Graph databases in electronic communications network: assessment based on query execution time

Oksana Herasymenko[1,*,†], and Anna Ivanytska[1,†]

[1] Taras Shevchenko National University of Kyiv, 60 Volodymyrska Street, Kyiv, 01033, Ukraine

Abstract
Graph databases are a powerful data structure that can be applied to solve a variety of problems. They are widely used in electronic communications network due to the need for effective management of complex network structures, where traditional relational databases do not always provide sufficient performance and flexibility. There are many graph mapping algorithms. Each of them has its own characteristics and best areas of use. In this paper, Neo4j, Memgraph and ArangoDB graph databases are considered and compared for query performance in an experiment on the same data set by measuring query execution time. Also, a usability estimation is given for Neo4j, Memgraph and ArangoDB, which based on number of GitHub users, number of image downloads, support ability, deployment ability and supported programming languages.

Keywords
Graph databases, communications, network, Cypher query language, average query execution time, Neo4j, Memgraph, ArangoDB

1. Introduction

Graph databases store data in the form of graphs. A graph is a collection of vertices (nodes) and edges (relations) connecting them. Graphs represent a set of entities called nodes, and a set of relations – the ways in which these entities are connected with each other [1].

Graph databases appeared relatively recently. Their value lies in the ability to work effectively with data that has complex relationships. This allows building new relationships quickly, which has a significant impact on business and other organizations.

They also provide high productivity when performing queries on graph structures, as they are optimized for working with relations and nodes [5].

Graph databases are widely used in various fields due to their ability to effectively model complex relationships between data. In health care, they are used to manage medical records and predict diseases, which improves the quality of medical services [2]. In business analytics, they are used to analyze the interaction between customers and companies. This allows to improve customer relationship management (CRM) and offers personalized recommendations based on graph models [3].

Also, graph databases are used in the field of machine learning to create graph-based models, which allows to improve the accuracy of forecasting and data analysis using graph neural networks (GNN) [4].

Information Technology and Implementation (IT&I-2024), November 20-21, 2024, Kyiv, Ukraine

* Corresponding author.
† These authors contributed equally.

oksgerasymenko@knu.ua (O. Herasymenko); aniaiv@knu.ua (A. Ivanytska)
0000-0001-6804-2125 (O. Herasymenko); 0009-0002-6753-3533 (A. Ivanytska)

Graphs are inherently additive, which means we can add new kinds of connections, nodes, labels, and subgraphs to an existing structure without breaking existing queries and application functionality.

Modern graph databases provide an ability to perform seamless development. In particular, the schema-free nature of the graph data model, combined with the ability to test the graph database application programming interface (API) and query language, allows to develop the application in a controlled manner.

Graph databases carry out a special role in electronic communications network, since a significant part of the algorithms performing in such systems are based on graphs. Therefore, the use of graph databases for this area is ponderable or even inevitable. Applying the graph databases in electronic communications network is discussed in the next section of this paper.

Importance of graph databases is unlikely to be overestimated in our time. However, these databases also have some limitations, which should be taken into account when choosing this technology to solve certain problems. One of the main limitations is scalability. Although graph databases are great at managing relationships, they can struggle with horizontal scaling, which is more easily implemented in other NoSQL databases. Another limitation is the complexity of graph algorithms and queries, which can be computationally intensive and slow down as the graph size increases. In addition, the integration of graph databases with existing data systems can be difficult, requiring significant changes in data models and application logic [5].

Also, it is worth noting that there is no standardized query language for graph databases. It depends on the particular database. This can make it difficult to migrate from one provider to another, as different query languages may differ in functionality and syntax.

2. Graph databases in electronic communications network

Graph databases have also found a prominent place in electronic communications network. They are widely used to improve network management, optimize traffic and ensure communication reliability. Graph databases make it possible to model complex network topologies more effectively, analyze connections between nodes and optimize routing.

For instance, paper [6] describes how graph structures are used in various network scenarios such as wireless, wired, and software-defined networks, including for network monitoring and failure prediction. Paper [7] presents a classification of recent studies using graph neural network (GNN)-based approaches to control policy optimization, including offloading strategies, routing optimization, virtual network function orchestration, and resource allocation.

Social networks are a large graph itself, that require multiple graph database servers to store and manage them. Each database server hosts a graph partition for load balancing purposes. Achieving these goals requires a dynamic redistribution algorithm. The following paper [8] provides a lightweight reallocation tool that dynamically changes the partition using a small number of resources. The redistribution tool has been integrated into Hermes, developed as an extension of the open-source Neo4j graph database framework to support partitioned graph data workloads.

The knowledge graph, built using this data, provides a single, semantically enriched model that can support complex, cross-referenced queries. For instance, the system can identify users affected by a network incident, analyze user behavior trends, and suggest optimizations for network performance. Integrating these diverse data models into a graph database enables a user-aware network monitoring approach that links technical data with business insights, enhancing operational efficiency and service quality for electronic communication providers.

The following paper [9] describes the use of a graph database, named Nepal, designed to support the automated management of networks, especially within virtualized and software-defined networks (SDN). Nepal's architecture organizes the network into a layered graph, with each layer representing different network components, from virtual functions to physical hardware. This structure allows operators to efficiently query connections and relationships between elements.

Nepal's unique query language treats network paths as core elements, making it easier to calculate data routes, analyze dependencies, and assess the impact of component failures. Its temporal features enable "time-travel" queries, allowing operators to view and troubleshoot past network states, helping them accurately diagnose issues based on historical configurations.

Paper [10] introduces FrauDetector, a framework for detecting fraudulent phone calls in electronic communications using graph-mining techniques. Unlike traditional classifiers, FrauDetector relies solely on electronic communication records, constructing weighted graphs to model interactions between users and remote numbers. By applying a weighted Hyperlink-Induced Topic Search (HITS) algorithm, it calculates a "trust value" for each phone number to assess fraud likelihood. Two graphs – a user-phone graph (UPG) and a contact book-phone graph (CPG) capture patterns in call frequency and duration. These patterns are analyzed to distinguish fraudulent from legitimate calls. Tests with real-world data from the Whoscall app show that FrauDetector outperforms traditional methods, especially when user profile information is limited. This framework offers a promising approach for detecting fraud in telecommunication networks.

3. Problem statement

Databases are a critical part of the electronic communications industry as they store and manage a vast amount of data relating to various aspects of electronic communications infrastructure, services and customers. Today's electronic communications companies' databases reflect a shift to more intelligent, secure, and flexible data management practices driven by the increasing volume and complexity of data. Leveraging these advances allows electronic communications companies to gain a competitive advantage, improve operational efficiency, and provide innovative services to their customers.

Graph databases are becoming indispensable components of electronic communications network as they are designed to store and process data with a significant level of relationships between entities and with the possibility of painless changes of these relationships.

The essence of such kinds of databases implementation is that actually they are a superstructure on other types of data models, such as relational, NoSQL or others. As already mentioned above, there is no standardization in this area, which causes portability and other issues.

Since the development and improvement of graph databases are still ongoing, it is important to study various aspects of their behavior, as their drawbacks identification can help to improve similar solutions and contribute to a better understanding of options and approaches to their use. In addition, such studies allow us to outline the possible problems of using one or another database as a mean of implementing a certain technological solution. Therefore, the topic of this study is relevant and requires meticulous work.

This study aims to evaluate the performance of query processing for queries of different purposes in different graph databases. To be more specific, the following selections in graph databases are of particular interest and importance:

- counting nodes, which meet the specified criteria;
- searching for the shortest path between nodes;
- to determine if there are any paths between nodes within reach in n-steps and others.

The obtained evaluations can also provide some support in choosing among graph databases for a specific task.

4. Related works

There can be found a lot of studies comparing graph databases and other types of databases. Let's briefly consider some of them. The most attention was paid to the works related to graph databases as they represent the most interest to the topic of this research.

4.1. A comparison of current graph database models

In paper [11], graph database models are systematically compared. The overview includes general features, data modeling features, and support for basic graph queries.

The paper presents comparison tables of such features: data storage, operation and management (availability of query language, graphical interface, API), data graph structure (simple, with attributes, hypergraph, nested graph, labeled node, with attributes, directed edges, labeled), query tools, integrity constraints.

4.2. A comparison of a graph database and a relational database

Article [12] compares graph Neo4j and relational MySQL databases. They were compared on the basis of objective indicators obtained during the experiment and subjective ones. The data for the experiment was randomly generated to obtain a directed acyclic graph (DAG). After loading the data into the respective databases, information about their size was provided. MySQL took up less memory in almost all cases. The time needed for database creation was not considered.

The experiment aimed to measure an execution time of such requests as: counting nodes of a graph with a depth of 4 and 128, counting the number of nodes that had a node parameter of a certain value.

Queries were made on a database with a different number of nodes (1000, 5000, 10000, 100000), which allowed us to get an insight into how the selected systems scaled. It should be noted that the experiment was conducted on numerical and linear parameters.

A subjective comparison was made on such characteristics as: maturity, level of support, ease of programming, flexibility and security.

As a result, Neo4j outperformed MySQL in graph traversal queries by several times. In queries that did not require traversal, MySQL was faster for integer processing and Neo4j for string processing.

4.3. An empirical comparison of graph databases

Article [13] presents GDB (Graph Database Benchmark), a distributed benchmarking platform with open-source Java code to test and compare different graph databases.

Comparison results of the following graph databases are also given: Neo4j, OrientDB, TitanBerkeleyDB, Titan-Cassandra and DEX.

As a result of the experiment, Neo4j got the best results with workaround workloads. Neo4j, DEX, Titan-BerkeleyDB, and OrientDB all achieved similar performance on intensive read-only workloads, but for read-write workloads, the performance of Neo4j, Titan-BerkeleyDB, and OrientDB drops dramatically.

Paper [14] describes a comprehensive experimental comparison of several graph database management systems (GDBs). The experiment aimed to evaluate and compare the performance of selected GDBs using a benchmarking tool called BlueBench, developed specifically for this purpose. Databases Neo4j, DEX, InfiniteGraph, OrientDB, Titan, TinkerGraph and others had been explored during the experiment.

The key performance indicators that had been taken into account during the experiment were execution time, scalability, transaction support, and query efficiency. The benchmarking involved running a series of predefined queries and operations on each database, measuring the tasks executing time and databases large-scale data handling. Tests were automated and aimed to be as consistent as possible across different systems to ensure fair comparison.

The results showed significant variations in between the different GDBs. Neo4j, DEX and TinkerGraph performed well in many areas among other graph databases.

5. Current study database assortment

Of particular interest in this study are those graph databases that are free, open-source and have use cases related with networks, traffic management and so on. Therefore, attention is focused on the following databases: Neo4j, Memgraph and ArangoDB. Let us provide some other arguments for this choice.

Neo4j is considered one of the most popular graph databases, it can be installed on the machine locally. This database has a lot of study materials, tutorials and examples so it is relatively easy to learn how to use it.

Memgraph is compatible with Neo4j and also uses Cypher language to write queries so it is easy to rewrite queries to use in this database. It is well-documented, and has libraries for various programming languages (at least twelve according documentation). This database also has good reviews and tutorials.

ArangoDB supports a lot of data models and, according to its documentation, can be a great fit for use cases like network operations, traffic management, collect IoT data etc.

5.1. Neo4j

Neo4j [15] is one of the leading open-source graph databases. It was developed in 2007 and is a Java-based No-SQL database. It is described as «high-speed graph database with unbounded scale, security, and data integrity for mission-critical intelligent applications» [16]. Key features of Neo4j:

- Cypher, a SQL-like query language, is used for queries;
- LPG (Labeled Property Graph) in Neo4j is a graph data model that represents data using nodes, relationships, and properties;
- the scheme is optional when deploying a data set;
- ACID compliant. Neo4j database integrity is based on atomicity, consistency, isolation, and durability;
- supports data export in *.json and Excel formats.

Neo4j Desktop is a local environment for developing and managing graph databases on the Neo4j platform. Neo4j Desktop version 4.28.0 was installed and DBMS of version 5.20.0 enterprise were used to conduct an experiment, which is described in the following section of this paper. It is important to mention that APOC 5.20.0 plugging was used to get *.csv file from *.dump file.

5.2. Memgraph

Memgraph is a graph database designed for real-time processing and analysis of large volumes of data. It was developed and presented in 2016. It is noted that «the fact that Memgraph is written in C++ and resides in memory means that it is much faster than anything we have seen on the market» [17].

Key features of Memgraph:

- supports Cypher query language;
- supports ACID in-memory transactions for fast access, storing all records in persistent memory;
- ensures high performance of both transactional and analytical queries, even in highly competitive environments;
- supports data export in *.csv and *.json formats.

Memgraph was deployed locally in a container using Docker version 4.33.0 for the current study. The following image for the container was used: memgraph/ memgraph-platform that included memgraph/memgraph-mage version latest and memgraph/lab version latest.

5.3. ArangoDB

ArangoDB is a native multi-model database that supports graph, document, and key-value data models. It was initially released in 2011 and is implemented in C++ as a NoSQL database. ArangoDB allows to create a complex application with various data models within a single backend.

Key features of ArangoDB:

- supports AQL (ArangoDB Query Language), which is similar to SQL and designed to handle both structured and unstructured data;
- provides multi-model capabilities, allowing the use of graph, document, and key-value stores in one engine;
- ACID compliance;
- ArangoDB's graph model supports both property graphs and the Resource Description Framework (RDF);
- features built-in clustering and horizontal scalability, facilitating large-scale distributed deployments;
- supports data export in formats like *.json and *.csv.

ArangoDB can be run in various environments, including local installations, containers and cloud platforms. The following image was used to run ArangoDB locally on Docker.

6. Experiment setup

To compare graph databases an objective and subjective indicators were used. The objective ones rest on comparing results of performing queries speed; subjective ones estimate the level of support and ability of use in different environments.

6.1. Dataset description

The Neo4j dataset of network management [18] was used for the experiment.

The dataset consists of 17 unique labels that form 18 different nodes and are interconnected by 12 unique relationship types. This forms approximately 83847 nodes and 181995 relationships.

It includes node types such as DataCenter, Router, Interface, Port, Rack, Switch, Version, Software, etc. Figure 1 shows the scheme of this dataset.

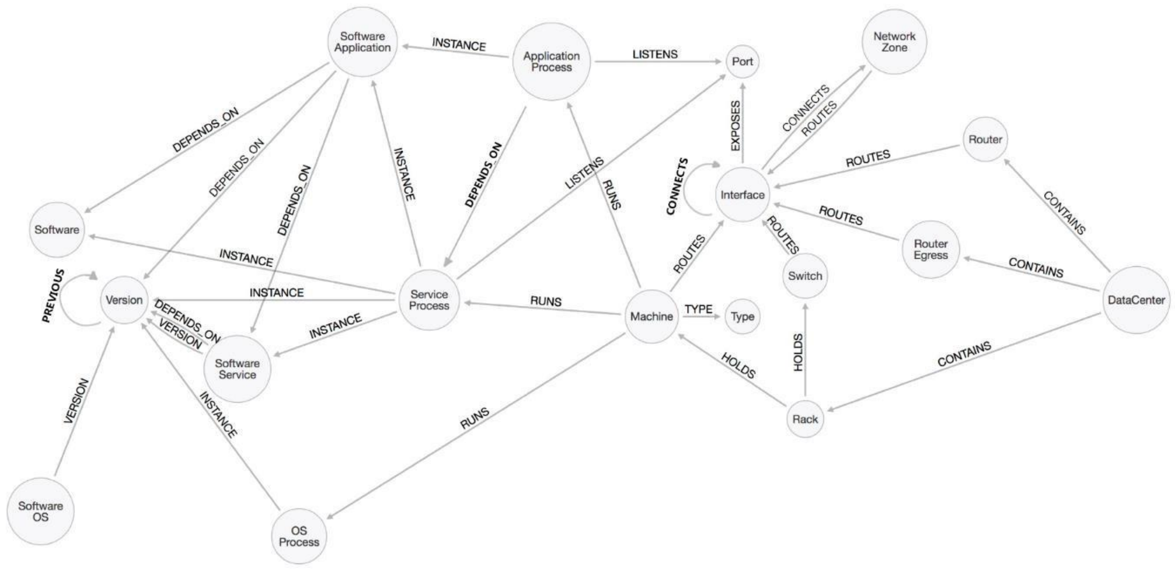


Figure 1: A scheme of network management dataset provided by Neo4j

Table 1
Number of nodes and relationships in graphs used as datasets for the experiment

	Number of nodes	Number of relationships	Comments
10less	8474	12970	Approximately 10 times less than full (in number of nodes)
5less	16854	25933	Approximately 5 times less than full (in number of nodes)
full	83847	181995	

This dataset was taken in three different sizes for the experiment. The largest one contains all nodes and relationships. The middle one has approximately five times less elements of the nodes which aggregate the most data. The smallest one has ten times less data than the largest dataset.

The exact number of nodes and relationships is shown in Table 1.

Performing same queries on graphs of different sizes and comparing obtained results allows us to evaluate scaling ability of databases.

6.2. Experiment conditions

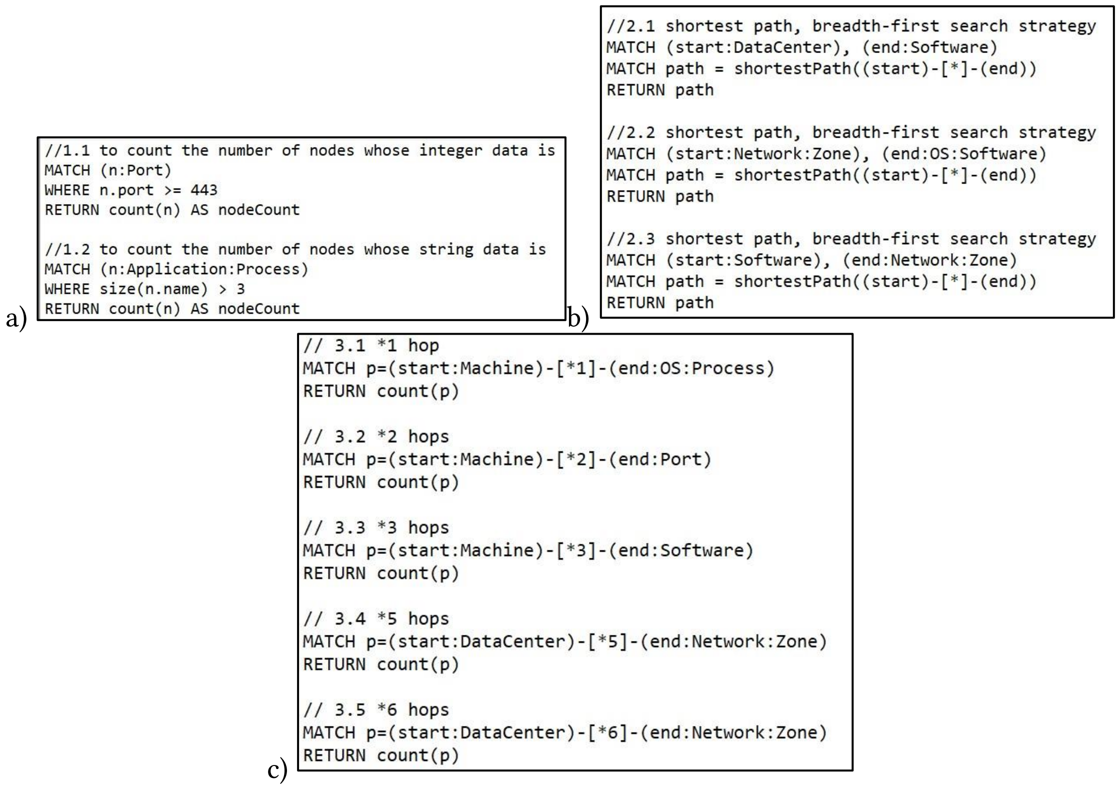
The experiment was carried out using a computer running on the Windows 10 Pro operating system. The machine has 8 GB of RAM and an Intel Core i3-6100 processor operating at a frequency of 3.70 GHz.

Graph databases were launched alternately on the machine during the experiment.

Each query was run ten times. Two maximum and minimum values were discarded and the average of the rest values was taken as the final result.

6.3. Queries

Ten different queries were used in databases' evaluation. They were grouped by similarity into three groups (Figure 2).



a)

```
//1.1 to count the number of nodes whose integer data is
MATCH (n:Port)
WHERE n.port >= 443
RETURN count(n) AS nodeCount

//1.2 to count the number of nodes whose string data is
MATCH (n:Application:Process)
WHERE size(n.name) > 3
RETURN count(n) AS nodeCount
```

b)

```
//2.1 shortest path, breadth-first search strategy
MATCH (start:DataCenter), (end:Software)
MATCH path = shortestPath((start)-[*]-(end))
RETURN path

//2.2 shortest path, breadth-first search strategy
MATCH (start:Network:Zone), (end:OS:Software)
MATCH path = shortestPath((start)-[*]-(end))
RETURN path

//2.3 shortest path, breadth-first search strategy
MATCH (start:Software), (end:Network:Zone)
MATCH path = shortestPath((start)-[*]-(end))
RETURN path
```

c)

```
// 3.1 *1 hop
MATCH p=(start:Machine)-[*1]-(end:OS:Process)
RETURN count(p)

// 3.2 *2 hops
MATCH p=(start:Machine)-[*2]-(end:Port)
RETURN count(p)

// 3.3 *3 hops
MATCH p=(start:Machine)-[*3]-(end:Software)
RETURN count(p)

// 3.4 *5 hops
MATCH p=(start:DataCenter)-[*5]-(end:Network:Zone)
RETURN count(p)

// 3.5 *6 hops
MATCH p=(start:DataCenter)-[*6]-(end:Network:Zone)
RETURN count(p)
```

Figure 2: Queries for the experiment by groups: a) nodes counting – first group, b) searching for the shortest path – second group, c) number of paths counting – third group

It is worth to be noted that each database has a slightly different syntax in writing queries. Figure 2 shows queries for Neo4j. Memgraph queries syntax is very similar to Neo4j but ArangoDB has its own query language – AQL.

Figure 2-a depicts two queries that count how many nodes meeting given criteria there are in a database. It is important to note that for ArangoDB it was possible to write queries only of the first group due to its nature.

Figure 2-b depicts three queries which search for the shortest path from one node type to another.

Figure 2-c shows six queries which determine how many paths there are in exactly n-hops, where n is in the range from 1 to 6. It is also important to note that some combinations of nodes have too many variations of paths for Memgraph to find. Considering this fact, nodes with smaller number of paths were taken for final queries.

7. Results and discussion

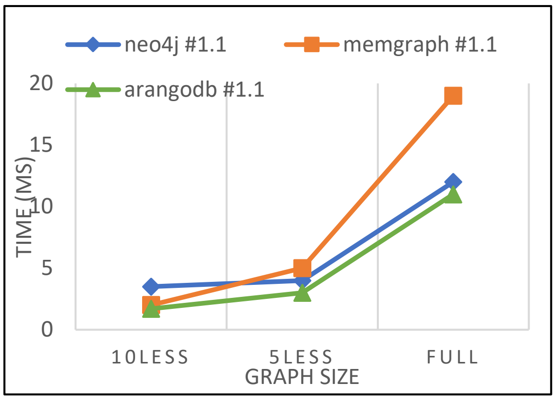
This section of the paper outlines obtained results in figures and graphical representation. Query execution time is represented in milliseconds. Section is divided into 4 subsections, three of them describe measurements of a separate group of queries specified above (subsection 7.1-7.3), and the last one introduces an estimation of the considered databases for the availability of support and the possibility of their use in various environments, which is an important criterion in choosing implementation tools for many projects.

7.1. First group of queries measurement results

First group of queries aims to count number of nodes which meet given criteria. Average query execution time in milliseconds is shown in Table 2, and Figure 3 depicts the dependence of the average query execution time on the size of the graph.

Table 2
Average query execution time for the first group of queries, in milliseconds

DB size \ DB type & query #	neo4j #1.1	neo4j #1.2	memg #1.1	memg #1.2	arango #1.1	arango #1.2
10less	3,5	3	2	2	1,7	0,7
5less	4	2,3	5	4,7	3	1
full	12	7	19	15	11	3,5

a) 

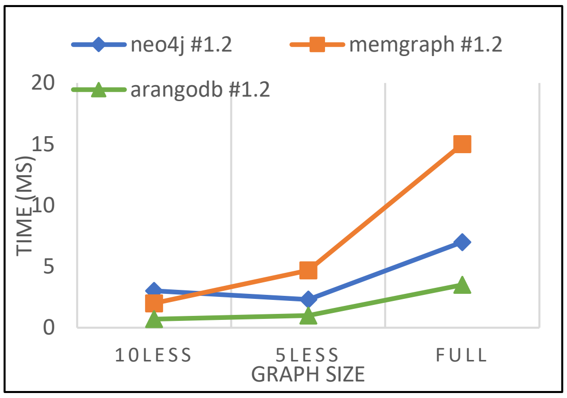
b) 

Figure 3: Dependency of the average query execution time on the graph size for query #1.1 a), query #1.2 b)

As can be seen, all databases showed similar results but Memgraph scaled slightly worse than other databases and ArangoDB showed the best results.

There won't be any representation of ArangoDB in further tables and at following line graphs as it wasn't possible to write such queries for this database.

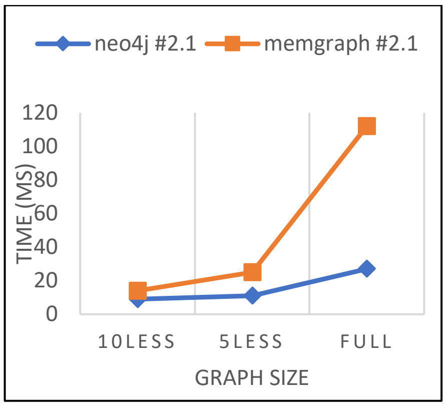
7.2. Second group of queries measurement results

Second group of queries aims to search for the shortest path from one node type to another. Average query execution time in milliseconds is shown in Table 3, and Figure 4 shows the dependence of the average query execution time on the size of the graph.

As can be seen, for this group of queries Neo4j showed better results, especially for the graph with a large number of nodes. For instance, for the largest considered graph, the average query execution time of query #2.1 in Memgraph is four times higher than in Neo4j, and for graphs with the number of nodes 16854 and 83847, it is near 2 and near 3 times higher respectively. The same pattern is observed with the other two queries. This indicates that on large graphs Memgraph works more slowly with this type of queries.

Table 3
Average query execution time for the second group of queries, in milliseconds

DB size \ DB type & query #	neo4j #2.1	neo4j #2.2	neo4j #2.3	memg #2.1	memg #2.2	memg #2.3
10less	9	10	37	14	37	112
5less	11	13	66	25	73	240
full	27	55	244	112	420	1480

a) 

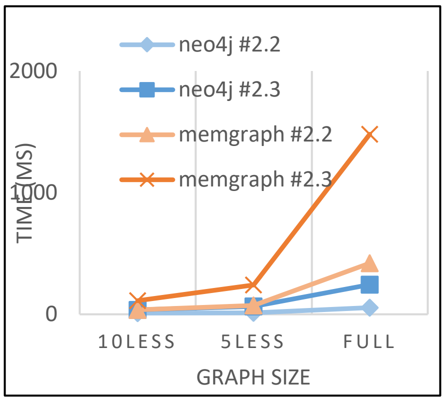
b) 

Figure 4: Dependency of the average query execution time on the graph size for query #2.1 a), queries #2.2 and #2.3 b)

It is important to note that the juxtaposition of the line graphs for the two other queries (queries #2.2 and #2.3) on the same scale is intended to show that both Memgraph queries (line graphs in shades of red) take longer to execute than both Neo4j queries (line graphs in shades of blue).

7.3. Third group of queries measurement results

Third group of queries aims to count how many paths there are in exactly n-hops from the certain node. Average query execution time in milliseconds is shown in Table 4, and Figure 5 depicts the dependence of the average query execution time on the size of the graph.

For this group of queries Neo4j and Memgraph showed similar results for queries #3.1, #3.3, #3.4 and #3.5.

Table 4
Average query execution time for the third group of queries, in milliseconds

DB type & query # / DB size	neo4j #3.1	neo4j #3.2	neo4j #3.3	neo4j #3.4	neo4j #3.5	memg #3.1	memg #3.2	memg #3.3	memg #3.4	memg #3.5
10less	4	8	20	94	18000	2	7	21	101	1010
5less	5	14	40	328	2857	6	15	43	311	3650
full	13	4398	270	4830	78280	20	228	222	5630	88800

For queries #3.1 and #3.5 Neo4j performed worse on the smallest graph size and better on the biggest. For the query #3.2 Neo4j performed much worse than Memgraph on the largest graph size.

To sum up, for the third group of queries, the performance of the considered databases in most cases was comparable, and the presence of fluctuations in some of them (query #3.2 on the full graph and query #3.5 on the 10less graph) requires further research.

7.4. Databases usability estimation

Faced with some difficulties while using the considered databases, it was decided to provide their usability estimation, taking into account various aspects of their practical application, support of programming languages, etc. Let us list the main criteria taken into account for aggregated indicator which is called usability factor:

- number of image downloads from the DockerHub platform. It reflects the demand and the prevalence of the database;

- number of users on the GitHub platform. It shows how many users are interested in the database improvement and may put an effort into its development;
- availability of user support from the database web page: value 1 if it is available on the web page and value 0 in another case;
- presence of an active community. Active community helps ameliorate the database which can be expressed in adding new features or drivers to support third-party tools, detection and elimination of vulnerabilities and errors. It is presented by a value 1 if an active community exists and a value 0 in another case;
- existence of images to deploy in the container. Containerization allows users to rapidly deploy applications and efficiently manage infrastructure. It is presented by a value 1 if it exists possibility of container deployment and value 0 in another case;
- availability to deploy on different cloud platforms, in particular, AWS, GCP and Azure are taken into account. More and more organizations hold their infrastructure in the cloud, which is why the possibility of database deployment in the cloud is significant. It is represented by a value from 0 to 3 according to the number of specified cloud platforms and the corresponding availability of instructions for deployment on them on the database web page;
- number of supported programming languages. Official libraries were taken into account.

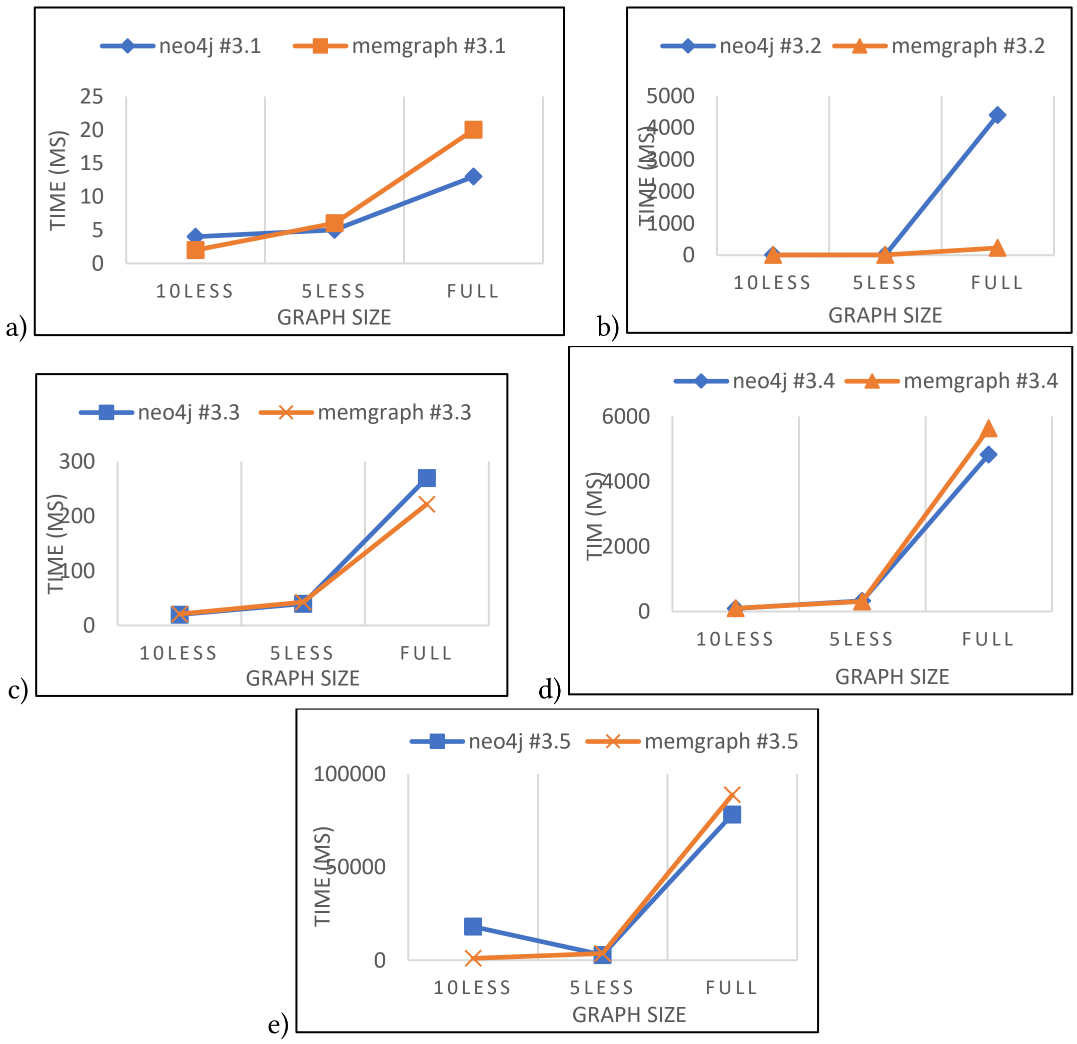


Figure 5: Dependency of the average query execution time on the graph size for query #3.1 a), query #3.2 b), query #3.3 c), query #3.4 d), query #3.5 e)

Table 5
Data for usability factor calculation

	Neo4j	Memgraph	ArangoDB
Years on the market	17	8	13
Number of image downloads	100M	100K	10M
Number of GitHub users	704	321	111
User support	1	1	0
Active community	1	1	1
Existence of images to deploy in container	1	1	1
Deployment on different cloud platform (AWS, GCP, Azure)	3	3	2
Number of supported programming languages	7	12	5
Usability factor	0.92	0.79	0.42

Table 5 contains data on the specified criteria for each of the databases. The data was collected from official online resources of databases, such as official web pages, GitHub repositories, etc. It is also worth noting that the table contains the number of years the database has been on the market, which is required for further calculation of the aggregated factor.

A following equation (1) was used to calculate usability factor

$$F_i = \left(\frac{1}{5} \cdot \frac{\frac{a_i}{y_i}}{\left(\frac{a}{y}\right)_{max}} + \frac{1}{5} \cdot \frac{\frac{b_i}{y_i}}{\left(\frac{b}{y}\right)_{max}} + \frac{1}{5} \cdot \frac{s_i + m_i}{2} + \frac{1}{5} \cdot \left(\frac{1}{2} c_i + \frac{1}{2} \cdot \frac{p_i}{3}\right) + \frac{1}{5} \cdot \frac{l_i}{l_{max}} \right), \qquad (1)$$

where a_i – number of image downloads of i-th database; y_i – number of years on the market of i-th database; b_i – number of GitHub users of i-th database; s_i – point of i-th database user support existence; m_i – point of active community i-th database existence; c_i – point of i-th database image existence to container deployment; p_i – number of i-th database supported cloud platforms; l_i – number of i-th database supported programming languages; $\left(\frac{a}{y}\right)_{max}$ – maximum value among all $\frac{a_i}{y_i}$; $\left(\frac{b}{y}\right)_{max}$ – maximum value among all $\frac{b_i}{y_i}$; l_{max}– maximum value among all l_i.

Obtained usability factors for considered databases are in Table 5. Figure 6 shows the contribution of each added to the overall usability factor value. It is important to note, that the third addend cumulates user support and active community points (Support in Figure 6) and the fourth addend cumulates database image existence to container deployment and normalized number of supported cloud platforms in equal parts (Deployment ability in Figure 6).

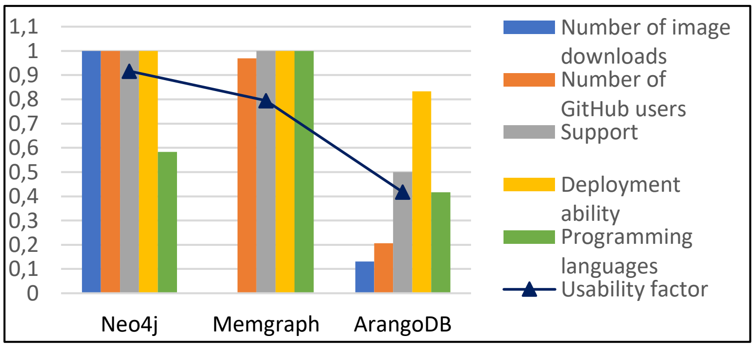


Figure 6: Contribution of each component to the overall usability factor value for Neo4j, Memgraph and ArangoDB databases

As can be seen, Neo4j has an advantage in most positions, which is caused on the one hand by a large number of supported features, and on the other hand by its widespread use among users.

8. Conclusions

During this study, three graph databases of various graph sizes were assessed through query execution time of 3 query groups. Experiments were conducted on the network management dataset and show that Neo4j demonstrates 2-4 times better performance for most queries compared to Memgraf. At the same time, its performance compared to ArangoDB was somewhat worse in the first group of queries. It was not possible to write the other two groups of queries in ArangoDB due to the peculiarities of its nature and the built-in query writing language. When performing the second group of queries, Neo4j showed significantly better results compared to Memgraph, especially for larger graphs, for instance, the average query #2.1 execution time in Memgraph is from 2 to 4 times higher than in Neo4j depending on the graph size. For the third group of queries, the results of Memgraph and Neo4j are comparable in almost all cases. However, it should be noted that the performance of Neo4j is slightly worse on the queries of the first and third groups when working with a graph of the smallest size. This may be due to the peculiarities of the internal implementation of Neo4j, as it is a NoSQL database by its nature.

Experienced some difficulties using ArangoDB, it was decided to explore databases' usability in more detail. Hence, they were estimated by the usability factor and Neo4j has the highest score 0.92. From the subjective point of view, it turned out, that Neo4j is the easiest database to work with. In addition to query complications, importing data into ArrangoDB requires splitting the file separately into nodes by node types and into relationships by their types, and running created files one by one distinctly. It slows down the import process and makes it confusing.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] Robinson, Ian, Jim Webber, and Emil Eifrem. Graph databases: new opportunities for connected data. " O'Reilly Media, Inc.", 2015.

[2] Cui, Hejie, et al. A Review on Knowledge Graphs for Healthcare: Resources, Applications, and Promises, 2023, doi:10.48550/arXiv.2306.04802.

[3] Henna, Shagufta, and Shyam Krishnan Kalliadan. Enterprise Analytics using Graph Database and Graph-based Deep Learning, 2021, doi:10.48550/arXiv.2108.02867.

[4] LI, Harry, et al. Knowledge graphs in practice: characterizing their users, challenges, and visualization opportunities. IEEE Transactions on Visualization and Computer Graphics, 2023, doi:10.1109/TVCG.2023.3326904.

[5] Besta, Maciej, et al. Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. ACM Computing Surveys 56.2, 2023, pp. 584–594. doi:10.1145/3604932.

[6] Jiang, Weiwei. Graph-based deep learning for communication networks: A survey. Computer Communications, 2022, doi:10.1016/j.comcom.2021.12.015.

[7] Tam, Prohim, et al. Graph neural networks for intelligent modelling in network management and orchestration: a survey on communications. Electronics, 2022, doi:10.3390/electronics11203371.

[8] Nicoara, Daniel, et al. Hermes: Dynamic Partitioning for Distributed Social Network Graph Databases. EDBT. 2015.

[9] Jamkhedkar, Pramod, et al. A graph database for a virtualized network infrastructure. Proceedings of the 2018 International Conference on Management of Data. 2018, doi.org/10.1145/3183713.3190653.

[10] Tseng, Vincent S., et al. Fraudetector: A graph-mining-based framework for fraudulent phone call detection. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2015, doi.org/10.1145/2783258.2788623.

[11] Angles, Renzo. A comparison of current graph database models. 2012 IEEE 28th International Conference on Data Engineering Workshops. IEEE, 2012, doi:10.1109/icdew.2012.31.

[12] Vicknair, Chad, et al. A comparison of a graph database and a relational database: a data provenance perspective. Proceedings of the 48th annual Southeast regional conference. 2010, doi:10.1145/1900008.1900067.

[13] Jouili, Salim, and Valentin Vansteenberghe. An empirical comparison of graph databases. 2013 International Conference on Social Computing. IEEE, 2013, doi:10.1109/SocialCom.2013.106.

[14] Kolomičenko, Vojtěch, Martin Svoboda, and Irena Holubová Mlýnková. Experimental comparison of graph databases. Proceedings of International Conference on Information Integration and Web-based Applications & Services. 2013, doi:10.1145/2539150.25391.

[15] Neo4j Closes Banner Year Marked by Customer Successes, Continued Industry Validation, Community Engagement, and Major Funding. URL: https://neo4j.com/pressreleases/2021-company-momentum/.

[16] Neo4j database. URL: https://neo4j.com/product/neo4j-graphdatabase/.

[17] Memgraph database. URL: https://memgraph.com/memgraphdb.

[18] Neo4j Network Management dataset on github. URL: https://github.com/neo4j-graphexamples/network-management.