vol11.html
9. Angles, R., Arenas, M., Barceló, P., Boncz, P.A., Fletcher, G.H.L., Gutierrez, C., Lindaaker, T., Paradies, M., Plantikow, S., Sequeda, J.F., van Rest, O., Voigt, H.: G-CORE: a core for future graph query languages. In: Proceedings of International Conference on Management of Data (2018)
10. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. **50**(5), 68 (2017)
11. AnzoGraph. https://www.cambridgesemantics.com/product/anzograph
12. Arpaci-Dusseau, A.C., Voelker, G. (eds.): Proceedings of the Symposium on Operating Systems Design and Implementation. USENIX Association (2018). https://www.usenix.org/conference/osdi18
13. ArrangoDB. https://www.arangodb.com
14. AboutYou Data-Driven Personalization with ArangoDB. https://www.arangodb.com/why-arangodb/case-studies/aboutyou-data-driven-personalization-with-arangodb
15. Balcan, M., Weinberger, K.Q. (eds.): Proceedings of the International Conference on Machine Learning. JMLR.org (2016). http://jmlr.org/proceedings/papers/v48
16. Batarfi, O., Shawi, R.E., Fayoumi, A.G., Nouri, R., Beheshti, S.M.R., Barnawi, A., Sakr, S.: Large scale graph processing systems: survey and an experimental evaluation. Cluster Comput. **18**(3), 1189–1213 (2015)
17. Basic Linear Algebra Subprograms. http://www.netlib.org/blas
18. Boncz, P., Salem, K. (eds.): PVLDB, Volume 10 (2016–2017). http://www.vldb.org/pvldb/vol10.html
19. Bridgeman, S., Tamassia, R.: A User Study in Similarity Measures for Graph Drawing, pp. 19–30. Springer, Berlin (2001)
20. Caley. https://cayley.io
21. Ching, A., Edunov, S., Kabiljo, M., Logothetis, D., Muthukrishnan, S.: One trillion edges: graph processing at facebook-scale. PVLDB **8**(12), 1804–1815 (2015)
22. Click Farm. https://en.wikipedia.org/wiki/Click_farm
23. Conceptual Graphs. http://conceptualgraphs.org
24. Cui, W., Qu, H.: A Survey on Graph Visualization. PhD Qualifying Exam Report, Computer Science Department, Hong Kong University of Science and Technology (2007)
25. Cytoscape. http://www.cytoscape.org

26. DGraph. https://dgraph.io
27. DTD and XSD XML Schemas. https://www.w3.org/standards/xml/schema
28. Dy, J.G., Krause, A. (eds.): Proceedings of the International Conference on Machine Learning. JMLR.org (2018). http://jmlr.org/proceedings/papers/v80/
29. Elasticsearch X-Pack Graph. https://www.elastic.co/products/x-pack/graph
30. Apache Flink. https://flink.apache.org
31. Apache Flink User Survey 2016. https://github.com/dataArtisans/flink-user-survey-2016
32. FullContact. https://www.fullcontact.com
33. Gephi. https://gephi.org
34. Apache Giraph. https://giraph.apache.org
35. Graph for Scala. http://www.scala-graph.org
36. Graph 500 Benchmarks. http://graph500.org
37. GraphStream. http://graphstream-project.org
38. Graph-tool. https://graph-tool.skewed.de
39. Graphviz. https://graphviz.readthedocs.io
40. Apache Spark GraphX. https://spark.apache.org/graphx
41. Apache TinkerPop. https://tinkerpop.apache.org
42. Group, W.: Common format for exchange of solved load flow data. IEEE Trans. Power App. Syst. **92**(6), 1916–1925 (1973)
43. GQL Standard. https://www.gqlstandards.org
44. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A Comparison of RDF Query Languages, pp. 502–517. Springer, Berlin (2004)
45. Herman, I., Melançon, G., Marshall, M.S.: Graph visualization and navigation in information visualization: a survey. IEEE Trans. Vis. Comput. Graph. **6**(1), 24–43 (2000)
46. Holten, D., van Wijk, J.J.: A User Study on Visualizing Directed Edges in Graphs. In: Proceedings of International Conference on Human Factors in Computing Systems (2009)
47. Holzschuher, F., Peinl, R.: Performance of graph query languages: comparison of Cypher, Gremlin and Native Access in Neo4j. In: Proceedings of the Joint EDBT/ICDT Workshops (2013)
48. Jagadish, H.V., Zhou, A. (eds.): PVLDB, Vol. 7 (2013–2014). http://www.vldb.org/pvldb/vol7.html
49. JanusGraph. http://janusgraph.org
50. Jayaram, N., Khan, A., Li, C., Yan, X., Elmasri, R.: Querying knowledge graphs by example entity tuples. In: Proceedings of International Conference on Data Engineering (2016)
51. JDBC. http://www.oracle.com/technetwork/java/overview-141217.html
52. Apache Jena. https://jena.apache.org
53. Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., Giannopoulou, E.: Ontology visualization methods: a survey. ACM Comput. Surv. **39**(4), 10 (2007)
54. Proceedings of the International Conference on Knowledge Discovery and Data Mining. ACM (2015). http://dl.acm.org/citation.cfm?id=2783258
55. Proceedings of the International Conference on Knowledge Discovery and Data Mining. ACM (2017). http://dl.acm.org/citation.cfm?id=3097983
56. Proceedings of the International Conference on Knowledge Discovery and Data Mining. ACM (2018). http://dl.acm.org/citation.cfm?id=3219819
57. Keeton, K., Roscoe, T. (eds.): Proceedings of the Symposium on Operating Systems Design and Implementation. USENIX Association (2016). https://www.usenix.org/conference/osdi16
58. Knowledge Graph at Siemens. https://youtu.be/9pmQXua9LWA?t=1109
59. LDBC Benchmarks. http://ldbcouncil.org/benchmarks
60. Letunic, I., Bork, P.: Interactive tree of life: an online tool for phylogenetic tree display and annotation. Bioinformatics **23**(1), 127–128 (2006)
61. Lu, Y., Cheng, J., Yan, D., Wu, H.: Large-scale Distributed graph computing systems: an experimental evaluation. PVLDB **8**(3), 281–292 (2014)
62. Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: A system for large-scale graph processing. In: Proceedings of International Conference on Management of Data (2010)
63. MATLAB. https://www.mathworks.com
64. Mattson, T., Bader, D.A., Berry, J.W., Buluç, A., Dongarra, J.J., Faloutsos, C., Feo, J., Gilbert, J.R., Gonzalez, J., Hendrickson, B., Kepner, J., Leiserson, C.E., Lumsdaine, A., Padua, D.A., Poole, S., Reinhardt, S.P., Stonebraker, M., Wallach, S., Yoo, A.: Standards for graph algorithm primitives. In: Proceedings of High Performance Extreme Computing Conference (2013)
65. Neo4j. https://neo4j.com
66. Detect Fraud in Real Time with Graph Databases. https://neo4j.com/whitepapers/fraud-detection-graph-databases
67. The 2016 State of the Graph Report. https://neo4j.com/resources/2016-state-of-the-graph
68. NetworKit. https://networkit.iti.kit.edu
69. NetworkX. https://networkx.github.io
70. GraphDB by Ontotext. https://www.ontotext.com/products/graphdb
71. OpenBEL. http://openbel.org
72. openCypher. http://www.opencypher.org
73. OrientDB. https://orientdb.com
74. Pienta, R., Tamersoy, A., Endert, A., Navathe, S., Tong, H., Chau, D.H.: VISAGE: Interactive visual graph querying. In: Proceedings of International Working Conference on Advanced Visual Interfaces (2016)
75. Precup, D., Teh, Y.W. (eds.): Proceedings of the International Conference on Machine Learning. JMLR.org (2017). http://jmlr.org/proceedings/papers/v70
76. Qiu, X., Cen, W., Qian, Z., Peng, Y., Zhang, Y., Lin, X., Zhou, J.: Real-time constrained cycle detection in large dynamic graphs. PVLDB **11**(12), 1876–1888 (2018)
77. Rath, M., Akehurst, D., Borowski, C., Mäder, P.: Are graph query languages applicable for requirements traceability analysis? In: Proceedings of International Conference on Requirements Engineering: Foundation for Software Quality (2017)
78. van Rest, O., Hong, S., Kim, J., Meng, X., Chafi, H.: PGQL: a property graph query language. In: Proceedings of Graph Data Management Experiences and Systems (2016)
79. Rodriguez, M.A.: The Gremlin Graph Traversal Machine and Language. CoRR arXiv:1508.03843 (2015)
80. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press (2016). https://dl.acm.org/citation.cfm?id=3014904
81. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press (2017). https://dl.acm.org/citation.cfm?id=3126908
82. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press (2018). https://dl.acm.org/citation.cfm?id=3291656
83. Sharma, A., Jiang, J., Bommannavar, P., Larson, B., Lin, J.: GraphJet: real-time content recommendations at twitter. PVLDB **9**(13), 1281–1292 (2016)
84. SNAP: Standford Network Analysis Project. https://snap.stanford.edu
85. Proceedings of the Symposium on Cloud Computing. ACM (2015). http://dl.acm.org/citation.cfm?id=2806777
86. Proceedings of the Symposium on Cloud Computing. ACM (2017). http://dl.acm.org/citation.cfm?id=3127479
87. Proceedings of the Symposium on Cloud Computing. ACM (2018). http://dl.acm.org/citation.cfm?id=3267809

88. Proceedings of the Symposium on Operating Systems Principles. ACM (2017). http://dl.acm.org/citation.cfm?id=3132747
89. Apache Spark—Preparing for the Next Wave of Reactive Big Data. https://info.lightbend.com/white-paper-spark-survey-trends-adoption-report-register.html
90. Sparksee. http://www.sparsity-technologies.com
91. Stardog. https://www.stardog.com
92. State Grid. http://www.sgcc.com.cn/ywlm/index.shtml
93. TigerGraph. https://www.tigergraph.com
94. The TPC-C Benchmark. http://www.tpc.org/tpcc
95. Vehlow, C., Beck, F., Weiskopf, D.: Visualizing group structures in graphs: a survey. Computer Graphics Forum **36**(6), 201–225 (2017)
96. OpenLink Virtuoso. https://virtuoso.openlinksw.com
97. Wang, C., Tao, J.: Graphs in scientific visualization: a survey. Computer Graphics Forum **36**(1), 263–287 (2017)
98. Zhao, Y., Yuan, C., Liu, G., Grinberg, I.: Graph-based preconditioning conjugate gradient algorithm for "N-1" contingency analysis. In: IEEE Power Energy Society General Meeting (2018)