Graph Databases

Krystyna Bajer, Anne Seidlitz, Sascha Steltgens, and Bastian Wormuth

1 Introduction

One major challenge for companies today is to analyze their data and gain knowledge and competitive advantage through understanding the relationships, correlations, and connections between different kinds of data. Graph databases are a powerful tool to find complex and dynamic relationships in highly connected data. Connected data is data whose interpretation and value require an understanding of the ways in which its elements are related (Robinson et al. 2013). But what is a graph database? It is a database that uses the graph structure and stores data as nodes, edges, and properties. Graph databases are based on graph theory, which is a mathematical construct on how to describe objects and the relationships between them (see Sect. 1.1).

In a relational database, entities are stored in tables and the relationship between entities is realized by joining tables with common keys. When representing many relationships, this representation can get complex and rigid. Graphs are simpler, they consist of nodes (*math.: vertices*), which have properties and labels. Nodes are connected by relationships (*math.: edges*), which have a type, usually a direction, and can also have properties. Edges represent an abstraction that is not directly implemented in a relational

K. Bajer (✉) · A. Seidlitz · S. Steltgens · B. Wormuth
ifb SE, Grünwald, Germany
e-mail: Krystyna.Bajer@ifb-group.com

V. Liermann and C. Stegmann (eds.), *The Digital Journey of Banking and Insurance, Volume III*, https://doi.org/10.1007/978-3-030-78821-6_3

database. Consequently, a graph database has no rigid table structures and is highly performant when querying complex relationships (Liermann and Tieben 2021). However, no uniform language like SQL (Structured Query Language) for relational databases exists to query data from a graph database. A popular language is Cypher, which is used by the Neo4j graph database. Typical providers and query languages are summarized in Table 1.

1.1 Mathematical Background

In the eighteenth century, Leonhard Euler introduced the basic idea of graph theory by solving the famous "Königsberg bridges problem", which subsequently led to the concept of an Eulerian graph (Wilson 2013). The city of Königsberg is separated by the River Pregel. The two islands were connected to each other and to the two mainland portions of the city by seven bridges. Euler proved that there is no path through all parts of the city that would cross each of those bridges once. This is a classic example of an optimization problem in graph theory. Another one is the "travelling salesman problem", i.e., to complete a circuit of the shortest length in a graph. The first textbook in this field was written by Dénes König in 1936 (König 1936).[1]

In graph theory, a simple graph $G = (N, E)$ is an ordered pair N and $E.N$ is a set of *nodes (math.: vertices)* and E is a set of *edges*. An edge $(n, n^{'})$ joins the nodes n and $n^{'}$. The nodes n and $n^{'}$ are *end vertices* of this edge and are *adjacent* to one another.

The most popular graph model variant is the property graph, which is a directed[2] labeled multigraph where the edges are directed, nodes and edges are labeled and can have properties, and there can be multiple edges between any two nodes. Properties are key/value pairs that represent metadata for nodes and edges (Fletcher et al. 2018). Figure 1 in Sect. 2.1 shows a data model and an explicit database example.

A single-labeled directed multigraph can be described as a tuple $G = (N, E, L, \delta, \lambda_N, \lambda_E)$, where N is a finite set of nodes, E is a finite set of edges, and L is a finite set of labels. The edge function $\delta : E \to N^2$ associates edges with pairs of nodes, $\lambda_N : N \to L$ is the node labeling function and $\lambda_E : E \to L$ is the edge labeling function. An edge $(x, y) \in E(G)$ is represented as a triple $(v, w, v^{'})$, where $v = \lambda_N(n)$, $w = \lambda_E(e)$, and $v^{'} = \lambda_N(n^{'})$ (Fletcher et al. 2018). A path ρ in a graph G is defined as a sequence of edges $(v_0, w_0, v_1), (v_1, w_1, v_2), \dots, (v_{m-1}, w_{m-1}, v_m)$, where

[1] For an English translation see König and Tutte, Theory of Finite and Infinite Graphs (2013).

[2] In a directed graph, the edges connecting two different nodes have different meanings, depending on their direction. In an undirected graph, an edge connecting two nodes has a single meaning.

Table 1 List of graph database technology providers (© ifb SE)

Company	Product	License	Query language	URL
Neo4j	Neo4j	Community Edition is GPLv3, commercial, and AGPLv3 options for enterprise and advanced editions	Cypher	www.neo4j.com
ArangoDB	ArangoDB	Free Apache 2, Proprietary	AQL	www.arangodb.com
OrientDB	OrientDB	Community Edition is Apache 2, Enterprise Edition is commercial	SQL-like	www.orientdb.com
Microsoft	Microsoft Azure Cosmos DB	Commercial	SQL-like	http://azure.microsoft.com/-services/-cosmos-db
Amazon	Amazon Neptune	Commercial	Gremlin, SPARQL	http://aws.amazon.com/neptune
IBM	IBM Graph	Commercial	SPARQL	https://www.ibm.com/uk-en/marketplace/graph
SAP	SAP HANA	Commercial	OpenCypher	www.sap.com
Oracle	Oracle Spatial and Graph	Proprietary	PGQL	https://www.oracle.com/

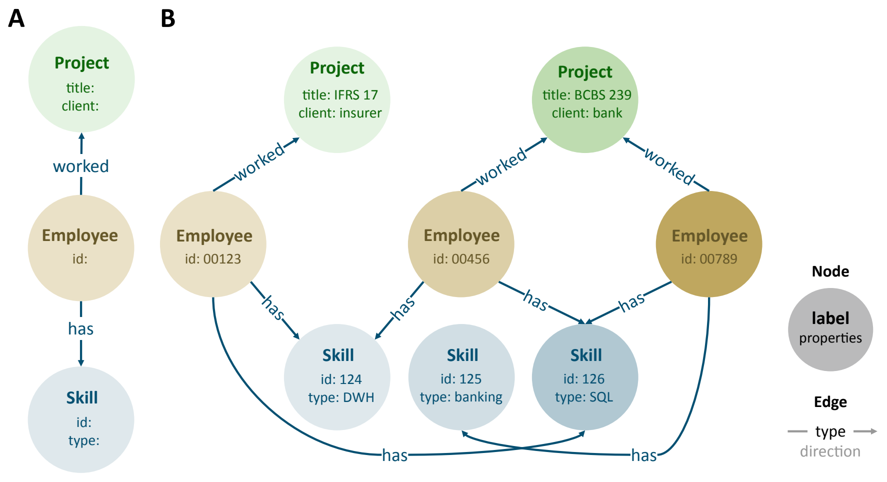

Fig. 1 Graph database model and example database (© ifb SE)

v_0 is the start node and v_m is the end note of the path. The label of ρ is the sequence of labels $w_1, w_2, \ldots, w_{m-1}$.

1.2 Graph Databases in Financial Services

Enterprises inside and outside of the financial sector want to analyze their data in an efficient way to gain the most possible knowledge and benefits. When the interesting information is in the relationship or connection in between data, for example, finding patterns indicating fraud, graph databases, and algorithms are an efficient tool for data analysis.

One prominent example is anti-money laundering. It is possible to model various entities involved, e.g., clients, accounts, transactions, and the relationship between these entities as a graph. By tracking how and where money is moving you can find patterns and detect money laundering behaviors. Section 4.1 discusses the "Panama Papers" as a special use case for detecting financial fraud in more detail.

Financial assets are very complex and substantial risks arising from mutual dependencies might be hidden, as the financial crisis in 2008 showed. To improve risk management, asset graphs can help to gain a better understanding of the relationships between assets and hence a clearer view of risks. They can also be used for real-time pricing of derivatives by considering the interdependencies among different assets, and therefore accurately reflect the risk and reward ratios (neo4j 2017).

A less obvious application for graph databases in the financial services sector is to handle information management with the help of a metadata graph. Data lineage and data flows are modeled as a graph for regulatory compliance and a complete picture of data and systems across the organization. It is possible to apply graphs for network and IT infrastructure monitoring to identify dependencies enabling network planning and impact and root cause analysis.

Other business areas where graph databases are already being used are master data management, identity and access management, and cybersecurity (Mathur 2020).

2 Technical Implementation

2.1 Data Model

Graph databases serve a large variety of purposes. In relational databases, which are the majority of databases not only in the financial sector, data is stored in normalized tables with columns and rows. Storing data in tables is viable for most types of data (especially data that is supposed to be aggregated or used to calculate KPIs) but may not be appropriate for other types of data occurring in the financial sector. If the data's main purpose is to be analyzed by its inherent connections, graph databases are the best choice. Whereas relational databases become inconvenient and slow if they are queried over a lot of database objects. Graph databases are built to support queries that span many database objects.

Nodes and edges have characteristics depending on the data model and information they are supposed to provide. The data model in Fig. 1A offers an impression of how data could be organized in a graph. In this case, the model defines characteristics for each database instance (nodes, edges). Nodes can have properties (key–value pairs) like IDs, a title, a type, other numeric values, etc., whereas edges have a type, a label, or any other numeric attribute[3] and also a direction.

Figure 1B shows a graph database of three employees (based on the data model in Fig. 1A), their skills, and the projects they worked in (nodes) as well as the employees' relationships (edges) to each type of node. It is imperative to the graph data which type of edges the data model allows. This example graph does not give any information about the relationships the nodes with type

[3] This refers to a weighted graph, where an edge can be weighted to qualify its weight or strength, e.g., cost, length, distance.

"Human" have with each other; those nodes are only indirectly connected. In order to display the relationships between the nodes of type "Human", it would be possible to add the edge of type "Colleague" to the data model, which would connect all three nodes of type "Human". A new type of node called "Employer" and a new edge of type "is employed" would, however, add more value than just the edge of type "Colleague". This is because the information "Employer" and "is employed" implies that all three nodes of type "Human" are colleagues, since they are employed at the same firm. But this new node and this new edge would additionally provide information about the employer.

A graph data model also carries information about the number of edges a single node of a particular type can have. In our example (Fig. 1B), the nodes of type "Human" can have multiple edges to nodes of type "Skill" and type "Project", but these cannot have any edge to any other type of nodes. If a "Project" node could have an edge to the "Human" type node, the graph database would increase in informational value, but it may lose its focus.

Obviously, the data model of any database is crucial for its value and purpose but in our example, the graph data model distinguishes a human-to-human relationship database from a skill- and experience-driven database. Hence, the data model of a graph database is imperative to its use case and small tweaks of the data model may add a disproportionate increase in information. This is because, in connected data, the nodes, whatever they may represent, are of less interest than the edges connecting the nodes. It is the edges that provide the interesting information. However, the bottom line is that, graph databases are a flexible storage technique for data that is highly connected and difficult to store in a relational model.

2.2 Storage

How graph databases are stored depends on the service provider. In general, a distinction is made between native and non-native storage of graph data. In a graph database where storage is native, each node and each edge are individual entities, somewhat comparable to object-oriented programming. In non-native storage graph databases, the data is stored in tables, introducing another layer of complexity to the interactions between database management system, database, and storage. Further, NoSQL storage types of graph databases include key–value storage or document-oriented databases. On a side note, most graph database providers utilize the ACID (atomicity, consistency, isolation, durability) set of properties for their database transactions. This ensures data validity despite errors, power failures, and other calamities.

2.3 Providers

The industry leader for graph databases is Neo4j (DB-Engines 2020), but Neo4j is most certainly not the only graph database technology provider. The company's competitors include Microsoft, Amazon, IBM, SAP, and Oracle, but also smaller firms like ArangoDB, OrientDB, Virtuoso, etc. Table 1 gives an overview of some of the technology providers.

2.4 Visualization of Graph Data

Unsurprisingly, visualization of graph data depends on the use case. A popular example is any navigation software, where information about the route to take is visualized on a map rather than presented as a table. To display graph data on a map leaflet.js (www.leafletjs.com) provides an open-source JavaScript library. Data representation on a map to visualize a route is obviously intuitive, as is the graphical display of nodes and edges corresponding to human interactions with anything. With these graphical views of graph data, it is easy to find highly connected nodes, recognize patterns, abnormalities, or other areas of interest. Providers like cytoscape.js (js.cytoscape.org), d3.js (d3js.org), neovis.js (for Neo4j graph databases only), and others offer software for graphical illustration of graph databases. Since a graph database can store any type of data, there are tools for visualizing graph data as charts, e.g., amCharts (amcharts.com), Chart.js (chartjs.org), and Tableau (tableau.com). Graph databases are also accessible with popular tools like R or Python. For R, Neo4j provides a package called neo4r including an API to access the Neo4j GraphDB with R. Several other packages help with visualization (e.g., visNetwork) of the data, making R a powerful and fully customizable tool to work with graph data in not only financial data science. For more information, please refer to (Liermann and Tieben 2021). Implementations for Python include NetworkX, a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks (Klein 2020).

All of the above mentioned tools offer a large variety of options to display graph data. However, everything depends on the use case. As an example, with tools like Tableau, it is possible to display financial fraud data by country/state (Barrasa 2016). Visualizing the data as such would allow executives to easily adjust products to locally higher financial risk. Displaying any travel-related number is not a challenge either, but visualization of queries with complex results becomes incomprehensible as a graphical view. Meaning that as soon as there are many nodes and edges to be displayed, a graphical view might be useful for developers or analysts but might not be the representation of choice for executives. Graph data analysisis therefore a double-edged sword: graph databases are fast and enable the user to find connections between nodes that are complex to find but visualizing these query results remains a challenge.

2.5 User-Friendly Approach to Graph Databases

For customers that do want to set up a graph database but do not want to start from scratch or with an empty, ETL tool-like feeling, vendors such as Structr (Structr GmbH 2020) provide a graph database framework to effortlessly get started with graph databases. Such tools deliver a codeless approach understandable for everyone to graph data modeling as well as data analysis with graphs.

3 Analysis of Graph Databases

The analysis of simple graph databases can be carried out with the help of queries or with algorithms for complex graph databases. Graph query languages serve this purpose. In this section, different queries are presented, which can be grouped into adjacency, pattern matching, reachability, and analytical queries. In the analytical queries, algorithms are discussed in more detail, whereby three selected algorithms are presented. In the last section, a selection of graph query languages are introduced with a description and a note of which graph model they are applied to.

Adjacency Queries

The adjacent query refers to both edges and nodes. For example, in case of an edge between two nodes, these two nodes are adjacent. Equally, if two edges have a common node, these edges are adjacent. Frequently asked and investigated subjects would be whether node v is the neighbor of node v' or who is the adjacent node to node v. Although these queries seem simple at first glance, they can be a challenge for sparse graphs.

To be able to define more complex neighborhood queries, the adjacency term can be extended (for more information Fletcher et al. [2018]).

For the adjacent query, it is possible to use an adjacency matrix. This matrix reflects relationships between the individual nodes. Each node has one row and one column, with n nodes a $n \times n$ matrix is created. In a simple graph, without edge weights and multiple edges, this matrix is a (0,1)-matrix with zeros on its diagonal. Due to the fact that this problem can be represented as a matrix, it is possible to apply methods of linear algebra.

Adjacency queries are particularly important in issues of influence. They are used to find individuals with similar interests (recommendation systems) or to obtain information. The adjacency matrix may further be used to calculate the path length in graphs. For example, assume a directed graph $G = (V, E)$ without edge weights or multiple edges with an adjacency matrix A. Then, the path length between node i and node j, which contains k edges, can be calculated by the power of the adjacency matrix A^k. Using A^k, it can be determined in row i and column j how many paths meet the requirement (Wikipedia 2020).

Pattern Matching Queries

This query searches a graph database for the set of all subgraphs that match a specific graph pattern. A simple example of such a searched graph pattern would be a small graph in which the edges and nodes are labeled by variables. The variables indicate unknown data and define the output of the query. In other words, values are assigned to these variables at the end of the query. An example would be the search for the colleagues $?x$, $?y$ of M1, who are also colleagues (Fletcher et al. 2018).

$$(M1, colleague, ?x), (M1, colleague, ?y), (?x, colleague, ?y)$$

where $?x$ and $?y$ are variables.

Pattern matching is used in areas such as within the pattern recognition field, to identify communities and social positions in social networks and in protein interaction networks.

Reachability Queries

The task of this query is to evaluate whether given nodes are connected by paths. They are also modeled as path or traversal problems in connection with graph databases, which allow restrictions by nodes and edges.

The areas in which the reachability queries are used range from social networks, discovering people with common interests, to biochemistry, to find specific biochemical paths between distant nodes. They are also used as a basis for the shortest path analysis.

Analytical Queries

Analytical queries measure quantitatively and usually in aggregated form topological features of the graph's database. They are supported either by ad hoc functions that hide complex algorithms or by special operators. Simple analytical queries include, for example, aggregate operators of query languages, such as i and max. These can be used to determine the number of nodes, the number of neighbors of a node (degree of a node), the length of a path, or the shortest path between two nodes. Algorithms are used for more complex analytical queries. In the following, a selection of such algorithms is presented (for more examples see (Hunger and Augsten 2020).

- **Shortest Path** calculates the shortest path of a node and all other nodes in a graph. Another name for the Shortest Path algorithm is the Dijkstra algorithm. It received this name from its developer, the Dutch computer scientist Edsger Dijkstra. This algorithm is used among others in Google Maps. In general, cities represent nodes and edges represent the road network. The edges of such a graph can also distinguish between motorway roads, country roads, and roads with toll costs. Thus, not only the shortest route but also the fastest or most cost-effective route can be determined.
- **PageRank** measures the importance of nodes in a graph. The importance of a node is measured by how many indirect and direct relationships point to a node. The more relations a node has, the higher its weight and the greater its effect. Google is famous for this algorithm. There, the search results are sorted according to their accumulated importance. Another example of the use of the PageRank algorithm can be found on Twitter and in biology to predict chain reactions within an ecological system.
- **Betweenness Centrality** is similar to the Shortest Path algorithm. It measures the number of shortest paths in a graph that run from a node. If a node is most often on the shortest paths, it forms a bridge between clusters and has a higher centrality value. Such nodes have a high influence on the information flow in a graph. This algorithm is mainly used for network analysis and fraud detection.

Many providers such as Neo4j, Orient, and Palantir offer tools for various applications, which take over the task of graph analysis. Neo4j has, for example, developed a tool (Privacy shield) which complies with the requirements of the European Union General Data Protection Regulation or a tool (Fraud Detection) for banks, insurance companies, and e-commerce to detect fraud (Sadowski and Rathle 2015).

3.1 Graph Query Languages

In comparison to relational databases, graph databases do not have a uniform language. This is because graph databases have no predefined standard. Over time, different query languages for different databases have developed (Fletcher et al. 2018).

Data query languages can be separated into the following groups: languages for edge-labeled graphs, languages for hypergraphs, languages for nested graphs, languages for property graphs, and RDF query language.

The most commonly used is Cypher, which is the query language of Neo4j and belongs to the category of property graphs. The easiest query in Cypher involves an expression with a clause START, MATCH, and RETURN, which can be used as follows:

```
START x=node:Employee(name="Name")
MATCH (x)-[:colleague]->(y)
RETURN y.name
```

In the example above, the name of a colleague of x is sought. The `START` command defines the node of the graph that is the start point. `Match` is used to find the pattern and `RETURN` specifies what the query should deliver. Further, Cypher can calculate paths from node `(a)` to node`(b)`. In the following example, only outgoing edges are recognized and the solution is stored in the path variable `p`.

```
p=(a)-[:knows*]->(b)
```

In addition, Cypher can compute specific operations on nodes, edges, attributes, and paths using built-in functions. In the example above, the shortest way from node `(a)` to `(b)` could be determined by the function `shortest Path(p)`. For more information about Cypher, please refer to (Liermann and Tieben 2021).

Table 2 contains additional graph query languageswith a description and grouping.

4 Business Use Cases

While Sect. 1.2 gave a rough overview of use cases for graph databases in the financial sector, this section focuses on the description of three explicit business use cases.

Table 2 Additional graph query languages with a description and grouping (© ifb SE)

Language	Description	Group
Gremlin	Open source-based query language for various graph databases (Neo4j, OrientDB, or DEX)	Languages for property graphs
SPARQL	W3C specified query language for RDF data models	RDF query languages
Blueprints	One of the first libraries created for the property graphs usable for different graph databases	Languages for property graphs
GraphQL	SQL-like query language	Languages for property graphs
Rexster	Multi-supported HTTP/REST interface for querying graph databases over the Internet	

4.1 Fraud Detection—Panama Papers

In 2016, the International Consortium of Investigative Journalists (ICIJ) exposed a highly connected network of illicit offshore bank accounts by analyzing a data leak of 2.6 terabytes consisting of 11.5 million leaked files from the Panamanian law firm Mossack Fonseca.

They used a graph database consisting of 840,000 nodes and 1.3 million relationships. The graph contains key entities such as “Company”, “Client”, “Intermediary”, “Address”, and “Officer” to reveal connections between these nodes. All these entities have a lot of properties, such as document numbers, share amounts, source ID, addresses, and citizenship. These entities have specific relationships, for example, an officer is the “director of” or “shareholder of” a company or has “similar name and address as” the intermediary. In this way, you can find suspicious relationships, such as companies that control other companies in the same country through a company in an offshore zone. An example is “Regula Limited”, an offshore legal entity registered in the Bahamas and the British Virgin Islands which was a subsidiary of Deutsche Bank. By querying the node(s) “Regula Limited” and all the companies it is connected to, you can see that this entity serves as an officer of several other offshore legal entities, and also served as the “intermediary” for several offshore entities.

By analyzing the connections between the nodes, ICIJ revealed that roughly 500 banks had registered nearly 16,000 shell companies for Mossack Fonseca clients. The British banking giant HSBC and its subsidiaries alone account for more than 2300 of the companies (Woodman 2016).

4.2 Lufthansa—In-Flight Entertainment System Management

An unexpected use of graph databases is applied by Lufthansa's in-flight entertainment management system (see Wilmes 2013 for more information). In order to provide customers with in-flight entertainment, a plane carries storage and broadcast units. To update the entertainment content, a switch of storage devices (HDDs or SDDs) is necessary. How can storage devices, the planes carrying them as well as the planes' current location be quickly identified in order to prevent a movie from being shown to customers? Or how can movies be controlled if all necessary licenses for a freshly installed movie are present? For this task, Lufthansa chose a graph database to manage its in-flight digital assets and its licenses and build a dashboard for the ground personnel to quickly identify and replace a storage device while any plane is at an airport.

4.3 Navigation Systems

Navigation systems, such as Google Maps and many more, use different algorithms to guide from a starting point to an end point in the best possible way. Among others, the shortest path algorithm is used, like the Dijkstra algorithm (see Sect. 3) and variants of it. This segment shows how a navigation system works using the Dijkstra algorithm. Assuming that a city is referred to in this example, nodes are intersections and edges are roads. At the nodes are the points where you can choose a path (edge). It is important to note that not all edges are the same. When deciding which road to use, you choose the one that will lead you to your destination first. Several factors are considered in this decision, such as traffic lights, the size of the road, traffic volume, speed limits, and so on. To illustrate this in the graph, weights are assigned to the edges, which reflects the estimated travel time of this section. If the edge between node X and node Y is described with the number 4, the estimated distance between X and Y on this path takes four minutes. With the help of the Dijkstra algorithm the "minimum distance" is determined. Meaning that if a start and end point is given with weighted edges in the graph, the sum of the weights, in other words the travel time, is minimized, which is the time between start and end point (Moussa 2020; Sanders et al. 2007; Crovari 2019).

The algorithm works as follows. Each node has assigned "costs". This cost is the value of the minimum path to reach this node. Before the algorithm starts, each node has an infinite cost, as no path has been found to reach this node yet. The start node has the cost of zero; as you are already at the start there are no costs to get there. Furthermore, the algorithm has a list of all possible nodes through which the end point can be reached. The algorithm now repeats a certain process, first the one with the lowest cost is extracted from the list of nodes to be visited. The nodes that have not yet been visited but are also accessible are assigned costs. If the cost of this node is lower, the cost of the "cheaper" node is selected, otherwise the cost of the first node is kept. The visited node is marked as visited and the process starts from scratch until the destination is reached. Mathematically, it has been shown that the Dijkstra algorithm always finds the shortest route, including navigating over roads that would otherwise never have been considered. Information such as the traffic situation and traffic jams can be included in the calculation by adjusting the weights.[4]

5 Summary

We observe not only the establishment of specific software and consulting companies for graph technologies as relevant contributors to the IT landscape, but also major software players like IBM, SAP, and Oracle adding graph technologies to their standard portfolio. The constantly growing amount of information, often highly connected data from social networks or similar, increases the demand for tools to support the efficient storage and analysis of such data sets. Therefore, graph technologies are here to stay and will remain an important piece of equipment for the modern data scientist.

Literature

Barrasa, Jesús. 2016. *Graph DB +Data Virtualization =Live Dashboard for Fraud Analysis*, November 30. Accessed November 5, 2020. https://jbarrasa.com/2016/11/30/graph-db-data-virtualization-live-dashboard-for-fraud-analysis/.

Crovari, Pietro. 2019. *GOOGLE MAPS AND GRAPH THEORY.* Impactscool Magazine, May 20. Accessed November 9, 2020. https://magazine.impactscool.com/en/speciali/google-maps-e-la-teoria-dei-grafi/.

DB-Engines. 2020. *Ranking.* solid IT gmbh. Accessed November 6, 2020. https://db-engines.com/en/ranking.

[4] Example of Dijkstra algorithm: https://www.youtube.com/watch?v=UG7VmPWkJmA&feature=youtu.be&t=33.

Fletcher, George, Jan Hidders, and Josep Lluis Larriba-Pey. 2018. *Graph Data Management.* Springer.

Hunger, Michael, and Stephan Augsten. 2020. *Graph-Analytik.* Dev Insider, September 21. Accessed November 6, 2020. https://www.dev-insider.de/5-wichtige-graph-algorithmen-im-ueberblick-a-963343/.

Klein, Bernd. 2020. *Python Course.* Accessed December 4, 2020. https://www.python-course.eu/networkx.php.

König, Dénes. 1936. *Theorie Der Endlichen und Unendlichen Graphen: Kombinatorische Topologie Der Streckenkomplexe.* Chelsea.

König, Dénes, and W. T. Tutte. 2013. *Theory of Finite and Infinite Graphs.* Translated by Richard McCoart. Springer Science & Business Media.

Liermann, Volker, and Marian Tieben. 2021. "Use Case—NFR—Using GraphDB for Impact Graphs." In *The Digital Journey of Banking and Insurance, Volume II—Digitalization and Machine Learning*, edited by Volker Liermann and Claus Stegmann. New York: Palgrave Macmillan.

Mathur, Nav. 2020. "Graph Technology for Financial Services." *neo4j.* Accessed November 12, 2020. https://neo4j.com/whitepapers/financial-services-neo4j/.

Moussa, Ramy. 2020. *How Google Maps Work: Fast Route Planning*, February 15. Accessed November 11, 2020. https://algorithmyou.com/2020/02/15/artificial-intelligence/how-google-maps-work-fast-route-planning/.

neo4j. 2017. *neo4j*, May 24. Accessed November 11, 2020. https://neo4j.com/blog/financial-services-neo4j-financial-asset-graphs/.

Robinson, Ian, Jim Webber, and Emil Eifrem. 2013. *Graph Databases.* O'Reilly Media, Inc.

Sadowski, Gorka, and Philip Rathle. 2015. *Fraud Detection Discovering Connections with Graph Database Technology.* Neo4J, January. Accessed December 16, 2020. https://neo4j.com/whitepapers/white-paper-fraud-detection/.

Sanders, Peter, Schultes, and Dominik. 2007. "Engineering Fast Route Planning Algorithms." *algo2.iti.kit.edu.* Springer-Verlag Berlin Heidelberg. Accessed December 17, 2020. http://algo2.iti.kit.edu/documents/routeplanning/weaOverview.pdf.

Structr GmbH. 2020. *Structr.* Accessed December 15, 2020. www.structr.com.

Wikipedia. 2020. "Adjazenzmatrix." *Wikipedia*, June 10. Accessed November 4, 2020. https://de.wikipedia.org/wiki/Adjazenzmatrix.

Wilmes, Michael. 2013. "Slideshare." *slideshare.net.* Accessed November 7, 2020. https://de.slideshare.net/neo4j/inflight-asset-management-with-neo4j-michael-wilmes-graphconnect-london-2013.

Wilson, Robin J. 2013. "History of Graph Theory." In *Handbook of Graph Theory*, edited by Jonathan L. Gross, Jay Yellen and Ping Zhang. CRC Press.

Woodman, Spencer. 2016. *Global Banks Team with Law Firms to Help the Wealthy Hide Assets.* Accessed November 9, 2020. https://www.icij.org/investigations/panama-papers/20160404-banks-lawyers-hide-assets/.