Querying in the Age of Graph Databases and Knowledge Graphs

Marcelo Arenas
marenas@ing.puc.cl
Universidad Católica & IMFD
Chile

Claudio Gutierrez
cgutierr@dcc.uchile.cl
DCC, Universidad de Chile & IMFD
Chile

Juan F. Sequeda
juan@data.world
data.world
USA

ABSTRACT

Graphs have become the best way we know of representing knowledge. The computing community has investigated and developed the support for managing graphs by means of digital technology. Graph databases and knowledge graphs surface as the most successful solutions to this program. This tutorial will provide a conceptual map of the data management tasks underlying these developments, paying particular attention to data models and query languages for graphs.

CCS CONCEPTS

• **Information systems** → **Graph-based database models**; • **Computing methodologies** → *Knowledge representation and reasoning*.

KEYWORDS

Graph databases; knowledge graphs; data models; querying

ACM Reference Format:
Marcelo Arenas, Claudio Gutierrez, and Juan F. Sequeda. 2021. Querying in the Age of Graph Databases and Knowledge Graphs. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3448016.3457545

1 INTRODUCTION

What does it mean to query a graph? What does it mean querying graph models? Any answers to these questions should try to understand the role of graphs as a conceptual tool to model data, information and knowledge. Graphs have a long tradition as medium of representation, and an impressive wide range of uses. Let us recall some highlights related to our area: underlying data structures (the hierarchical and networks database systems of the sixties [64]); semantic networks; graph neural networks; entity relationship model; XML; graph databases; the Web as universal network of information (and later of data and knowledge); and knowledge graphs. This non-exhaustive list indicates at least that some reflection is necessary before addressing the goal of this tutorial

Following usual practices, we performed an initial examination of what is being researched disciplinary in the area of data, graphs and knowledge. We explored five of the most salient keywords that today represent research around this area: "graph database"; "RDF"; "SPARQL"; "property graph", "knowledge graph", by analyzing papers in computer science (those indexed by DBLP)[1] having these strings in their titles.[2] Figure 1 shows the evolution of the number of publications of papers with these keyword in the title from 2010 to 2020.

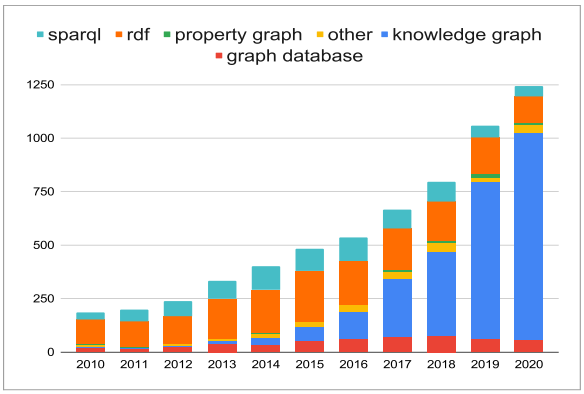


Figure 1: Number of knowledge graph related publications. Source: DBLP.

In this preliminary exploration we can observe the following. The growth of "knowledge graph" papers can be seen starting in 2013, which correlates with the year after the Google's Knowledge Graph announcement. The amount of publications about "RDF" and "SPARQL" continue to be stable. However we observe a decline when compared to "knowledge graph". In 2015, 70% of knowledge graphs papers were about RDF/SPARQL, while that went down to 14% in 2020. Papers about "graph database" are comparatively small and there is no significant growth, while papers about "property graph" are negligible. The main takeaway message from this seems to be that publications about knowledge graphs are significantly increasing and in some sense "dominate" the area. Thus, when addressing graphs as a model for data to knowledge, we cannot ignore the obvious knowledge graph hype.

This poses the question: What are knowledge graphs and what is their relation to graph databases? In addressing this question we should be cautions about two extremes: On one hand, as Jeffrey Ullman wrote, avoid to get "engaged in hand-wringing over the idea that we [the database discipline] are becoming irrelevant" [66]. On the other hand, try to understand if there is someting really new under this new hype about knowledge graphs. Our preliminary hypothesis –one that we will follow here– is that this rather

[1]This includes all types of publications indexed by DBLP.
[2]Data: https://data.world/juansequeda/dblp-knowledge-graph

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457545

vague notion with no clear borders (see e.g. Appendix A in [39]) encompasses a great variety of methods and practices dealing with data, information and knowledge, orbiting around the gravitation center of graph models.

Our intention in this brief overview is to try to convey a rough cartography of what is means to query in this new scenario, and instill in the audience some doubts and reflections we have about the development of our area as a whole. We feel that this reflection is more important than ever today, when big data, deep learning and other trends of the beginning of the 21st century have shaken computing as we know it.

More concretely, the goal of this tutorial is to provide in Section 3 a unified and simple view of the data models behind graph databases and knowledge graphs, and show some recently established results on querying such graphs. More specifically, we focus in Section 4 on the fundamental task of extracting knowledge from graphs in the form of nodes and paths satisfying a pattern, and we study new paradigms on path extraction, the inclusion of knowledge in some graph analytic tasks, and the connection of declarative with procedural frameworks for node extraction.

But before approaching this goal, we considered necessary in Section 2 to establish a general picture of the area in order to better organize the conceptual map of the wide diversity of existing methods and techniques.

2 DATABASES, GRAPHS AND KNOWLEDGE GRAPHS

Before going to technicalities, we will present a conceptual overview of the main notions of the area and how they interplay. Our goal is to contribute to a necessary discussion of the reasons why graph have become so prominent in data and knowledge management.

2.1 Graph data and querying

Data is the raw material of our area. Databases originated in the need to store, keep safe and private, organize and operate in a efficient, reliable and permanent form, big quantities of data in computers. From these basic and essential tasks, it was developed what is probably the most important functionality of databases, namely querying, that boils down to friendly, expressive, and efficient languages for defining, updating, transforming and extracting data. That is why query languages play such a relevant role in our discipline.

One of the landmark advances in the field was the notion of "data independence" [23]. As computing is essentially about communicating humans and machines, or better, human knowledge and machine operation, for methodological and practical reasons one should separate the physical level (the one revolving around the machine and data) from the logical level (revolving around the way humans model reality and knowledge). Our working hypothesis is that graphs are an appropriate way of representing, both, data and knowledge. And this would be the reason why graphs began to be so prominent in this field of data and knowledge management.

Graphs have a simple data structure consisting of nodes and edges, which has the nice operational properties of expressing relations, presenting data in a rather holistic way (neither ordered nor sequentially), and last but not least, having a flexible structure that permits growing and shrinking (adding/deleting nodes and edges) and integration (of different graphs) in a natural way.

Over this nice data structure, floats a similarly nice group of conceptual ideas. First, entities, represented by nodes; second, connectivity, represented by edges and paths; and third, emergent (global) properties that the structure produces. Then it comes as no surprise that languages for querying graphs deal with extracting information about these three main features (we will discuss them in detail in Section 4): (i) local properties (nodes and neighborhoods), where pattern matching and its extensions play a key role, being usually approached with logical methods; (ii) connectivity, where paths and more complicated structures need to be extracted, usually by means of a mixture of regular expressions and logical methods; and (iii) global properties that need essentially different methods and approaches than those of (i) and (ii), and which are usually put under the graph analytics umbrella.

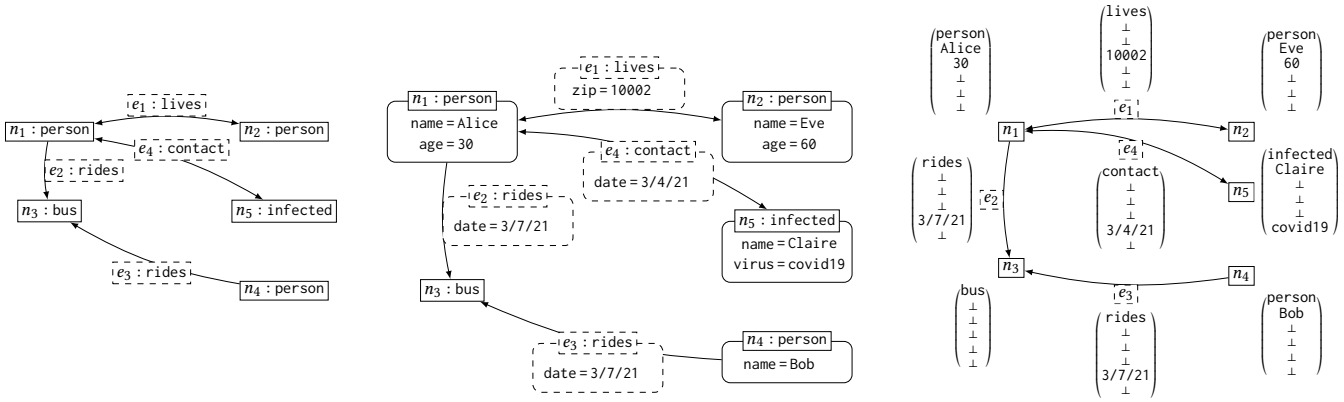
2.2 Some paradigmatic examples

To get a flavor of how these structures and ideas work in practice, let us briefly review three paradigmatic examples.

The first attempt to represent knowledge in the form of graphs was the notion of *Semantic Network* [54, 55, 57]. Although first purely graphic, researchers formalized it using logical methods. Graphs (called networks in this field) are used because they are good objects to represent knowledge. Besides being a bridge to visualization of knowledge, semantic networks have the feature (shared with graphs in general) of highlighting and facilitating the discovery and representation of relationships.

Classical relational databases are flexible enough to represent a graph, e.g. by a two attribute relation storing its edges. In this representation, nodes are entries and paths are constructed by successive joins. Why then do we need *graph databases*? There are at least two reasons: joins are expensive and thus, reasoning about paths becomes very costly. Furthermore, global properties are no easy to compute with classical queries based on logical methods. Hence, it comes as no surprise that people have tried since the early days of databases to develop graph databases. First at the hardware level with the hierarchical and network models of the sixties [64]; then at a more logical level in the eighties where we found the "golden years" of graph databases [7, 11, 63]. However, many reasons, among them the limitations of hardware and software, did not permit the popularization of these systems [7].

The Web is the first comprehensive system for representing, integrating and "producing" knowledge at big scale, whose key idea is taking advantage of the structure and features of the graph model. The Web was originally thought as a universal network, that is, nodes and edge were designed to be directly interpreted by humans. However, due to its scale, soon emerged the need to enrich the network to allow automation of functionalities. This is the origin of the *semantic Web*, where semantics means "understandable by machines" [18]. The semantic Web contained the two ideas that were to originate the current notion of knowledge graph: a graph structure to organize the data; and a system which encodes not only information (i.e. data to be interpreted directly by humans), but it is also organized to derive new facts from the current ones, that is, deals essentially with knowledge. However, the universality of



(a) A labeled graph (b) A property graph (c) A vector-labeled graph that represents the property graph in (b)

Figure 2: Three graph data models.

the semantic Web became a problem for private organizations, due to privacy and property rights concerns. Google overcame this by developing the notion of knowledge graph as a "finite", manageable, controlled and usually private Semantic Web. Nevertheless, the main concepts, namely, ontologies to integrate knowledge, and the Web as medium, would become almost standards.

2.3 Knowledge Graphs

The cases of semantic networks, graph databases and the semantic Web point to systems in which graphs supports the representation, integration and production of knowledge. This rather fuzzy idea is what is at the core of the notion of knowledge graph, so it can be defined as a software object (artifact) that represents (codifies), integrates and produces knowledge. In order to perform these functionalities, knowledge graphs rely on the model of graphs, so they encompass many previous developments [26, 35].

Knowledge representation is incorporated mainly trough standardized languages, metadata and ontologies on one hand, and semantic networks and other non-formal language representations on the other. Lately, low-level representations like vector-labeled graphs were added. Notice that all of them use the flexibility of graphs for model representation (see Section 3). *Integration of knowledge* is achieved essentially via the aforementioned features of graph extensibility and integration (assuming a good and standardized representation like an ontology). Thus, a knowledge graph is capable of integrating knowledge from different sources, by linking or materializing them in one place, and in widely different formats.

Finally, *producing new knowledge* is probably the main "added-value" as opposed to classical repositories of knowledge. Knowledge graphs have the capabilities of: deducing, e.g. by means of logical reasoners or neural networks; linking, that is, relating different pieces of knowledge beforehand isolated; learning, through new data and learning algorithms; and of course, generating new knowledge by human intervention through refined ways of querying them (see section 4). This new knowledge is not only user-oriented, but is utilized in parallel to "complete" and enrich the knowledge already present. As a clear example of this, we see the rapid development of knowledge graph embeddings [19, 21], and its use in the refinement and completion of knowledge graphs [36, 43, 52, 56].

These features make a knowledge graph a highly multifaceted software object. One of the best examples of this is Wikipedia. It is a repository that represents knowledge in the form of a large graph (implemented via Web link protocols), with the capability of integrating knowledge from different sources in different formats. Moreover, Wikipedia has a mechanism to "extract" or "produce" knowledge. In this case, these are the multiple interfaces, most of them human interfaces. Wikipedia is a knowledge graph oriented toward a final human user, that is, it does not (or at least was not conceived to) feed another software systems. DBpedia [12] and Wikidata [69] are derived systems oriented to supply this facet.

3 GRAPH DATA MODELS

In this section, we give a simple unifying view of the most popular graph data models, from the simplest ones used in graph databases to the models that have emerged to store, integrate and produce knowledge.

The following are two basic ingredients to define graph data models. Assume that **Const** is a set of constants, or strings, that can be used for different purposes, for example as node identifiers, edge identifiers, labels, property names or actual values (such as integers, real values or dates). Moreover, define a multigraph as a graph where multiple edges can connect two nodes, that is, a tuple (N, E, ρ) where $N \subseteq$ **Const** is a set of nodes, $E \subseteq$ **Const** is a set of edges and $\rho : E \to N \times N$ is a function indicating the starting and ending node of each edge.

As a first data model, we consider labeled graphs, which are a popular and simple way to represent semi-structured data. Formally, a labeled graph is a tuple $\mathcal{L} = (N, E, \rho, \lambda)$ where (N, E, ρ) is a multigraph and $\lambda : (N \cup E) \to$ **Const** is a function indicating the label of each node and edge. Such graphs have been called heterogeneous graphs in the literature [39, 65], as opposed to edge-labeled graphs where labels are only associated to edges [6]. But here we prefer the simple term labeled graph to indicate that both

nodes and edges are labeled. An example of such a graph storing information about people and their contacts is shown in Figure 2(a).

It is worth saying a few words about RDF [24], a class of labeled graphs that is widely used in practice. A first characteristic that distinguish RDF graphs is that edges are replaced by triples, and they are not assigned identifiers. Formally, an RDF graph is a set of triples (s, p, o) such that $s, p, o \in \mathbf{Const}$, so that (s, p, o) represents an edge from s to o with label p. A second important feature of RDF graphs is that **Const** is considered as a set of Uniform Resource Identifiers (URIs [17, 25]), that can be used to identify any resource used by Web technologies. In this way, RDF graphs have a universal interpretation: if $c \in \mathbf{Const}$ is used in two different RDF graphs, then c is considered to represent the same element.

As a second model we consider property graphs, which are widely used in graph databases [28, 49, 59, 67]. Property graphs are defined as the extension of labeled graphs where nodes and edges can have values for some properties. Formally, a property graph is a tuple $\mathcal{P} = (N, E, \rho, \lambda, \sigma)$ where (N, E, ρ, λ) is a labeled graph, and $\sigma : (N \cup E) \times \mathbf{Const} \to \mathbf{Const}$ is a partial function such that if $\sigma(o, p) = v$, then v is said to be the value of property p for object o. Besides, it is assumed that each node or edge in $\mathcal{P}$ has values for a finite number of properties [5, 6, 49]. In Figure 2(b), we show an example of a property graph that extends the labeled graph in Figure 2(a), including as properties the name and age of a person, the zip code of the address for two people that live together, the date when someone rides a bus, and the date a contact between two people occurs.

As a final model, vector-labeled graphs are defined in a way that unifies the use of labels and properties, and allows to include in a simple way extra values that are necessary for message-passing graph algorithms [42], such as the Weisfeiler-Lehman graph isomorphism test [33, 34, 70], and when graphs are used as input of graph neural networks [48, 60]. Formally, a vector-labeled graph of dimension d, with $d \geq 1$, is a tuple $\mathcal{V} = (N, E, \rho, \lambda)$ where (N, E, ρ) is a multigraph and $\lambda : (N \cup E) \to \mathbf{Const}^d$ is a function that assigns a vector of values to each node and edge in the graph [39], which is called a vector of features of dimension d. Hence, labels and properties are replaced by vectors of values from **Const** in vector-labeled graphs, as shown in Figure 2(c). In this figure, the string $\perp$ is used to represent the fact that a row in a vector does not have a value.

4 QUERY FUNCTIONALITIES

In this section we will present some recently obtained results on querying graphs. We will focus on the fundamental task of extracting knowledge from graphs in the form of nodes and paths satisfying a pattern, and study new paradigms on path extraction, the inclusion of knowledge in some graph analytic tasks, and the connection of declarative with procedural frameworks for node extraction.

As mentioned before, extracting nodes and paths is a fundamental task when retrieving knowledge from graphs [6, 28]. Regular expressions form the core of such an extraction task, so before going into the details of the results shown in this section, we formalize the notion of regular expression for the data models presented in Section 3. More precisely, a regular expression over a labeled graph $\mathcal{L} = (N, E, \rho, \lambda)$ is given by the following grammar:

$$\begin{aligned} \mathit{test} &::= \ell \mid (\neg \mathit{test}) \mid (\mathit{test} \vee \mathit{test}) \mid (\mathit{test} \wedge \mathit{test}) \\ r &::= ?\mathit{test} \mid \mathit{test} \mid \mathit{test}^- \mid (r + r) \mid (r/r) \mid (r^*), \end{aligned} \quad (1)$$

where ℓ is a node or edge label in $\mathcal{L}$. An answer to r over $\mathcal{L}$ is a path whose labels conform to r. Formally, such a path is a sequence $p = n_0 e_1 n_1 e_2 \cdots e_i n_i$, where $n_0, n_1, \ldots, n_i \in N$ and $e_1, \ldots, e_i \in E$. Moreover, the starting and ending nodes of p are defined as $\mathit{start}(p) = n_0$ and $\mathit{end}(p) = n_i$, respectively, and the concatenation of p with a path $p' = n_i e_{i+1} n_{i+1} e_{i+2} \cdots e_{i+j} n_{i+j}$ is defined as $\mathit{cat}(p, p') = n_0 e_1 n_1 e_2 \cdots e_i n_i e_{i+1} n_{i+1} e_{i+2} \cdots e_{i+j} n_{i+j}$. With this terminology, the evaluation of r over $\mathcal{L}$, denoted by $[\![r]\!]_{\mathcal{L}}$, is recursively defined as follows (omitting the usual interpretation for Boolean connectives $\neg$, $\vee$ and $\wedge$):

$$\begin{aligned} [\![?\ell]\!]_{\mathcal{L}} &= \{n \mid n \in N \wedge \lambda(n) = \ell\} \\ [\![\ell]\!]_{\mathcal{L}} &= \{n_0 e_1 n_1 \mid \rho(e_1) = (n_0, n_1) \wedge \lambda(e_1) = \ell\} \\ [\![\ell^-]\!]_{\mathcal{L}} &= \{n_0 e_1 n_1 \mid \rho(e_1) = (n_1, n_0) \wedge \lambda(e_1) = \ell\} \\ [\![r_1 + r_2]\!]_{\mathcal{L}} &= [\![r_1]\!]_{\mathcal{L}} \cup [\![r_2]\!]_{\mathcal{L}} \\ [\![r_1/r_2]\!]_{\mathcal{L}} &= \{\mathit{cat}(p_1, p_2) \mid p_1 \in [\![r_1]\!]_{\mathcal{L}} \wedge p_2 \in [\![r_2]\!]_{\mathcal{L}} \wedge \\ &\qquad \mathit{end}(p_1) = \mathit{start}(p_2)\} \\ [\![r^*]\!]_{\mathcal{L}} &= N \cup [\![r]\!]_{\mathcal{L}} \cup [\![r/r]\!]_{\mathcal{L}} \cup [\![r/r/r]\!]_{\mathcal{L}} \cup \cdots \end{aligned}$$

Notice that $?\ell$ is used to test the label of a node, ℓ is used to follow an edge with label ℓ, and ℓ^- is used to follow the opposite direction of an edge with label ℓ. Besides, as an example of a test with Boolean connectives, observe that $[\![(\neg\ell_1 \wedge \neg\ell_2)^-]\!]_{\mathcal{L}} = \{n_0 e_1 n_1 \mid \rho(e_1) = (n_1, n_0) \wedge \lambda(e_1) \neq \ell_1 \wedge \lambda(e_1) \neq \ell_2\}$. Hence, if $\mathcal{L}$ is the labeled graph in Figure 2(a), then

$$\begin{aligned} [\![\texttt{?person/contact/?infected}]\!]_{\mathcal{L}} &= \{n_1 e_4 n_5\}, \\ [\![\texttt{?person/rides/?bus/rides}^-\texttt{/?person}]\!]_{\mathcal{L}} &= \{n_1 e_2 n_3 e_3 n_4, \\ &\qquad n_4 e_3 n_3 e_2 n_1\}. \end{aligned} \quad (2)$$

As property graphs are an extension of labeled graphs, the grammar in (1) can be easily expanded to consider property values:

$$\mathit{test} ::= \ell \mid (p = v) \mid (\neg \mathit{test}) \mid (\mathit{test} \vee \mathit{test}) \mid (\mathit{test} \wedge \mathit{test}).$$

In particular, $(p = v)$ is used to verify whether the value of property p is v, with $p, v \in \mathbf{Const}$. Formally, the evaluation of a regular expression r over a property graph $\mathcal{P} = (N, E, \rho, \lambda, \sigma)$, denoted by $[\![r]\!]_{\mathcal{P}}$, is defined as for the case of labeled graphs but with three additional cases:

$$\begin{aligned} [\![?(p = v)]\!]_{\mathcal{P}} &= \{n \mid n \in N \wedge \sigma(n, p) = v\} \\ [\![(p = v)]\!]_{\mathcal{P}} &= \{n_0 e_1 n_1 \mid \rho(e_1) = (n_0, n_1) \wedge \sigma(e_1, p) = v\} \\ [\![(p = v)^-]\!]_{\mathcal{P}} &= \{n_0 e_1 n_1 \mid \rho(e_1) = (n_1, n_0) \wedge \sigma(e_1, p) = v\}. \end{aligned}$$

For example, we can extend regular expression (2) to indicate that the date of the contact between a person and an infected person is March 4th 2021:

$$\texttt{?person/(contact} \wedge \texttt{(date = 3/4/21))/?infected} \quad (3)$$

Regular expressions for vector-labeled graphs are defined exactly in the same way. If $\mathcal{V} = (N, E, \rho, \lambda)$ is a vector-labeled graph of dimension d, then a regular expression over $\mathcal{V}$ is defined by modifying grammar (1) to consider the following tests:

$$\mathit{test} ::= (f_i = v) \mid (\neg \mathit{test}) \mid (\mathit{test} \vee \mathit{test}) \mid (\mathit{test} \wedge \mathit{test}),$$

where $i \in \{1, \ldots, d\}$ and $v \in \textbf{Const}$. In particular, $(f_i = v)$ is used to verify whether the value of the i-th feature is v, which is formally defined as follows:

$$\begin{aligned}
[\![?(f_i = v)]\!]_{\mathcal{V}} &= \{n \mid n \in N \wedge \lambda(n)_i = v\} \\
[\![(f_i = v)]\!]_{\mathcal{V}} &= \{n_0 e_1 n_1 \mid \rho(e_1) = (n_0, n_1) \wedge \lambda(e_1)_i = v\} \\
[\![(f_i = v)^-]\!]_{\mathcal{V}} &= \{n_0 e_1 n_1 \mid \rho(e_1) = (n_1, n_0) \wedge \lambda(e_1)_i = v\},
\end{aligned}$$

where $\lambda(n)_i$ refers to the i-th feature of d-dimensional vector $\lambda(n)$, and likewise for $\lambda(e_1)_i$. Thus, for example, regular expression (3) can be rewritten as follows over the vector-labeled graph in Figure 2(c):

$$?(f_1 = \texttt{person})/(f_1 = \texttt{contact} \wedge f_5 = \texttt{3/4/21})/?(f_1 = \texttt{infected}).$$

4.1 Path extraction: enumeration, uniform generating and approximate counting

Computing the complete set of answers to a graph query can be prohibitively expensive [8, 44]. As a way to overcome this limitation, the idea of enumerating the answers to a query with a small delay has recently attracted much attention [46, 62]. More specifically, the computation of the answers is divided into a preprocessing phase, where a data structure is built to accelerate the process of computing answers, and then in an enumeration phase, the answers are produced with a polynomial-time delay between them.

Unfortunately, because of the data structures used in the preprocessing phase, these enumeration algorithms usually return answers that are similar to each other [14, 27, 62]. In this respect, the possibility of generating an answer uniformly, at random, is a desirable condition to improve the variety, if it can be done efficiently [1, 2, 13, 37]. However, how can we know how complete is the set of answers calculated by such algorithms? A third tool that is needed then is an efficient algorithm for computing, or estimating, the number of solutions to a query.

In the following we will present some recent results on efficient enumeration, uniform generation and approximate counting of paths conforming to a regular expression [9, 10]. We will give an overview of two of these results for labeled graphs, but the reader must keep in mind that they can be readily adapted to property and vector-labeled graphs.

The length of a path $p = n_0 e_1 n_1 e_2 \cdots e_k n_k$, denoted by $|p|$, is defined as k. The problem Count has as input a labeled graph $\mathcal{L}$, a regular expression r over $\mathcal{L}$ and a number k (given in unary as a string 0^k), and the task is to compute the number of paths $p \in [\![r]\!]_{\mathcal{L}}$ with $|p| = k$, which is denoted by $\textsc{Count}(G, r, k)$. The problem Count is known to be intractable; in fact, it is SPanL-complete [4], which implies that if Count can be solved in polynomial time, then P = NP [4]. However, in this tutorial we will show that Count can be efficiently approximated [9]. More precisely, we will present a randomized algorithm $\mathcal{A}$ that receives as input $\mathcal{L}$, r, k and an error $\varepsilon \in (0, 1)$, and computes a value $\mathcal{A}(G, r, k, \varepsilon)$ such that:

$$\Pr\left(\left|\frac{\textsc{Count}(G, r, k) - \mathcal{A}(G, r, k, \varepsilon)}{\textsc{Count}(G, r, k)}\right| \leq \varepsilon\right) \geq 1 - \left(\frac{1}{2}\right)^{100},$$

that is, with a very high probability the algorithm returns a value whose relative error is at most ε. Moreover, the algorithm works in polynomial time in the size of $\mathcal{L}$, r, and the values k, $1/\varepsilon$.

The problem Gen has the same input $\mathcal{L}$, r, k as Count, but the task is to generate uniformly, at random, a path $p \in [\![r]\!]_{\mathcal{L}}$ with $|p| = k$. In this tutorial, we will show that this problem can be solved efficiently [10]. More precisely, we will present a randomized algorithm $\mathcal{B}$ that is divided into a preprocessing and a generation phase. In the preprocessing phase, the algorithm construct with a very high probability a data structure, which can be repeatedly used in the generation phase to produce paths $p \in [\![r]\!]_{\mathcal{L}}$ of length k with uniform distribution.

4.2 Graph analytics: including knowledge

Graph analytic makes reference to a series of techniques to analyze the structure and content of a graph as a whole. Typical applications include clustering [61], computation of connected components and the diameter of a graph, computation of shortest paths between pairs of nodes, calculation of centrality measures [51], such as betweenness centrality [29] and PageRank [20], and community detection, such as finding the subgraph of a graph with the largest density [30, 45], to identify groups with a rich interaction in a network [41] or groups with suspicious behaviour [40, 53].

How should knowledge be included in such techniques? We focus here on the task of computing centrality. Given a labeled graph $\mathcal{L} = (N, E, \rho, \lambda)$ and nodes $a, b, x \in N$, let $S_{a,b}$ be the set of shortest paths from a to b in $\mathcal{L}$, and $S_{a,b}(x)$ be the set of paths in $S_{a,b}$ including node x. Then the betweenness centrality of a node x of $\mathcal{L}$ is defined as [29]:

$$bc(x) = \sum_{a,b \in N : a \neq x \wedge b \neq x} \frac{|S_{a,b}(x)|}{|S_{a,b}|}$$

This definition does not use the labels in $\mathcal{L}$, which may be a problem if not all the shortest paths passing through a node need to be considered to measure its centrality. Of course, not including some nodes and edges in the computation can be a solution to this problem. But, unfortunately, in many cases this is not enough as the pattern defining the paths to be taken into account can be more complicated. As an example, consider the labeled graph in Figure 2(a), and assume that we want to measure the centrality of bus n_3 as a transportation service with respect to other buses. In this case, we should only consider the shortest paths confirming to the regular expression $r = \texttt{?person/rides/?bus/rides}^-\texttt{/?person}$. That is, we must consider the paths where the bus is used as a transportation service for people, and not, for example, the paths with information about the company that owns it. In fact, if $S_{a,b,r}$ is the set of shortest paths from a to b conforming to the regular expression r, and $S_{a,b,r}(n_3)$ is the set of paths in $S_{a,b,r}$ including node n_3, then the centrality of n_3 can be redefined as follows:

$$bc_r(n_3) = \sum_{a,b \in N : a \neq n_3 \wedge b \neq n_3} \frac{|S_{a,b,r}(x)|}{|S_{a,b,r}|}$$

This definition can be generalized to any regular expression r. For example, the regular expression $r_1 = \texttt{?infected/rides/?bus/rides}^-\texttt{/}(\texttt{?person/(lives} + \texttt{contact)})^*\texttt{/?person}$ can be used in conjunction with betweenness centrality to measure the important of a bus in the propagation of an infection. In fact, r_1 is used to find pairs (a, b) of people such that a is infected and shared a bus with a person c, and b is connected to c through a path of arbitrary length of people that lives together or have been in contact with each other.

Betweenness centrality can be computed efficiently, as there exists an efficiently algorithm for the following problem: given a labeled graph $\mathcal{L}$, a pair of nodes a, b in $\mathcal{L}$ and a length k, count the number of paths of length k from a to b in $\mathcal{L}$. However, as mentioned in Section 4.1, the situation is different if regular expressions are considered as the previous problem is intractable [4]. How can we overcome this limitation? In this tutorial, we show how the tools presented in Section 4.1 can be used to provide an efficient randomized approximation algorithm for $bc_r(\cdot)$.

We conclude by pointing out that it is a challenging question how knowledge should be considered in centrality measures. In a recent article [58], the authors provide a natural and general framework to specify centrality measures, where betweenness centrality can be defined, but still without taking labels into consideration.

4.3 Node extraction: declarative versus procedural frameworks

The task of matching a pattern against a graph is fundamental when extracting knowledge. We have considered this problem for regular expressions, but such patterns can be specified in other frameworks, ranging from logic-based declarative languages [15, 38] to more procedural frameworks such as graph neural networks [48, 60]. The goal of this part of the tutorial is to show a recently established tight connection between these apparently different frameworks [16, 50, 71], which has interesting corollaries in terms of the use of declarative formalisms to specify patterns, versus the use of procedural formalisms to efficiently evaluate them.

Let $r = \texttt{?person/rides/?bus/rides}^{-}\texttt{/?infected}$. How should this regular expression be evaluated over the labeled graph $\mathcal{L}$ in Figure 2(a)? To think about this problem, let us focus on the task of retrieving the nodes a that can reach a node b by following a path conforming to r, that is, a path $p \in [\![r]\!]_{\mathcal{L}}$ such that $start(p) = a$ and $end(p) = b$. Pattern r is then used to retrieve the list of people who are possibly infected because they shared a bus with infected people. This regular expression can be specified in first-order logic:

$$\varphi(x) \;=\; \texttt{person}(x) \wedge \exists y \exists z\, (\texttt{rides}(x,y) \wedge \texttt{bus}(y) \wedge \texttt{rides}(z,y) \wedge \texttt{infected}(z)),$$

considering node labels as unary predicates and edge labels as binary predicates. This expression can be evaluated efficiently if the number of variables in it is bounded by a fixed constant [68]. Moreover, only unary and binary predicates are used in it, and they are placed in a sequence in which values of variables can be forgotten, allowing them to be reused. Indeed, the following first-order logic formula that uses two variables is equivalent to $\varphi(x)$:

$$\psi(x) \;=\; \texttt{person}(x) \wedge \exists y\, (\texttt{rides}(x,y) \wedge \texttt{bus}(y) \wedge \exists x\, (\texttt{rides}(x,y) \wedge \texttt{infected}(x))).$$

Thus, regular expression r can be evaluated efficiently by noticing that the result of any join in r is always a binary table, so no auxiliary relations with an arbitrary number of columns need to be stored. This idea has been successfully used in a variety of scenarios [3, 31, 32, 47, 68], and we are convinced that it should be kept in mind, not only as it provides an efficient way to evaluate regular expressions, but also as it allows to establish a tight connection with the more procedural and popular formalism of graph neural networks.

A graph neural network $\mathcal{G}$ receives as input a vector-labeled graph $\mathcal{V} = (N, E, \rho, \lambda)$, generates from it a vector-labeled graph $\mathcal{V}' = (N, E, \rho, \lambda')$, and then uses a classification function that returns either *true* or *false* for each $n \in N$ based on $\lambda'(n)$. In this way, $\mathcal{G}$ is a classifier [48, 60]. But also $\mathcal{G}$ can be considered as a unary query [16] that is true for a node n of a vector-labeled graph $\mathcal{V}$ if and only if the output of $\mathcal{G}$ is *true* for n. Hence, it is fundamental to understand the expressiveness of graph neural networks as a query language, in particular because they can act as an efficient procedural counterpart of more declarative query formalisms. And here again the use of a logic with a fixed number of variables plays a key role: it is proved in [22] that the Weisfeiler-Lehman test for graph isomorphism [70] has the same expressive power as an extension of first-order logic with counting and with a fixed number of variables, it is proved in [50, 71] that the Weisfeiler-Lehman test can be used to characterize the expressiveness of graph neural networks, and these ingredients are combined in [16] to provide a characterization of graph neural networks in terms of a logic with a fixed number of variables. Interestingly, the Weisfeiler-Lehman test is a message-passing graph algorithm [42], which is an algorithmic model intimately related with graph neural networks.

5 TAKEAWAY MESSAGES

The richness of the manifold technical developments in the area, part of which we reviewed, deserves to be encompassed in a conceptual map. Graphs have become ubiquitous in data and knowledge management. We argued that one of the main drivers of this blooming is the dual character of graphs: on one hand, being a simple, flexible and extensible data structure; and on the other, being one of the most deep-rooted form of representing human knowledge.

Graphs (as representation) unveil social aspects that are relatively far from being the main concerns of our area. Traditionally, we divided our labor between designers, organizing the conceptual boxes (through schemata and metadata) that contain data; and we, data people, dealing with preserving and transforming such data. Knowledge graphs mixed both worlds making difficult to trace a clear frontier between them.

Today we witness highly efficient techniques, particularly from the area of statistics, that are coping our area. They have to do with the massive and automated collection of new type of data, thus unstructured and uncertain. We have been using them mainly as tools, but large graphs force to incorporating them harmonically into our discipline. Much of this has to do with the classic counterpoint between logic and statistics, but it goes much deeper.

Finally, we are convinced that we should re-think the very notion of "querying" in graphs. We try to organize current research in three big areas, namely entities/nodes, relationships/connectivity, and emergent/global properties. But orthogonal to this is the extension of classical queries as languages for transforming data into data plus a final interpretation, into a loop with a continuous process of interaction between humans and data. Probably the allure of graphs today has to do with this loop.

ACKNOWLEDGMENTS

This work was funded by ANID - Millennium Science Initiative Program - Code ICN17_002.

REFERENCES

[1] Serge Abiteboul and Gilles Dowek. 2020. *The Age of Algorithms.* Cambridge University Press.

[2] Serge Abiteboul, Gerome Miklau, Julia Stoyanovich, and Gerhard Weikum. 2016. Data, Responsibly. *Dagstuhl Reports* 6, 7 (2016), 42–71.

[3] Natasha Alechina and Neil Immerman. 2000. Reachability Logic: An Efficient Fragment of Transitive Closure Logic. *Log. J. IGPL* 8, 3 (2000), 325–337.

[4] Carme Àlvarez and Birgit Jenner. 1993. A Very Hard log-Space Counting Class. *Theor. Comput. Sci.* 107, 1 (1993), 3–30.

[5] Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutiérrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. 2018. G-CORE: A Core for Future Graph Query Languages. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018.* 1421–1432.

[6] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5 (2017), 68:1–68:40.

[7] Renzo Angles and Claudio Gutierrez. 2008. Survey of Graph Database Models. *ACM Comput. Surv.* 40, 1, Article 1 (Feb. 2008), 39 pages.

[8] Marcelo Arenas, Sebastián Conca, and Jorge Pérez. 2012. Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012.* ACM, 629–638.

[9] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2019. Efficient Logspace Classes for Enumeration, Counting, and Uniform Generation. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019.* 59–73.

[10] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2020. Efficient Logspace Classes for Enumeration, Counting, and Uniform Generation. *SIGMOD Rec.* 49, 1 (2020), 52–59.

[11] Malcolm P. Atkinson, François Bancilhon, David J. DeWitt, Klaus R. Dittrich, David Maier, and Stanley B. Zdonik. 1992. The Object-Oriented Database System Manifesto. In *Building an Object-Oriented Database System, The Story of O2.* 3–20.

[12] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007 (Lecture Notes in Computer Science)*, Vol. 4825. 722–735.

[13] Martin Aumüller, Rasmus Pagh, and Francesco Silvestri. 2020. Fair Near Neighbor Search: Independent Range Sampling in High Dimensions. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020.* 191–204.

[14] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Proceedings of CSL.* 208–222.

[15] Pablo Barceló. 2013. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013.* 175–188.

[16] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. 2020. The Expressive Power of Graph Neural Networks as a Query Language. *SIGMOD Rec.* 49, 2 (2020), 6–17.

[17] Tim Berners-Lee, Roy Fielding, Larry Masinter, et al. 1998. Uniform resource identifiers (URI): Generic syntax.

[18] Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The Semantic Web. *Scientific American* 284, 5 (May 2001), 34–43.

[19] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.* 2787–2795.

[20] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Comput. Networks* 30, 1-7 (1998), 107–117.

[21] HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. 2018. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Trans. Knowl. Data Eng.* 30, 9 (2018), 1616–1637.

[22] Jin-yi Cai, Martin Fürer, and Neil Immerman. 1992. An optimal lower bound on the number of variables for graph identifications. *Comb.* 12, 4 (1992), 389–410.

[23] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (June 1970), 377–387.

[24] Richard Cyganiak, , David Wood, and Markus Lanthaler. 2014. RDF 1.1 concepts and abstract syntax, W3C Recommendation 25 February 2014.

[25] Martin Dürst and Michel Suignard. 2005. *Internationalized resource identifiers (IRIs).* Technical Report. RFC 3987, January.

[26] Dieter Fensel, Umutcan Simsek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. 2020. *Knowledge Graphs - Methodology, Tools and Selected Use Cases.* Springer.

[27] Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. 2020. Efficient Enumeration Algorithms for Regular Document Spanners. *ACM Trans. Database Syst.* 45, 1 (2020), 3:1–3:42.

[28] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An Evolving Query Language for Property Graphs. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018.* 1433–1445.

[29] Linton C Freeman. 1977. A Set of Measures of Centrality based on Betweenness. *Sociometry* (1977), 35–41.

[30] Andrew V Goldberg. 1984. *Finding a maximum density subgraph.* University of California Berkeley.

[31] Georg Gottlob, Christoph Koch, and Reinhard Pichler. 2002. Efficient Algorithms for Processing XPath Queries. In *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002.* 95–106.

[32] Georg Gottlob, Christoph Koch, and Reinhard Pichler. 2005. Efficient Algorithms for Processing XPath Queries. *ACM Trans. Database Syst.* 30, 2 (2005), 444–491.

[33] Martin Grohe. 2011. From Polynomial Time Queries to Graph Structure Theory. *Commun. ACM* 54, 6 (2011), 104–112.

[34] Martin Grohe and Pascal Schweitzer. 2020. The Graph Isomorphism Problem. *Commun. ACM* 63, 11 (2020), 128–134.

[35] Claudio Gutiérrez and Juan F. Sequeda. 2021. Knowledge Graphs. *Commun. ACM* 64, 3 (2021), 96–104.

[36] William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding Logical Queries on Knowledge Graphs. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada.* 2030–2041.

[37] Sariel Har-Peled and Sepideh Mahabadi. 2019. Near Neighbor: Who is the Fairest of Them All?. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada.* 13176–13187.

[38] Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013.

[39] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2020. Knowledge Graphs. *To appear in ACM Computing Surveys. CoRR* abs/2003.02320 (2020).

[40] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. FRAUDAR: Bounding Graph Fraud in the Face of Camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016.* 895–904.

[41] Jon M. Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *J. ACM* 46, 5 (1999), 604–632.

[42] H. T. Kung. 1982. Why Systolic Architectures? *Computer* 15, 1 (1982), 37–46.

[43] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.* 2181–2187.

[44] Katja Losemann and Wim Martens. 2013. The Complexity of Regular Expressions and Property Paths in SPARQL. *ACM Trans. Database Syst.* 38, 4 (2013), 24:1–24:39.

[45] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2020. Efficient Algorithms for Densest Subgraph Discovery on Large Directed Graphs. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020.* 1051–1066.

[46] Wim Martens and Tina Trautner. 2019. Dichotomies for Evaluating Simple Regular Path Queries. *ACM Trans. Database Syst.* 44, 4 (2019), 16:1–16:46.

[47] Maarten Marx. 2005. Conditional XPath. *ACM Trans. Database Syst.* 30, 4 (2005), 929–959.

[48] Christian Merkwirth and Thomas Lengauer. 2005. Automatic Generation of Complementary Descriptors with Molecular Graph Networks. *J. Chem. Inf. Model.* 45, 5 (2005), 1159–1168.

[49] Justin J Miller. 2013. Graph Database Applications and Concepts with Neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, Vol. 2324.

[50] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.

[51] Mark Newman. 2018. *Networks.* Oxford university press.

[52] Heiko Paulheim. 2017. Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic Web* 8, 3 (2017), 489–508.

[53] B. Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. 2010. EigenSpokes: Surprising Patterns and Scalable

Community Chipping in Large Graphs. In *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II (Lecture Notes in Computer Science)*, Vol. 6119. 435–448.

[54] Ross Quillan. 1963. *A Notation for Representing Conceptual Information: An Application to Semantics and Mechanical English Paraphrasing.* Systems Development Corporation.

[55] Ross Quillian. 1967. Word Concepts: A Theory and Simulation of some Basic Semantic Capabilities. *Behavioral science* 12, 5 (1967), 410–430.

[56] Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over Knowledge Graphs in Vector Space Using Box Embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020.*

[57] Richard H Richens. 1956. Preprogramming for Mechanical Translation. *Mech. Transl. Comput. Linguistics* 3, 1 (1956), 20–25.

[58] Cristian Riveros and Jorge Salas. 2020. A Family of Centrality Measures for Graph Data Based on Subgraphs. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark (LIPIcs)*, Vol. 155. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 23:1–23:18.

[59] Ian Robinson, Jim Webber, and Emil Eifrem. 2015. *Graph Databases: New Opportunities for Connected Data.* O'Reilly Media, Inc.

[60] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Trans. Neural Networks* 20, 1 (2009), 61–80.

[61] Satu Elisa Schaeffer. 2007. Graph Clustering. *Comput. Sci. Rev.* 1, 1 (2007), 27–64.

[62] Luc Segoufin. 2013. Enumerating with Constant Delay the Answers to a Query. In *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013.* ACM, 10–20.

[63] Ehud Shapiro, David H. D. Warren, Kazuhiro Fuchi, Robert A. Kowalski, Koichi Furukawa, Kazunori Ueda, Kenneth M. Kahn, Takashi Chikayama, and Evan Tick. 1993. The Fifth Generation Project: Personal Perspectives. *Commun. ACM* 36, 3 (1993), 46–103.

[64] Michael Stonebraker and Joey Hellerstein. 2005. What goes around comes around. *Readings in database systems* 4 (2005), 1724–1735.

[65] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *Proc. VLDB Endow.* 4, 11 (2011), 992–1003.

[66] Jeffrey D. Ullman. 2020. The Battle for Data Science. *IEEE Data Eng. Bull.* 43, 2 (2020), 8–14.

[67] Oskar van Rest, Sungpack Hong, Jinha Kim, Xuming Meng, and Hassan Chafi. 2016. PGQL: A Property Graph Query Language. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, Redwood Shores, CA, USA, June 24 - 24, 2016.*

[68] Moshe Y. Vardi. 1995. On the Complexity of Bounded-Variable Queries. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA.* 266–276.

[69] Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: A Free Collaborative Knowledge Base. *Commun. ACM* 57, 10 (2014), 78–85.

[70] Boris Weisfeiler and Andrei Leman. 1968. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series* 2, 9 (1968), 12–16.

[71] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.*