Proceedings of the 16th Conference on Computer Science and Intelligence Systems pp. 59–68
DOI: 10.15439/2021F29
ISSN 2300-5963 ACSIS, Vol. 25

An efficient approach towards the generation and analysis of interoperable clinical data in a knowledge graph

Jens Dörpinghaus*, Vera Weil‡, Sebastian Schaaf†, Tobias Hübenthal‡
* German Center for Neurodegenerative Diseases (DZNE), Bonn, Germany, Email: jens.doerpinghaus@dzne.de
† Fraunhofer Institute for Algorithms and Scientific Computing, Schloss Birlinghoven, Sankt Augustin, Germany
‡ Department of Mathematics and Computer Science, University of Cologne, Germany

***Abstract*—Knowledge graphs have been shown to play an important role in recent knowledge mining settings, for example in the fields of life sciences or bioinformatics. Contextual information is widely used for NLP and knowledge discovery tasks, since it highly influences the exact meaning of expressions and also queries on data.**

The contributions of this paper are (1) an efficient approach towards interoperable data, (2) a runtime analysis of 14 real-world use cases represented by graph queries and (3) a unique view on clinical data and its application, combining methods of algorithmic optimisation, graph theory and data science.

I. Introduction

Personalized, or more precisely, stratified medicine aims for matching certain risk groups and possibly yet unknown subgroups to treatments, ultimately optimizing patients' responses, mainly to available drugs. This explicitly includes strategies beyond guidelines and off-label usage of substances in order to increase the effect and/or to decrease undesired side effects. For this purpose, collected primary data of the examined persons have to be linked with data from secondary sources like publications or databases in an application-oriented way. [1]

Considering both amount and complexity of input data, at least early steps of data processing require computer methods and thus machine-readable data. While these early steps are highly automated and deeply influence subsequent analyses, proper modeling of data and derived knowledge is key for each input, analysis and output layers of a system supporting decisions by human expert users. Considering data of interest to be highly heterogeneous as well as evolving and growing over time, the abstraction to semantic entities and relations appears favorable. Thus, graphs appear as a feasible basis for modeling both data and metadata. Moreover, external resources could be linked to such a knowledge graph. Finally, for querying contents modeled in such way, highly generic and powerful graph algorithms are available.

But how can clinical patient data from a database be efficiently stored as a knowledge graph using a suitable data schema? And how can queries be generated afterwards using the data linked in this way? What is the runtime of an application-related query for suitable literature for a given patient?

In this work, building on the already stored *PubMed* data from [2], a model will be presented that efficiently embeds the collected primary data into structured, domain-specific environments, in this case ontologies. The thereby generated knowledge graph will then be used to examine individual queries in terms of efficiency. See figure 1 for an illustration schema. For the purpose of illustration of real-world data a small section of the underlying graph in Neo4j can be seen in Fig. 2.

The contributions of this paper are an efficient approach towards interoperable data, a runtime analysis of 14 real world use cases represented by graph queries and a unique view on clinical data and its application combining methods of algorithmic optimisation, graph theory and data science. This paper is divided into seven sections. After an introduction, the second section gives a brief overview over the state of the art and related work. The third section describes the theoretical and practical background and the methods used for our novel approach. Therefore, we will refer to both knowledge graphs and dedicated algorithms. In the fourth section, we present our optimization approaches to tackle the interoperability challenges within our use cases. The fifth section covers applications from real-world use cases like query finding, with the experimental results on both artificial and real-world scenarios in the subsequent section. After that, we present a detailed evaluation in section six. Our conclusions and outlooks are drawn in the final section. We will propose a

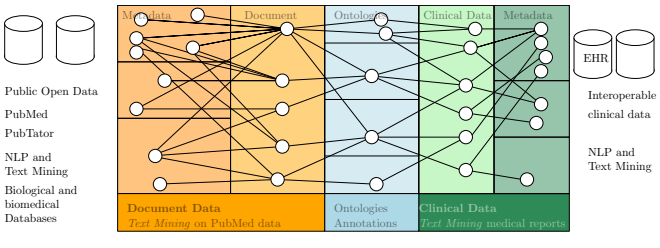

Fig. 1. Illustration of some knowledge graph layers found in the testing environment. Here, we can see document data extracted using Text Mining on PubMed Data with their metadata (e.g. authors, publication venues, publication date) and named entities from different ontologies. They form the interoperability layer to the clinical data. In the background we use medical reports analyzed with the help of text mining methods.

novel algorithmic approach which presents promising performance. The results show a significant improvement for new algorithms for knowledge discovery on clinical data.

II. Related work

In scientific research, expert systems provide users with several methods for knowledge discovery. They are widely used to find relevant or novel information. A popular example in biomedical research is to try to find molecular pathways; controlled reaction mechanisms within biological organisms, which might be misregulated in pathogenic states. Obviously, understanding these cascades, their players and relations to diseases is key to design and apply drugs in a targeted way.

Being confronted with patients' clinical data and with expert knowledge in the back of their minds, clinical researchers usually consider an initial idea and start integrating external content such as scientific papers. The most common approach is inquiring with a search engine about some terms to find closely related information. Effectively, users most frequently query for additional documents or patient files to adjust the search query. Similarly, for a given set of documents or patients the question might be on commonalities considering a certain topic. Both approaches are heavily related to the context of data points, see for example [3] for PubMed data. Topic labelling – or cluster labelling – is constantly being explored in several research fields.

In principle, the way external data sources and manually curated data are integrated is key. Although several commercial solutions exist, Fakhry et al. state that the "adoption and extension of such methods in the academic community has been hampered by the lack of freely available, efficient algorithms and an accompanying demonstration of their applicability using current public networks." [4]. This and the emerging improvements on large-scale Knowledge Graphs and machine learning approaches are the motivation for our novel approach on semantic Knowledge Graph embeddings for biomedical research utilising data integration with linked open data. Several similar approaches (often in the context of drug-repurposing) such as Bio2RDF [5], hetionet [6], or OpenPHACTS [7] have already been described. Our approach is more focussed on integrating the literature itself in a FAIR

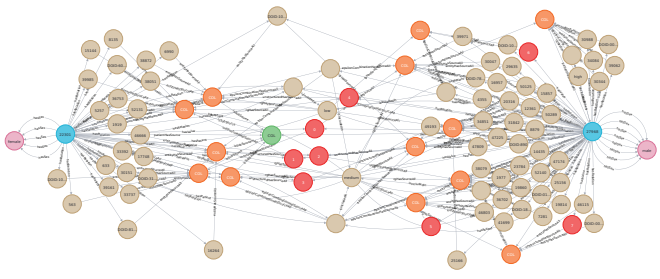

Fig. 2. Example illustration of the proposed knowledge graph centered around "Alzheimer's disease". We find different patients (blue nodes), values (red nodes) and ApoE allele combinations (orange nodes). The light brown nodes refer to entities, for example from Disease Ontology. These are highly important for the interoperability of the knowledge graph.

[10] and open knowledge graph, which is also accessible as a public resource.

In recent decades the field of natural language processing (NLP) and knowledge discovery as well as data mining and the management of information systems as the related fields are emerging. It is not exactly the focus of this paper, but since we are relying on data obtained from biomedical texts, we should note that several authors like Manning et al. [8] or Clarc et al. [9] give an overview about the algorithmic part of computational linguistics and NLP. In addition there is a constant interest in using graphs for these problems, see [10].

III. Background

Using graph structures to house data carries several advantages for the integration of knowledge and its targeted re-extraction. According to their generic character such integrative knowledge graphs are important for the life sciences, medical research and associated fields, not least supporting their interconnection on a formal level. Considering systems medicine applications, knowledge graphs provide grounds for holistic approaches unraveling disease mechanisms. In these and other common settings pathway databases play an important role. As a basis, biomedical literature and text mining are used to build knowledge graphs, see [11]. As part of the studies on integrative data semantics within clinical research, data on patients suffering from certain diseases have been collected by various institutions. In our case, the data was available in the form of a NoSQL Mongo database. In addition, several databases and ontologies can implicitly form a knowledge graph. For example Gene Ontology, see [12], DrugBank, see [13] or [14] cover large amounts of relations and references which other fields can refer to.

In [2] we collected 27 real world questions and queries in scientific projects to test the performance and output of the knowledge graph. We could show that the performance of several queries was very poor and some of them even did not terminate. In order to identify limitations and understand the underlying problems, we carried on with our work. The testing system is based on *Neo4j* and holds a dense large-scale labeled property graph with more than 71M nodes and 850M edges. They are based on biomedical knowledge graphs as described in [11].

The more specific data model for clinical research used in this work initially contains general variables about a patient, e.g., site of measurement (*site*), sex (*sex*), and genotypes (ApoE). Additionally, a variety of neuropsychological testing results (NPT), recorded disorders (disturbances), laboratory measurements (LAB), liquid biopsy markers (LIQ), spinocerebellar ataxia characterisations (SCA) and diagnoses (both ICD10-encoded and free text) are given as variables.

Due to the sensitivity of personal clinical data, a sample of artificial data is used within this work. They do not cover the full data schema presented later (see Fig. 6). However, the simplified model is sufficient to present the principle of the work, and can be easily transferred or extended to a full data set using the data schema.

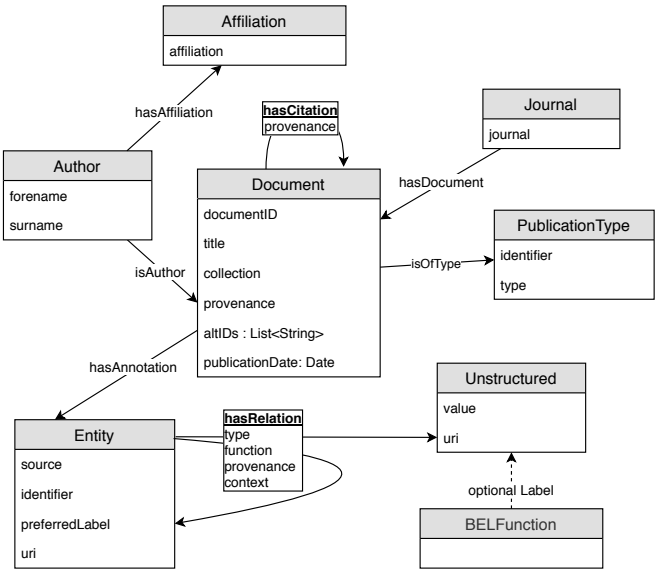

Fig. 3. Formal data schema of the knowledge graph of the *PubMed* database.

This work is also based on the Paper *Towards context in large scale biomedical knowledge graphs* [2], in which an efficient polyglot persistence design for storing and querying a knowledge graph based on the documents contained in the *PubMed* database is presented on the basis of a created data schema. There, based on questions from the biomedical domain, a knowledge graph is created by resorting to the data schema given in Fig. 3. [2]

Data schema and practical implementation are based on clinical questions, which are the result of several interviews with employees from the biomedical field. They form the basis for the knowledge graph requirements.

IV. Optimization approaches

A. Data Schema and Knowledge Graph Foundations

Based on the given data, we need to tackle the challenge of how to arrange data points in a suitable schema and thus store them efficiently in a knowledge graph so that the clinical questions can be answered using graph queries. Simultaneously, the later connection to the graph of the *PubMed* database in [2] needs to be solved. For this purpose, we use the schema represented in Fig. 3. Extended with data from clinical studies an enrichment with context data is also possible, applying a mapping between the entities and several ontologies. Here, a mapping means a function $M : E(D) \rightarrow E(O)$, where $E(D)$ describes the entities of the given data and $E(O)$ the entities of the used ontologies. Thus, the mapping consists of edges that create a logical relation between biomedical data and lexicons in the knowledge graph, see [2].

First, the used classes of the model presented in Fig. 6 should be introduced. The initial and central class is the patient itself. It does not contain any other attributes except for an ID, as those are outsourced for the reasons mentioned above.

The next class to be considered is the patients' gender which is represented by the node `sex`. Another class is the ApoE risk type. It describes a genetic condition that is closely associated with risks for Alzheimer's disease. For each patient, either a 2-tuple $(\epsilon i, \epsilon j)$, $i, j \in \{1, 2, 3, 4\}$ or two 2-tuples $(rs429358, rs7412)$ of SNPs, which in turn form a $\epsilon i, i \in \{1, 2, 3, 4\}$, are given. The individual alleles in this case again consist of a tuple of SNPs, which can be either of type C or T. For illustration, the small model is shown in Fig. 4.

This construction results in the class of the risk group, which can have different patient-specific expressions. These expressions are implicitly given by the risk types. Three different categories are distinguished with the following designations:

- low-risk: patients whose risk type corresponds to a tuple $(\epsilon i, \epsilon j)$, $i, j \in \{1, 2, 3\}$.
- medium-risk: patients whose risk type corresponds to a tuple $(\epsilon i, \epsilon j)$ with $i = 4, j \in \{1, 2, 3\}$ or $i \in \{1, 2, 3\}, j = 4$.
- high-risk: patients whose risk type corresponds to the tuple $(\epsilon 4, \epsilon 4)$.

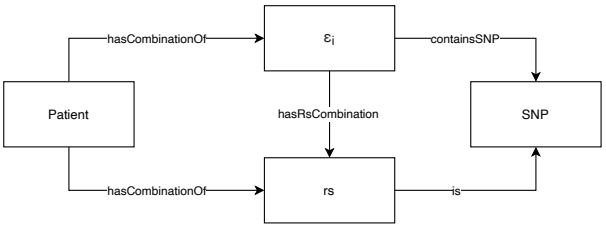

Fig. 4. Structure of ϵi-tuples and rs-codes. For further explanaition regarding the biology behind this concept, see [15]

In our case, the ϵ tuple is used for the artificial data. However, the model can of course be applied to the alleles as well, since there is a unique bijective mapping between the set of alleles relevant to the ApoE and the ϵ-tuples. [15]

Next, we consider measurements of clinical trials and studies. These include, as already mentioned in. section III, LAB, LIQ, NPT, ApoE, SCA, and diagnoses. They are collected in the class of `attributes` and belong to a parent category, *topic* stored under `unstructured`. Values are assigned to the `attributes` as part of the patient examinations. Like the *topics*, these are created as separate objects of the `Unstructured` class and linked to the `attributes` class via relations. The values themselves are not initially associated with any entity. Another class `unit` is created, which stores the associated unit for each measured value. This, as mentioned above, again allows for a wider range of viewing and interrogation options for the graph. In addition, one can connect the units to a suitable ontology, for example UCUM (Unified Code for Units of Measure, [16]), using appropriate relations.

Furthermore, one or more diagnoses can be assigned to a patient. These are also defined as a separate class and are stored as a *diagnosis code*. This matches the *diagnosis codes*

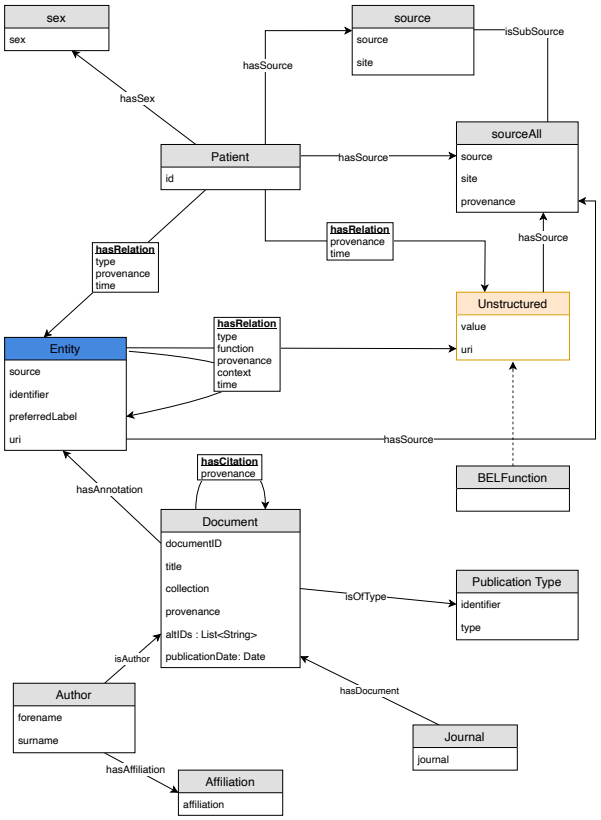

Fig. 5. Formal data schema of the knowledge graph combining clinical data and document data.

of the *Disease Ontology* in its descriptive usage, so it is also used by default in subject-related literature. [17]

In addition, we use the class `time`. Every measurement and every examination is provided with a timestamp. Thus, a temporal hierarchy can be determined for any subset of the total data and, under certain circumstances, even trends regarding the course of the disease can be identified. Commonly, the time stamps describe the date of an examination, but might also retroactively refer to the date of the first occurrence of a symptom or an event.

The last classes used are `source` and `sourceAll`. Here, `source` serves as the source of a dataset of a specific clinical institute of the DZNE. The location where the data was collected is also stored. This enables to locally delineate the data from each other and allows for combining datasets of different origins in one graph without losing their affiliation.

The class `sourceAll`, inheriting from `source`, is almost identical to the latter, but has the additional attribute *provenance*. This makes it possible to unambiguously define and record the relationships between different instances of classes within a given data set. The classes mentioned above have in different relations to one another. However, the outsourcing of attributes and the resulting increase in the number of nodes also increases the number of edges. In a directed graph $G = (V, E)$ with a number of nodes $n = |V|$, the number of edges in the worst case is $|E| = n \cdot (n-1)$, since each node $v \in V$ can have an edge to any other node $u \in V, u \neq v$.

The naming of the edges follows the Dublin Core standards, see [18]. These are simple standards of the Dublin Core Metadata Initiative for data formats of documents or objects. The vocabulary listed there has partially been replaced by more domain-specific expressions, while preserving the basic structure. As an example the term *hasFormat* is replaced by the more adequate *hasSex* which keeps the original structure and represents the `patient-sex` relationship.

In addition to the name or label of an edge, further information is required. Each relation of two classes receives a timestamp, which is stored in the attribute *time*. This can be given explicitly by a concrete date or also implicitly by the relationship of the classes and relations to each other. In addition, the edges receive an attribute *provenance*. This correlates with the attribute of the same name of the `sourceAll` node and assigns an internal membership to relations between several classes. Thus, the *provenance* attribute, whether within a node class or as an attribute of a relation, prevents data ambiguity.

These considerations yield the schema in Fig. 6. However, we need to create a more generic schema for the graph database. Therefore, we extend the schema from [11] by combining all blue marked entities from Fig. 6. Thus, the schema in Fig. 5 is obtained.

B. Creating interoperable data

We use the bulk import function of *Neo4j* to load the data. For this, we converted the input data into CSV files. This import consists of the following steps:

Algorithm 1 INTEROPERABLE-DATA

Require: import file f
Ensure: CSV export c
createPathEnvironment(c)
createStandardNodes()
impCSVFile(f)
writeNodes(f,c)
writeEdges(f,c)
writeHeaders(c)
return c

The method `createPathEnvironment(`c`)` creates the path environment for the export c, `createStandardNodes()` creates the standard nodes (e.g. for `sex`). They can be outsourced of the main methods below as they are static and do not depend on the input data.These first steps can be done in linear time.

The main method is `impCSVFile(`f, c`)`. First, it gets the import file f containing the clinical data. Then, for each entry all necessary nodes and relations are created while avoiding duplicates with the help of lists and sets. Regarding the time complexity of the look-up function, the latter is far superior to the former: Sets in Python are implemented as hash tables using keys as indices and therefore provide an average look-up time of $\mathcal{O}(1)$ instead of lists which need $\mathcal{O}(n)$ (see [19].

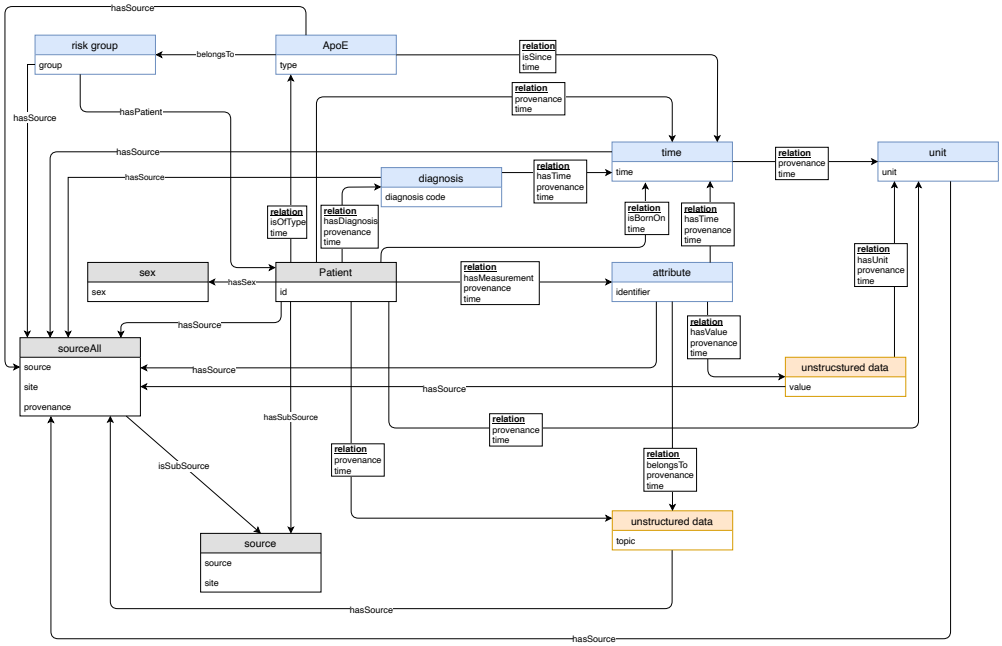

Fig. 6. Detailed data schema deduced from the IDSN data model. In orange, the node class `Unstructured` represents unstructured data considered complementary contextual information provided for entities and patients.

`writeNodes`(f, c) and `writeEdges`(f, c) then print the CSV files from the nodes and edges created beforehand. Both simply go through a list of stored nodes which takes $\mathcal{O}(n)$ time. Lastly, `writeHeaders` creates the necessary header files for the bulk import which happens in $\mathcal{O}(1)$ (see [20]).

With out first approach using lists to avoid duplicate nodes and edges we achieved a total time complexity of $\mathcal{O}(n^2)$ where n represents the size of our import data f. When replacing lists with sets we could lower the total time complexity to $\mathcal{O}(n)$. The explicit runtimes shown in Fig. 11 in section VI-D prove the theoretical difference.

V. Use-cases and Graph Queries

We obtain our use-cases from clinical questions. These are ordered and categorized within this section in order to later analyze their efficiency in VI and to consider comparative values by means of complexity theory. Preliminary work has been carried out in [21] and [22]. There, biomedical questions are examined and optimized as well. For this purpose, six different literature sources are used, some of which cite different and some common categories or classes for graph-based queries. In [2], a hierarchy for the classification of the queries is created with reference to fitting literature sources and using the author's own criteria.

This section focuses on the individual biomedical issues. They were collected through interviews with clinical and biomedical professionals. The process is similar to the one in [2] First the input and output for all questions is described. Since the graph contains only a subset of the actual intended data, the original clinical questions have to be replaced. However, the categories of the queries, assigned as proposed in [2], shall be maintained. From a biomedical perspective, these questions may not make sense or might not be of interest; however, from a computer science perspective they are isomorphic to the original ones. The questions are numbered in the listing so that they correlate with the later queries.

In spite of no biological question to base upon, queries 13-15 have been added to test the algorithms provided within *Neo4j*. The associated queries formulated in *Cypher* can be found in table I.

VI. Evaluation

The queries created in V are now to be applied to the graph and evaluated with respect to their runtimes. For this purpose, the actual runtime in *Neo4j* is measured for multiple executions and related to the time complexity of the algorithms. The queries are presented according to their categories. Here, the numbering is done following table I.

A. RPQ

The simplest query in the class of Graph Navigation Queries is whether a certain path exists in the graph, see [22]. Path queries specified with regular expressions are commonly referred to as *Regular Path Queries (RPQ)*, see [23].

Most of the queries are RPQs (Q1, Q2, Q3, Q5, Q7, Q11, Q12). According to [24], these have a polynomial runtime due to transformation into a non-deterministic finite automaton.

Query 1 refers to the question *For which patients do complete neuropsychological tests exist?*. This question can be substituted by *Which patients have the most distinct HGNC values?*. The corresponding Cypher query is:

TABLE I
CLINICAL QUESTIONS, THEIR COMPLEXITY CLASS AND REPLACEMENTS FOR TESTING PURPOSE. HERE DC REFERS TO *Degree Centrality*, SP TO *Shortest Path* AND BC TO *Betweenness Centrality*.

	Class	Question	Replacement
1	RPQ	For which patients do complete neuropsychological tests exist?	Which patients have the most distinct HGNC values?
2	RPQ	Which measurement values are most common in the context of {diagnosis1}?	Which patients are found most often in the context of a risk group {RiskGroup1}?
3	RPQ	Which measurements are collected at the same time?	Which HGNC values are most commonly collected during a certain visit {visit1}?
4	CRPQ	What does the chronological order of the measured values of entity {entity1} and risk group {RiskGroup2} for a patient {patient1} look like?	What does the chronological order of the measured HGNC values {entity1} and risk group {RiskGroup2} for a patient {patient1} look like?
5	RPQ	How many patients received a diagnosis within two days of their visit?	How many patients received a diagnosis on their first visit?
6	CRPQ	Which patients diagnosed with {diagnosos2} underwent neuropsychological testing {number1} days beforehand?	Which patients diagnosed with {diagnosis2} at visit {visit2} had the HGNC value {HGNC_value} at their previous visit?
7	RPQ	Do people of age {age1} come for examination more often than others?	Do people of sex {sex1} come for examination more often than others?
8	CRPQ	Which entity {entity2} do patients without any diagnosis have in common?	Which patients are diagnosed with exactly {number} distinct diagnoses and to which risk groups do they belong?
9	DC	How often does an allel tuple {allel tuple} appear amongst all patients?	Which risk group is most common amongst all patients?
10	ECRPQ	What literature {literature1} can be found for patient {patient2} diagnosed with {diagnosis3}?	Which HGNC values {HGNC_value2} can be found for patient {patient2} diagnosed with {diagnosis3}?
11	RPQ	How many patients underwent neuropsychological testing {npt1} and at the same time have laboratory value {LAB_value1}?	How many patients are diagnosed with {diagnosis4} and at the same time have an HGNC value {HGNC_value}?
12	RPQ	How many Patients suffer from disturbance {disturbance1} and what sex are they?	How many patients are diagnosed with {diagnose5} and what sex are they?
13	SP	-	What is the shortest path between entity {entity4} and entity {entity5} and what is on this path?
14	BC	-	Which patient connects entities most strongly?

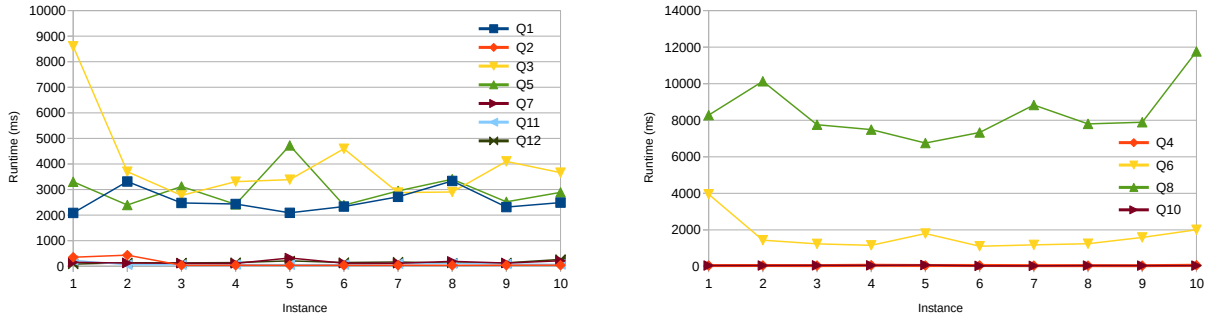

Fig. 7. Runtimes of different RPQ queries (left) and runtimes of (E)CRPQs queries (right) in milliseconds. Mean values are 2558.5 (Q1), 109.8 (Q2), 3990.7 (Q3), 3010.5 (Q5), 160.3 (Q7), 90.9 (Q11) and 154.5 ms (Q12) for RPQ and 53.6 (Q4), 1671.7 (Q6), 8402.6 (Q8) and 52.2 ms (Q10).

(Q1)
```
MATCH (e:Entity {source:'HGNC'}) <- [:hasRelation]-(p:Patients) RETURN p.patient AS Person, COUNT(DISTINCT e) AS number ORDER BY number DESC LIMIT 10
```

As another example query 3 answers the question *Which measurements are collected at the same time?*. It can be substituted by *Which HGNC values are most commonly collected during a certain visit {visit1}?*. The corresponding Cypher query is:

(Q3)
```
MATCH (e:Entity {source:'HGNC'}) <- [:hasRelation]-(p:Patients)-[:hasValue] -> (v:Unstructured {value:'2'}) RETURN e.preferredLabel AS HGNC_Wert, COUNT(e) AS number ORDER BY number DESC LIMIT 10
```

The respective average runtimes of each query can be seen in 7. As we can see, we have slower (Q1, Q3 and Q5) and faster queries(Q2, Q7, Q11 and Q12).

As discussed in [22], queries become slower with increasing number of queried attributes and used relations. Here, we first consider the queried attributes. However, these do not differ fundamentally for the two groups of fast and slow RPQs. They vary between one and three, but several attributes are needed even for the faster running queries. The comparison of the relations leads to more conspicuousness: Regarding the number of relations used by the slow (Q3 and Q4) and the fast

Queries (Q7 and Q12), on first sight, there is no difference. They all make use of two relations. But when looking at the nodes used within the queries a notable difference can be found:One of the nodes used in Q7 and Q12 ist `sex` which only has two possible specifications: male and female. The queries Q3 and Q4 have a similar structure, but instead of using `sex` they use the far larger class `entity` which holds more than 50,000 nodes. The former class contains only two distinct nodes (male and female), while the latter class contains over 50,000 nodes. Thus, many more nodes need to be checked, which explains the speed difference.

B. CRPQ and ECRPQ

Conjunctive queries and *RPQ*s can be combined in the class *CRPQ*, see [23], which then can be extended further by the extended CRPQs which include the possibility to specify path variables and even allows the output of a query to be a path. According to the schema presented in [22] they both are subproblems of pattern matching in graphs. We consider both CRPQs (Q4, Q6, Q8) and ECRPQ (Q10, see Fig. 8) together.

For example, Query 4 is answering the question *What does the chronological order of the measured values of entity entity1 and risk group RiskGroup2 for a patient patient1 look like?* This question can be substituted with *What does the chronological order of the measured HGNC values entity1 and risk group RiskGroup2 for a patient patient1 look like?* The Cypher query can be formulated as follows:

(Q4)
```
MATCH (u:Unstructured) <-[:hasValue]-(p:Patients {patient: '22504'}) - [:hasRelation] -> (e:Entity {source: 'HGNC'}) MATCH (r:RiskGroups) - [:hasPatient] -> (p) RETURN e.preferredLabel AS HGNC, r.riskgroup AS RiskGroup, u.value AS VisitNo ORDER BY u.value DESC
```

Question 10 (ECRPQ) answers the question *What literature {literature1} can be found for patient {patient2} diagnosed with {diagnosis3}?* In our testing environment we can substitute this query with *Which HGNC values {HGNC_value2} can be found for patient {patient2} diagnosed with {diagnosis3}?*. The Cypher query is expressed as follows:

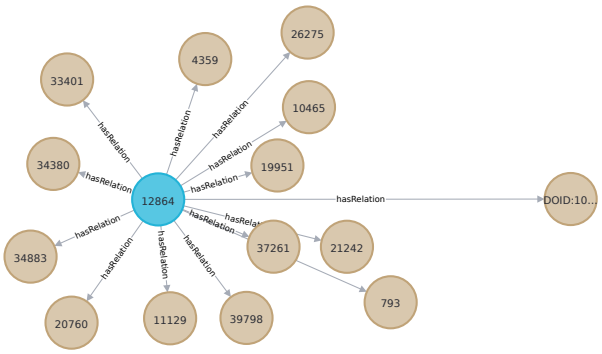

Fig. 8. Example output of ECRPQ query Q10.

```
10: MATCH p=(e1:Entity {preferredLabel:"Alzheimer's disease"}) <- [:hasRelation] -(A:Patients {patient: "12864"}) - [:hasRelation] -> (e2:Entity {source:"HGNC"}) RETURN p
```

The results and runtimes can be found in Fig. 7.

According to [25], the class of CRPQ is $\mathcal{NP}$-complete, so it can probably not be solved efficiently. The figure mentioned above clearly shows this fact because, besides very good results for Q4 and Q10, one also finds very poor runtimes for Q6 and Q8. The last-mentioned query immediately catches the eye, since, according to I, it covers several of the above-mentioned aspects. It is the only one of the (E)CRPQs presented here that works globally, accesses four different attributes and uses two different relations. Q6 does not search globally, indeed only locally, but it also has two different edge types, each of which occurs in both conjugate subquerys, and even queries four different attributes.

In particular Q4 is interesting. The query only searches locally, but queries five different attributes and uses three different relation types. Nevertheless it has quite a low runtime and the structure of the graph might offer an explanation for this seeming discrepancy. A directed graph $G = (V, E)$ with $n = |V|$ nodes can have up to $n \cdot (n-1)$ edges under the assumption that there are no duplicate edges, because each node v can have an edge (v, u) to any other node $u \in V, v \neq u$. Looking at the source files used here, the low density of the graph is immediately noticeable. An example of this is provided by the input CSV file, which contains about 30,000 patients, but at most six visits per patient, which in turn include fewer than 20 relations. So there are no more than 120 edges per patient. Looking at the whole graph, this ratio can also be seen when importing into the database: with half a million nodes, only slightly more than 2.6 million edges are created. Thus, it is noticeable in Q4 that the small portion of data that the query looks at is rather sparse. There is very little data on a patient, so only a very limited number of possibilities needs to be considered. This may explain the quick response of the database.

Lastly, we take a look at the ECRPQ Q10. It uses three different attributes but only one edge type. Here, we find the same result as previously discussed for Q4: only a very limited amount of data is available for the patient, which is likely to have a significant impact on the speed of the query.

C. Other results

Finally, we take a closer look at the algorithms used. All three are algorithms integrated into *Neo4j* and provided via the Graph Data Science 1.1.3 plug-in (Q9, Q13, see Fig. 9 and Q14). First, we explore Q9 with Degree Centrality.

(Q9)
```
CALL gds.alpha.degree.stream({ nodeProjection: ['Entity','Patients'], relationshipProjection: 'hasPatient' }) YIELD nodeId, score RETURN gds.util.asNode(nodeId).identifier AS name, score AS numberOfPatients ORDER BY
```

`numberOfPatients DESC LIMIT 3` Since the incident edges of each node are counted, for a dense graph $G = (V, E)$ we obtain a complexity of $\mathcal{O}(|V|^2)$ in the worst case, i.e. every node $v \in V$ is incident to every other $u \in V, u \neq v$. In [22], it is already shown that the algorithms implemented in *Neo4j* have such a high running time on large networks that they become practically almost useless. However, the runtimes and the average of query Q9 shown in Fig. 10 initially show otherwise for Degree Centrality. Once again, the fact that the graph is very sparse comes into play. The nodes have only a few direct neighbors and this has a strong effect on the runtime of the algorithm. Here, we use Degree centrality for a node with only a few different specifications. Thereby, the long runtime can be circumvented by constructing three individual queries for the different risk groups instead of using the algorithm in the first place.

Similarly, Q14 calculates the betweenness centrality:

(Q14)
```
CALL gds.graph.create
('myUndirectedGraph',
["Patients","Entity"], {hasRelation:
{orientation: 'UNDIRECTED'}})
CALL gds.alpha.betweenness.stream
('myUndirectedGraph') YIELD
nodeId, centrality RETURN
gds.util.asNode(nodeId).preferredLabel
AS entityLabel, centrality AS number ORDER
BY number DESC LIMIT 10
```

Q13 computes a shortest path. The Cypher query calls the function `shortestPath`:

(Q13)
```
MATCH(entity1:Entity {identifier:
'DOID:0040005'}) MATCH(entity2:Entity
{identifier: '41022'}) CALL
gds.alpha.shortestPath.stream({startNode:
entity1, endNode: entity2, nodeProjection:
'*', relationshipProjection:
{all:{type: '*', orientation:
'UNDIRECTED'}}}) Yield nodeId, cost RETURN
gds.util.asNode(nodeId), cost
```

The runtime is shown in Fig. 10. Q13 has a long runtime which is not surprising, as centrality measures for knowledge graphs are quite complex. While several efficient algorithms have been proposed, see [26], and some more specific problems are known to be $\mathcal{NP}$-hard. Here, good examples are Group Closeness Maximization (GCM), see [27] or the Maximum Betweenness Centrality, see [28].

There are several algorithms to compute shortest paths

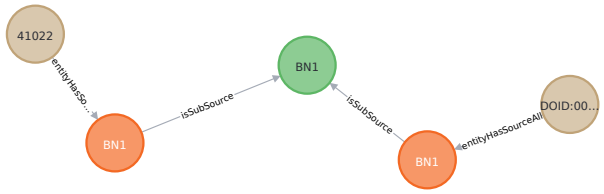

Fig. 9. Example output of shortest path query Q13.

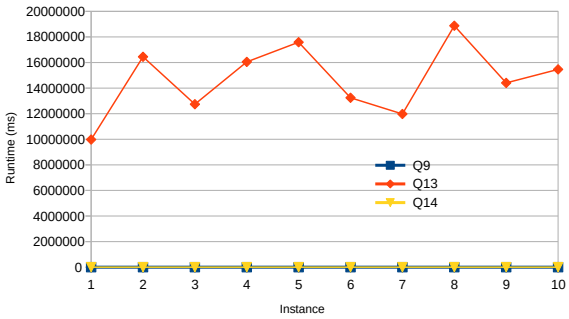

Fig. 10. Runtimes of different queries (Q9 degree centrality, Q13 betweenness centrality shortest path and Q14) in miliseconds. The mean values are 350.3 (Q9), 14,677,102.6 (Q13) and 1,118.5 ms (Q14).

in graphs. The algorithm used in *Neo4j* is said to be a variant of Dijkstra's algorithm, which has a time complexity of $\mathcal{O}(|V| \cdot log|V| + |E|)$ [22]. For the given query, the algorithm runs very fast on the existing graph. However, as shown in the source above, the runtime grows tremendously with graphs less favourable to the algorithm, so the result obtained here should be viewed with caution. Finally, the problem becomes clear with the third algorithm. According to [29], Betweenness Centrality has a running time of $\mathcal{O}(|V|^3)$, but this can be improved with Brandes' algorithm to a time complexity of $\mathcal{O}(|V| \cdot |E|)$. According to the documentation of *Neo4j*, this is also the algorithm used there (cf. [30]). The query executed here, as shown by the average and the exact values in Fig. 10, ran for several hours before coming to a result. This not only stands in stark contrast to the other two algorithms, especially the one for shortest paths, but also renders them almost unusable for practical applications.

D. Import

As introduced above, we proposed two different algorithmic approaches to import the data and generate interoperable data. For testing purposes, we created three different test sets: Two small data sets with 1,300 and 36,000 records as well as a large set with 135,000 data records.

See Fig. 11 for a detailed runtime overview. The optimized approach using sets is faster and thus more competitive for large data sets.

VII. Conclusion and Outlook

Here, we presented a novel approach that annotates clinical research data with contextual information. The result is a knowledge graph representation of data, the context graph. It contains computable statement representation. We discuss the impact of this novel approach using 14 real-world use cases and graph queries. This graph allows to compare research data records from different sources as well as the selection of relevant data sets using graph-theoretical algorithms. See Fig. 12 for an illustration of Alzheimer's data points.

This proof of concept of a biomedical knowledge graph combines several sources of data by relating their contextual

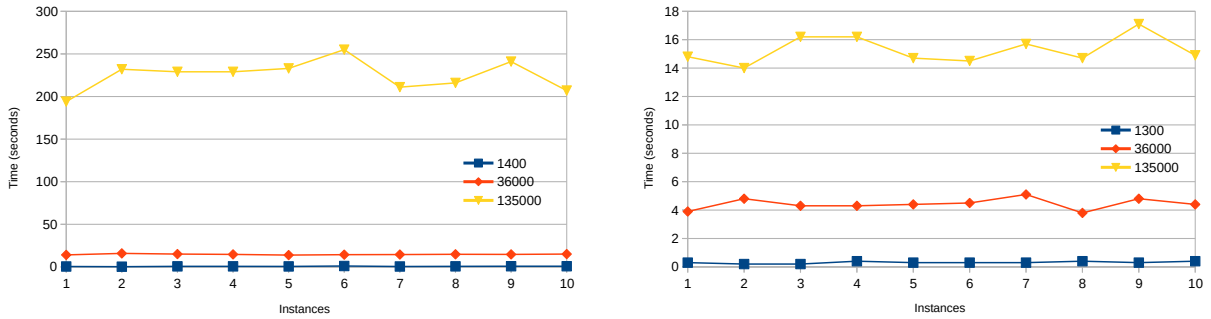

Fig. 11. Runtime of INTEROPERABLE-DATA, see Algorithm 1, with lists (left) and sets (right) with different import data size (1300–135000 data points). The speedup factor is between 2 (for small instances) and 13 (for large instances).

data to one another. We processed data from clinical research, biomedical publications and presented a generic and efficient approach towards interoperable data.

Furthermore, we discussed the runtime analysis of 14 real-world use cases represented by graph queries. As stated in previous works and discussed in our paper, performance for some semantic queries remains a major problem due to the massive latency for requesting detailed data points. Thus, the next step is to integrate the results presented in [22] in our information systems to improve the practical execution times for those and similar queries.

Storing and querying a giant knowledge graph as a labeled property graph is still a technological challenge. Here, we demonstrate how our data model is able to support the understanding and interpretation of biomedical data, especially in the context of clinical trials. We presented several real-world use cases that utilize our massive, generated knowledge graph. To date, we restricted our work to some smaller subgraphs. We plan to integrate these graphs into larger knowledge graphs, for example interaction networks. That will improve this unique view on clinical data and its application combining methods of algorithmic optimisation, graph theory and data science.

Considering the integration of further related biomedical knowledge resources, a variety of highly specific, field-dependent databases appears available at hand. For example, imaging techniques such as DICOM files from radiology reports are a promising extension for the future. Files of this format offer far more than just the image data, but contain a variety of additional (meta-)information concerning the patient, the applied imaging techniques, the circumstances of the examination, affiliated publications and much more. These meta-data could be reorganized in form of a graph and then added to the already existing model. However, it is important to find a suitable sequence for importing data, avoiding node ambiguity, e.g. considering already existing patient IDs.

While our proof of concept is both functional and generic, extending the knowledge graph to further fields of research, e.g. towards genomic and pharmacologic information or demographic background data, is feasible and just a matter of modelling connectors to the relevant sources. Moreover, these

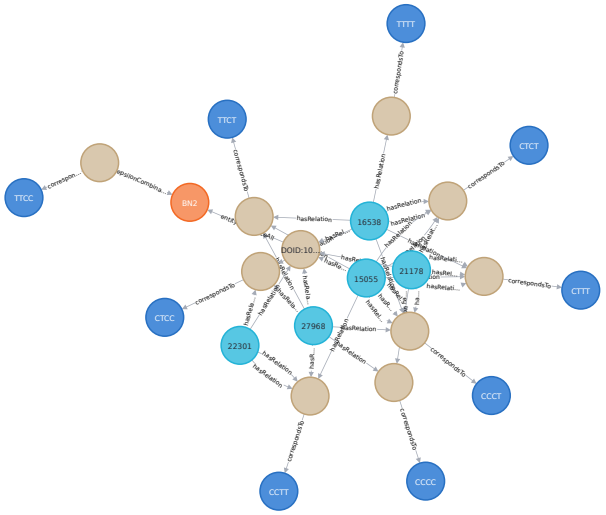

Fig. 12. Example nodes with output surrounding `preferredLabel:"Alzheimer's disease"`. It describes the data foundation for novel approach that annotates clinical research data with contextual information. The result is a knowledge graph representation of data, the context graph. It contains computable statement representation.

occasionally bulk data might be queryable on the fly through interfaces dynamically translating graph (sub-)queries into e.g. SQL. Despite considerable increases in expected running times, such partially distributed approaches might be favorable over integrated, warehouse-like solutions provisioning giant, largely unused graphs. However, efficient generation of remote queries and the re-integration of their results would obviously be key.

REFERENCES

[1] "Integrative Daten-Semantik für die Neurodegenerationsforschung https://www.idsn.info/de/idsn.html," Juli 2020. [Online]. Available: https://www.idsn.info/de/idsn.html

[2] J. Dörpinghaus, A. Stefan, B. Schultz, and M. Jacobs. (2020) Towards context in large scale biomedical knowledge graphs. [Online]. Available: http://arxiv.org/abs/2001.08392

[3] C. S. Burns, R. M. Shapiro, T. Nix, J. T. Huber *et al.*, "Examining medline search query reproducibility and resulting variation in search results," *iConference 2019 Proceedings*, 2019.

[4] C. T. Fakhry, P. Choudhary, A. Gutteridge, B. Sidders, P. Chen, D. Ziemek, and K. Zarringhalam, "Interpreting transcriptional changes using causal graphs: new methods and their practical utility on public networks," *BMC bioinformatics*, vol. 17, no. 1, pp. 1–15, 2016.
[5] F. Belleau, M.-A. Nolin, N. Tourigny, P. Rigault, and J. Morissette, "Bio2rdf: towards a mashup to build bioinformatics knowledge systems," *Journal of biomedical informatics*, vol. 41, no. 5, pp. 706–716, 2008.
[6] D. S. Himmelstein, A. Lizee, C. Hessler, L. Brueggeman, S. L. Chen, D. Hadley, A. Green, P. Khankhanian, and S. E. Baranzini, "Systematic integration of biomedical knowledge prioritizes drugs for repurposing," *Elife*, vol. 6, p. e26726, 2017.
[7] L. Harland, "Open phacts: A semantic knowledge infrastructure for public and commercial drug discovery research," in *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 2012, pp. 1–7.
[8] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.
[9] A. Clark, C. Fox, and S. Lappin, *The handbook of computational linguistics and natural language processing*. John Wiley & Sons, 2013.
[10] H. Mirisaee, E. Gaussier, C. Lagnier, and A. Guerraz, "Terminology-based text embedding for computing document similarities on technical content," *arXiv preprint arXiv:1906.01874*, 2019.
[11] J. Dörpinghaus and A. Stefan, "Knowledge extraction and applications utilizing context data in knowledge graphs," in *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2019, pp. 265–272.
[12] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig *et al.*, "Gene ontology: tool for the unification of biology," *Nature genetics*, vol. 25, no. 1, p. 25, 2000.
[13] D. S. Wishart, Y. D. Feunang, A. C. Guo, E. J. Lo, A. Marcu, J. R. Grant, T. Sajed, D. Johnson, C. Li, Z. Sayeeda *et al.*, "Drugbank 5.0: a major update to the drugbank database for 2018," *Nucleic acids research*, vol. 46, no. D1, pp. D1074–D1082, 2017.
[14] K. Khan, E. Benfenati, and K. Roy, "Consensus qsar modeling of toxicity of pharmaceuticals to different aquatic organisms: Ranking and prioritization of the drugbank database compounds," *Ecotoxicology and environmental safety*, vol. 168, pp. 287–297, 2019.
[15] "SNPedia https://www.snpedia.com/index.php/APOE," Juli 2020.
[16] "UCUM- The Unified Code for Units of Measure http://unitsofmeasure.org," Juli 2020.
[17] J. Hastings, *The Gene Ontology Handbook*. Springer, 2017, ch. Primer on Ontologies, pp. 3–13.
[18] "Dublin Core Metadata Initiative https://www.dublincore.org/specifications/dublin-core/," Juli 2020.
[19] M. Gorelick and I. Ozsvald, *High Performance Python: Practical Performant Programming for Humans*. O'Reilly Media, 2014. [Online]. Available: https://books.google.de/books?id=bIZaBAAAQBAJ
[20] "Import in Neo4j https://neo4j.com/docs/operations-manual/current/tools/neo4j-admin-import/," Juli 2020.
[21] J. Dörpinghaus and A. Stefan, "Knowledge extraction and applications utilizing context data in knowledge graphs," in *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2019, pp. 265–272.
[22] J. Dörpinghaus and A. Stefan, "Optimization of Retrieval Algorithms on Large Scale Knowledge Graphs," 2020.
[23] P. T. Wood, "Query Languages for Graph Databases," *SIGMOD Rec.*, vol. 41, no. 1, pp. 50–60, apr 2012. [Online]. Available: http://doi.acm.org/10.1145/2206869.2206879
[24] A. O. Mendelzon and P. T. Wood, "Finding Regular Simple Paths in Graph Databases," *SIAM Journal on Computing*, vol. 24, no. 6, pp. 1235–1258, 1995. [Online]. Available: https://doi.org/10.1137/S009753979122370X
[25] P. T. Wood, "Query Languages for Graph Databases," *SIGMOD Rec.*, vol. 41, no. 1, p. 50–60, Apr. 2012. [Online]. Available: https://doi.org/10.1145/2206869.2206879
[26] F. Grando, D. Noble, and L. C. Lamb, "An analysis of centrality measures for complex and social networks," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.
[27] C. Chen, W. Wang, and X. Wang, "Efficient maximum closeness centrality group identification," in *Australasian Database Conference*. Springer, 2016, pp. 43–55.
[28] M. Fink and J. Spoerhase, "Maximum betweenness centrality: approximability and tractable cases," in *International Workshop on Algorithms and Computation*. Springer, 2011, pp. 9–20.
[29] M. Fink, "Zentralitätsmaße in komplexen Netzwerken auf Basis kürzester Wege," Master's thesis, Julius-Maximilians-Universität Würzburg: Institut für Informatik, 2009.
[30] "Neo4j Betweenness Centrality https://neo4j.com/docs/graph-data-science/current/algorithms/betweenness-centrality/," Juli 2020.