Towards Flexible Indices for Distributed Graph Data: The Formal Schema-level Index Model FLuID

Till Blume
ZBW – Leibniz Information Centre for Economics
Christian-Albrechts-Universität zu Kiel
Kiel, Germany
tbl@informatik.uni-kiel.de

Ansgar Scherp
ZBW – Leibniz Information Centre for Economics
Christian-Albrechts-Universität zu Kiel
Kiel, Germany
asc@informatik.uni-kiel.de

ABSTRACT

Graph indices are a key to manage huge amounts of distributed graph data. Instance-level indices are available that focus on the fast retrieval of nodes. Furthermore, there are so-called schema-level indices focusing on summarizing nodes sharing common characteristics, i. e., the combination of attached types and used property-labels. We argue that there is not a one-size-fits-all schema-level index. Rather, a parameterized, formal model is needed that allows to quickly design, tailor, and compare different schema-level indices. We abstract from related works and provide the formal model FLuID using basic building blocks to flexibly define different schema-level indices. The FLuID model provides parameterized simple and complex schema elements together with four parameters. We show that all indices modeled in FLuID can be computed in $\mathcal{O}(n)$. Thus, FLuID enables us to efficiently implement, compare, and validate variants of schema-level indices tailored for specific application scenarios.

Keywords

linked data, schema-level indices, formal model

1. INTRODUCTION

Summarizing data can help to efficiently manage huge amounts of data, and for many application scenarios indices are available that fit the specific information need [11]. For graph data, we can distinguish between instance-level indices and schema-level indices. Instance-level indices focus on the fast retrieval of nodes or answering queries regarding reachability, distance, and shortest path [17]. For example, they can be queried to search for metadata about a book by its title "Towards a clean air policy" [21]. Schema-level indices (SLIs) focus on summarizing nodes sharing common characteristics, i. e., the combination of attached types and used property-labels. Thus, SLIs support the efficient execution of structural queries, e. g., searching for bibliographic metadata using the type *bibo:book* [3]. For example,

30th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 22.05.2018 - 25.05.2018, Wuppertal, Germany.

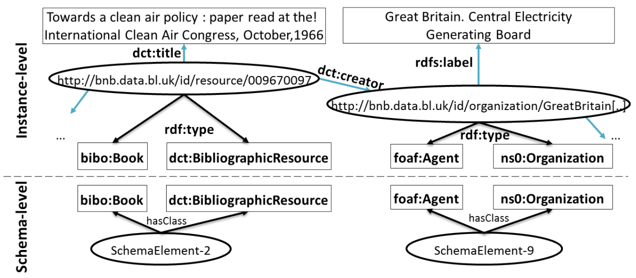
Towards a clean air policy : paper read at the! International Clean Air Congress, October,1966
Great Britain. Central Electricity Generating Board
Instance-level
dct:title
rdfs:label
dct:creator
http://bnb.data.bl.uk/id/resource/009670097
http://bnb.data.bl.uk/id/organization/GreatBritain[..]
rdf:type
rdf:type
bibo:Book
dct:BibliographicResource
foaf:Agent
ns0:Organization
Schema-level
hasClass
hasClass
SchemaElement-2
SchemaElement-9

Figure 1: A bibliographic metadata record provided by the British National Library (instance-level information) and a type-only schema structure (schema-level information).

instead of indexing the instance-level information illustrated in Fig. 1, a simple schema-level index would only memorize the combined use of the types, e. g., *bibo:Book* and *dct:BibliographicResource*. With such SLIs, search systems like LODeX [1, 2] and LODatio [10] support their users in finding and exploring data sources. In the past, different SLIs have been developed for different purposes that lack a common formalization and thus compatibility and comparability [9, 14, 16, 4, 13, 2, 20, 19]. We firmly believe the future development and further research on this topic can benefit from a common formal model. Thus, we conduct an in-depth study of existing approaches. We abstract from the related work and provide the formal model FLuID (Formal schema-Level Index model for the web of Data) consisting of basic building blocks to flexibly define SLIs.

The remainder of the paper is organized as follows: In Sect. 2, we discuss the related work. In Sect. 3, we show how equivalence relations can model any SLI. Subsequently in Sect. 4, we define our building blocks as equivalence relations, i. e., schema elements and their parameterizations. In Sect. 5, we analyze the space and build-time complexity of indices defined with FLuID and in Sect. 6.1 we conduct an empirical evaluation to support the analysis. We then outline a processing pipeline as well as a search prototype in Sect. 6.2, before we conclude in Sect. 7.

2. RELATED WORK

Schema-level indices (SLIs) support to efficiently execute structural queries over distributed graph data. Structural queries focus on how nodes (resources) are described, i. e., which combinations of types and properties are used to model the resources. There are various different possibilities and variants of how to define an SLI and different definitions of SLI allow for capturing different schemas. In the following, we present an overview of SLIs with emphasis on their schema structure, their application scenario, and how they were formalized.

Characteristic Sets [16] summarize instances along common incoming properties and outgoing properties. They were defined as sets of instances using a first-order-logic expression over triples. They were evaluated with respect to the accuracy of cardinality estimations for queries in RDF databases. SemSets [4] are defined as sets of instances that share the same outgoing properties which are connected to a common target resource. They were defined as sets over their own "Property Graph Data Model". They were developed to discover semantically similar sets in knowledge graphs in order to improve keyword-based ad-hoc retrieval. Christodoulou et al. [3] applied a hierarchical clustering algorithm on RDF data in order to determine clusters, i. e., sets of instances that are characterized by the same set of properties. They were defined as sets in an textual definition. The clusters are annotated using the RDF type information of the clustered instances, which is then used to derive a schema from the data sources on the Web of Data. ABSTAT [20] and LODeX [1, 2] summarize instances based on a common set of RDF types and properties linking to resources with the same set of types. ABSTAT's schema structure was informally defined as triples in a textual description. Additionally, ABSTAT selects a minimal number of types from the set of types such that all remaining types are sub-classes of the selected types. LODeX's schema structure was defined using a new grammar for their own model. LODeX uses a clustering of RDF types to select a representative type. Thus, they can comprehensively visualize several datasets hosted on DataHub. TermPicker [19] summarizes instances based on a common set of types, a common set of properties, and a common set of types of all property-linked resources. The schema structure was informally introduced by examples. The goal was to make data-driven recommendations of vocabulary terms.

One of the first SLIs using bisimulation is DataGuides [9, 14]. Bisimulation operates on state transition systems and defines an equivalence relation over states [18]. Two states are considered equivalent (or bisimilar) if they change into equivalent states with the same type of transition. Interpreting a labeled graph as a representation of a state transition system allows for the application of bisimulation on RDF data in order to discover structurally equivalent parts in the graph. Thus, DataGuides [9, 14] summarizes instances for which all outgoing paths for the whole subgraph are equivalent. DataGuides were evaluated on relational database systems using the Object Exchange Model (OEM). Since then, several SLIs adapted the idea of bisimulation and applied a stratified k-bisimulation on RDF and OEM [15, 13]. A stratified k-bisimulation is a bisimulation where the maximum length of the considered path is k edges long [18]. Another example is SchemEX [13], that summarizes instances similar to ABSTAT and LODeX, based on a common set of types and properties linking to resources with a common set of types. However, it does not perform any selection of types for the purpose of cluster labeling.

All SLIs presented above define a single, fixed schema structure. Considering the primary focus of the paper, the schema structures are often defined informally in a textual description or only explained by examples [4, 3, 20, 19]. One investigated SLI is defined using a mathematical model, which, however, was designed not for the SLI directly but rather the surrounding context [2]. Only few indices are defined using basic first-order-logic [9, 16, 13], which could be reused. However, to the best of our knowledge, there is only a single parameterization of an SLI suggested by Tran et al. [22]. They model a label-parameterized and height-parameterized index. With label-parameterization, only specific properties are considered and height-parameterization limits the maximum path-length of the subgraphs stored in the index. In summary, there exists no single, fully parameterized model which formally describes SLIs in general and which can be reused in order to develop, compare, and validate SLIs.

3. EQUIVALENCES OVER GRAPH DATA

A data graph G is defined by $G \subseteq V_{UB} \times P \times (V_{UB} \cup L)$, where V_{UB} denotes the set of URIs and blank nodes, P the set of properties, and L the set of literals. A triple is a statement about a resource $s \in V_{UB} \cup P$ in the form of a subject-predicate-object expression $(s, p, o) \in G$. Moreover, the properties P can be divided into disjoint subsets $P = P_{type} \dot{\cup} P_{rel}$, where P_{type} contains the properties denoting type information and P_{rel} contains the properties between instances in the data graph. If not stated otherwise, P_{type} only contains *rdf:type* and P_{rel} all $p \in P \setminus P_{type}$.

As discussed in Sect. 2, SLIs summarize instances based on their schema, i. e., common use of types and properties. For SLIs, we can distinguish between *abstract schema level* and the *entity mapping level* [7]. In our context, the abstract schema level defines the schema given by the index definition, e. g., taking only properties into account. We call these the *Schema Elements*. The entity mapping level is a concrete assignment of an instance to such an Schema Element. We will call Schema Elements with instances mapped to them *Instantiated Schema Elements*.

Each instance uses a defined set of types and properties and thus exactly one schema. Therefore, the mapping of instances to Instantiated Schema Elements is unique. SLIs partition the data graph into disjoint subsets of instances, where each subset is described by an Instantiated Schema Element. Equivalence relations can describe this graph partitioning in a formal way [8].

Definition 1 (Equivalence Relation). *For a given set X, an equivalence relation on X is a subset $EQR \subseteq X \times X$, that is reflexive, symmetric, and transitive. When $(x, y) \in EQR$, we say that x is equivalent to y or $x \sim y$. For any $y \in X$, the subset of X of all x that are equivalent to y is called the equivalence class of y, denoted $[y]_{EQR}$.*

Any two equivalence classes are either disjoint or coincide. This means that any equivalence relation on X defines a partition (decomposition) of X, and vice versa [8]. Furthermore, it can be shown that the intersection of two equivalence relations over X is also an equivalence relation.

In order to ensure the correctness of the approach, we formally define instances as equivalence relation over the data graph G. With instances being defined as equivalence relation any equivalence relation on top of instances consequently will be an equivalence relation over the data graph.

Definition 2 (instance). *Instances are sets of triples in the data graph G sharing a common subject URI. The* *equivalence relation $I \subseteq G \times G$ is defined as $((i_1, p_1, o_1), (i_2, p_2, o_2)) \in I \Leftrightarrow i_1 = i_2$. We write $[i]_I$ or I_i to denote the equivalence class of the instance with subject URI i.*

This definition of an instance maps each triple in G to exactly one instance determined by its subject URI. Thus, Def. 2 defines a partition over the data graph G and consequently qualifies as an equivalence relation [8]. In the context of SLI, we call equivalence classes of instances the schema elements. We connect the schema elements to generated instance information by using the notion of payload [10]. The payload comprises information about the actual data, e. g., all instances or only references to their data source. In summary a SLI can be defined over the data graph G, an equivalence relation EQR, and an n-tuple of payload functions PAY.

DEFINITION 3 (SCHEMA-LEVEL INDEX (SLI)). *Formally, a schema-level index is a 3-tuple (G, EQR, PAY), where G is the data graph which is indexed, EQR is an equivalence relation over instances in G, and PAY is an n-tuple of payload functions, which map instance information to equivalence classes in EQR.*

4. FLuID'S BUILDING BLOCKS

The FLuID model consists of basic building blocks, which can be combined to define any SLI. We have simple and complex schema elements, which can be further specialized with our four parameterizations. This section is organized into two parts: first we define simple schema structures and then we continue with complex schema structures.

4.1 Simple Schema Structures

We start by defining a simple schema element called Object Cluster. Object Clusters partition the data graph by mapping instances based on a common set of neighboring objects. Please note that the definition qualifies as equivalence relation since it is reflexive, symmetric and transitive.

DEFINITION 4 (OBJECT CLUSTER OC). *Object Clusters partition the data graph by mapping instances $[i_1]_I$ and $[i_2]_I$, based on a common set of triples where only the object is considered. The equivalence relation OC is defined as follows: $([i_1]_I, [i_2]_I) \in OC \Leftrightarrow \forall(i_1, p_1, o_1) \exists(i_2, p_2, o_2) : o_1 = o_2 \wedge \forall(i_2, p_2, o_2)\ \exists(i_1, p_1, o_1) : o_1 = o_2$*

The Object Cluster summarizes instances, not taking any property information into account. This can be changed using our first parameterization, the label parameterization lp, which allows ignoring a certain set of properties.

DEFINITION 5 (LABEL PARAMETERIZATION lp). *The label parameterization is a function $lp(EQR, P_r)$, which takes as input an equivalence relation EQR and a set of properties $P_r \subseteq P$ and returns an equivalence relation EQR_{P_r}. The returned equivalence relation EQR_{P_r} is a restriction of EQR in terms that all assertions about the triples in EQR only need to be true iff the property of the triple is included in the parameter property set P_r.*

Restricting any schema element with such a property set in fact relaxes the constraints given by the schema element. For example, the label parameterization lp applied on the Object Cluster OC using the properties P_{type} summarizes instances which have the same set of resources connected over the property rdf:type. This means any other object is not relevant to determine the equivalence. Please note, any label parameterized schema element still qualifies as equivalence relation since the same principle as before applies.

To sufficiently cover all SLIs we need more schema elements in FLuID. Therefore, we can analogously define two further simple schema elements called Property Cluster (PC) and Property-Object Cluster (POC). The PC summarizes instances based on the same properties ($p_1 = p_2$) and the POC based on the same property-object tuples ($p_1 = p_2 \wedge o_1 = o_2$). The Property-Object Cluster is sufficient for a schema structure defined by SemSets [4] since it compares objects and properties combined.

So far, our schema elements OC, PC, and POC only take outgoing properties into account. However, schema structures like Characteristic Sets [16] consider also incoming properties. To address incoming properties, an undirected version of the three simple schema elements can be defined by additionally considering the incoming triples $(x, p, i) \in G$ with i as the subject of the instance being in object position. We omit the formal definition of all undirected schema elements and only present the undirected Property Cluster $u\text{-}PC$ as an example. $([i_1]_I, [i_2]_I) \in u\text{-}PC \Leftrightarrow ([i_1]_I, [i_2]_I) \in PC \wedge \forall(x_1, p_1, i_1) \in G\ \exists(x_2, p_2, i_2) \in G : p_1 = p_2$ (and vice versa). The undirected Property Cluster $u\text{-}PC$ resembles the schema structure of Characteristic Sets [16].

4.2 Complex Schema Structures

The simple schema elements introduced above summarize instances by comparing incoming and outgoing triples of an instance. However, some SLIs like SchemEX [13], TermPicker [19], ABSTAT [20], and LODeX [1, 2] define schema structures that go beyond the scope of a single instance. The simple schema elements are already combinations of equivalence relations by using the identity equivalence "=" on properties and objects. We define complex schema elements as an extension of simple schema elements.

DEFINITION 6 (COMPLEX SCHEMA ELEMENT CSE). *A complex schema element partitions the data graph by summarizing instances based on three given equivalence relations $\sim^s$, $\sim^p$, and $\sim^o$. It can be described as 3-tuple $CSE := (\sim^s, \sim^p, \sim^o)$. Two instances $[i_1]_I, [i_2]_I$ are considered equivalent, iff $i_1 \sim^s i_2 \wedge p_1 \sim^p p_2 \wedge o_1 \sim^o o_2$ holds true for all triples in both instances.*

Example 1. We demonstrate the benefit of complex schema elements by defining $CSE\text{-}1 := (PC, T, PC)$ and $CSE\text{-}2 := (PC, =, PC)$, with T being an arbitrary tautology. Since T considers all properties equal, the Property Cluster in object position of CSE-1 considers sets properties. In contrast, CSE-2 uses the identity equivalence on predicate position, thus, all 2-hop property paths have to match exactly. The two instances $[i_3]_I$ and $[i_4]_I$ with outgoing properties as illustrated in Fig. 2 are considered equal according to the equivalence of CSE-1 since the 1-hop properties are equal and the 2-hop properties are equal. However, according to CSE-2, they are not considered equal, since the property paths are not identical.

We apply the same concept to model TermPicker [19]:

$$(lp(OC, P_{type}) \cap lp(PC, P_{rel}), lp(=, \emptyset), lp(OC, P_{type}))$$

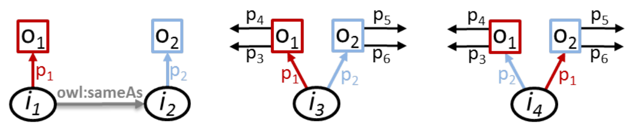

Figure 2: Sample data graph which is summarized to either three Object Clusters or one instance parameterized Object Clusters using SameAs Instances $[I]_\sigma$.

To model TermPicker, we make use of the intersection of the label parameterized Object Cluster and the label parameterized Property Cluster. This way, instances need to have the same type sets and the same Property Cluster. The independence of the objects' type sets and the connecting properties can be achieved using $lp(=, \emptyset)$ as predicate equivalence $\sim^p$ in the complex schema element. The identity equivalence on the empty set is a tautology. The schema of ABSTAT [20], LODeX [1, 2], and SchemEX [13] is defined straigth forward:

$$(lp(OC, P_{type}), lp(=, P_{rel}), lp(OC, P_{type}))$$

To generalize the concept of taking multiple neighboring instances into account, we define the chaining parameterization cp. As suggested by Tran et al. [22], we parameterize regarding the maximal path length of the subgraph structure. As illustrated in Fig. 2, the complex schema element can consider the neighborhood of up to two hops. When chaining k schema elements, the pattern is recursively applied up to k hops.

Definition 7 (Chaining Parameterization cp). *The chaining parameterization is a function $cp(CSE, k)$, which takes a complex schema element $CSE := (\sim^s, \sim^p, \sim^o)$ and a chaining parameter $k \in \mathbb{N}$ as input and returns an equivalence relation CSE_k. Formally, this chaining of k complex schema elements up to length k can be recursively defined as bisimulation [18]. Two instances $[i_1]_I$ and $[i_2]_I$ are equivalent according to $cp(CSE, k)$ if three conditions hold true: (1) For $k = 0$ the subject equivalence $i_1 \sim^s i_2$. (2) For $k > 0$ all three equivalences $i_1 \sim^s i_2 \wedge p_1 \sim^p p_2 \wedge o_1 \sim^o o_2$. (3) For $k > 0$ the recursion step $([o_1]_I, [o_2]_I) \in cp(CSE, k-1)$.*

4.2.1 Support for Unions of Instances

FLuID supports a parameterization of the instance definition which allows considering instances that resemble the same real-world entity by using the *owl:sameAs* property. In order to take this information into account, we formally introduce SameAs instances.

Definition 8 (SameAs Instance). *The equivalence relation σ summarizes instances based on the semantics of owl:sameAs in equivalence classes $[I]_\sigma$, called SameAs instance. For all instances $[i_1]_I, [i_2]_I \in [I]_\sigma$, there is a path over all edges (independent of the edges direction) labeled owl:sameAs in G from i_1 to i_2.*

Furthermore, it can be shown, that the assignment of an instance to a SameAs Instance is unique, by reducing the problem to finding weakly connected components in an *owl:sameAs*-labeled subgraph of G, as is has been done by Ding et al. [6]. With the notion of σ, we can now define the instance parameterization to consider SameAs instances instead of single instances.

Definition 9 (Instance Parameterization ip). *The instance parameterization is a function $ip(EQR, \sigma)$, which extends any simple or complex schema element EQR to additionally consider all connected instances following the instance equivalence relation σ. The returned equivalence relation EQR_σ is an extension of EQR, which restricts the triples to be in $[I]_\sigma$.*

As an example, we apply the instance parameterization ip on the Object Cluster equivalence relation OC using the SameAs instances: $(I_1, I_2) \in ip(OC, \sigma) \Leftrightarrow \forall(i_1, p_1, o_1) \in [I_1]_\sigma\ \exists(i_2, p_2, o_2) \in [I_2]_\sigma : o_1 = o_2$ (and vice versa).

As the example shows, the instance parameterized OC considers the SameAs network [6] and thus merges instances. Fig. 2 shows an example graph. According to the Object Cluster definition, the instances $[i_1]_I$, $[i_2]_I$, and $[i_3]_I$ are not equivalent. Summarizing $[i_1]_I$ and $[i_2]_I$ to a SameAs instance $[I]_\sigma$ leads to the equivalence of all three instances.

4.2.2 Support for Ontology Inferencing

In the Web of Data, there are assertions about individuals and assertions about RDF types and properties [5]. For example, a dataset can contain the following assertions:

<http://bnb.data.bl.uk/doc/resource/009670097> **<dct:creator>**
<http://bnb.data.bl.uk/id/organization/GreatBritain[..]> .
<dct:creator> **<rdfs:domain>** <bibo:Document> .
<dct:creator> **<rdfs:range>** <foaf:Person> .

The triples using *rdfs:domain* and *rdfs:range* allow inferring additional knowledge about individuals using the property *dct:creator*. The schema summarization tool ABSTAT [20] incorporates information derived from an ontology by inferring triples based on a subtype schema graph. ABSTAT's schema graph is constructed by extracting the contained schema assertions. We extend the idea of the schema graph from ABSTAT but include all RDFS properties in the schema graph. Thus, our RDFS schema graph contains hierarchical dependencies of *rdfs:subClassOf* and *rdfs:subPropertyOf* in a tree structure with further cross connections regarding *rdfs:range* and *rdfs:domain*.

Definition 10 (Schema Graph). *Let $SG := (V_C \cup P, \mathcal{E})$ be an edge-labeled directed multigraph and $\mathcal{E} \subseteq (V_C \cup P) \times (V_C \cup P)$. The set of nodes is the union of the set of RDF classes and properties. The edge-label function $\phi : \mathcal{E} \to P$ assigns labels from a given set of possible properties P to all edges $e \in \mathcal{E}$.*

We construct the RDFS schema graph by extracting all triples containing RDFS vocabulary terms, namely all properties $P_{RDFS} = \{$*rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:range*, *rdfs:domain*$\}$ and label the schema graph using the RDFS edge-label function ϕ_{RDFS}. In the following, we denote the schema graph constructed using the labeling function ϕ_{RDFS} with SG_{RDFS}. Having the hierarchically dependencies of types and properties represented using a Schema Graph, additional triples can be inferred if possible, e. g., when a property p_1 is used in the data graph and exists as node in the schema graph.

Definition 11 (Inferencing Parameterization). *The inferencing parameterization is a function $\Phi(G, SG)$, which takes any data graph G and schema graph SG as input and based on the entailment rules defined in the schema graph SG returns a data graph G_Φ, which additionally includes all inferred triples.*

5. SPACE AND TIME COMPLEXITY

In this section, we analyze the complexity of the computation process of SLIs defined with FLuID. To this end, we conduct a space and time complexity analysis of the computation process with a particular focus on the impact of the parameterizations. To compute an SLI, the instance data needs to be mapped to instantiated schema elements. This can be done, for example, by extracting all properties of an instance to form a Property Cluster. For instances using the same properties, the same schema element is computed. This can be efficiently implemented using hash maps, which ensure constant time access. As discussed in Sect. 4, each SLI defined with FLuID can be described as a combination of parameterized simple schema elements using parameterized complex schema elements. Simple schema elements can check the equivalence of two instances without considering neighboring instances. Since instances partition the data graph (Def. 2) and schema elements partition instances, for each triple in the data graph one operation for each simple schema element is needed.

Schema Elements. We denote with c the number of simple schema elements given by the concrete SLI definition using FLuID. Thus, without parameterizations, we have linear space and time complexity in the order of $\mathcal{O}(c \cdot n)$, with c simple schema elements in the definition and n triples in the data graph. Please note, the undirected schema elements are an exception, since considering incoming properties produces an overlap of triples. The incoming property of instance $[i_1]_I \in G$ is the outgoing property of another instance $[i_2]_I \in G$. Thus, for undirected schema elements, we may have to consider each triple twice.

Label Parameterization. The label parameterization reduces the number of considered objects and properties for each simple schema element by restricting the properties p to be in the set P_r (Def. 5). Considering all excluded properties $P \setminus P_r$ and that each property can occur more than one time in the dataset, we can define a constant $l \geq |P \setminus P_r|$, which denotes the number of occurrences of excluded properties in the dataset. Thus, the space complexity is still linear, but we can find a lower upper bound $\mathcal{O}(c \cdot (n - l))$. The time complexity is unchanged.

Instance Parameterization. The instance parameterization aggregates instances and thus does not impact the overall size of the index. Aggregating instances to unions can be done in constant time like triples are aggregated to instances in constant time using hash maps. Thus, the time complexity remains unchanged.

Inference Parameterization. The inferencing parameterization requires additional space to store all inferred types and properties. According to our definition of schema graph construction (Sect. 4.2.2), types and properties are only added to the schema graph, if there exists a triple in the dataset using a property in P_{RDFS}. We can assume that we have a limited number of such schema triple $s << n$ in the schema graph compared to the data graph size n. Furthermore, the complexity depends on the number of additional triples $g \leq s$ that can actually be inferred for each triple in the data graph. For example, we have two triples $(s_1, p_1, o_1) \in G$ and $(s_2, p_2, o_2) \in G$ with p_1 in the schema graph and p_2 not in the schema graph. Then, only for property p_1, for example, all super-properties can be fetched by following the *subPropertyOf* relations, e. g., $\{p_3, p_4\}$. Thus, we have an upper bound for space complexity using the inference parameterization in the order of $\mathcal{O}(c \cdot (n - l) \cdot g)$. Please note, with a linear dependency $g = f(n)$, we would end up with a quadratic complexity. In the worst case, we extract a fully connected (complete) schema graph. That means, for each indexed triple, all s possible triples in the schema graph are inferred. Furthermore, the worst case requires all triples in the data graph to use properties from P_{RDFS}. This is unrealistic for real-world datasets.

In our experiment described in the subsequent section, we use two datasets. From processing these datasets, we know that the smaller dataset has 2.0% RDFS properties and the larger dataset has 0.6% RDFS properties. This leads to a factor of $g < 1.001$. Thus, it appears safe to assume that there is no linear dependency of g and n and that the complexity is not quadratic.

The schema graph can be implemented using hash maps, which guarantees constant time for lookup and addition operations. Inferencing operations are linear in the number of inferrable types and properties. Thus, we have the same time complexity as for the space complexity. Furthermore, all triples s in the schema graph should be excluded from the index using the label parameterization, which would increase the number of excluded triples l.

Chaining Parameterization. The chaining parameterization defines the instance's neighborhood up to a maximum path length of k. Thus for each instance, we need to store up c^k instantiated simple schema elements. The definition of complex schema elements allows avoiding the computation of c^k simple schema elements for each instance. For each instance, c simple schema elements need to be computed. When considering the instance's neighborhood up to a maximum path length of k, the c computed simple schema elements for each instance can be reused.

Overall Complexity. The space complexity of the index is in the order of $\mathcal{O}(c^k \cdot (n - l) \cdot g)$ and the overall time complexity is in the order of $\mathcal{O}(c \cdot k \cdot n \cdot g)$ with c simple schema elements defined in the SLI, the chaining parameter k, l excluded properties in the data graph using the label parameterization, and g inferrable triples using the inference parameterization as constant factors independent of n. Thus, indices defined with FLuID can be computed in linear time and space with respect to the number of triples n.

6. EVALUATION AND PROTOTYPE

6.1 Empirical Evaluation

We empirically evaluate the RDFS schema graph SG_{RDFS} of two crawled datasets to support our claims regarding the parameters s and g in our complexity analysis from Sect. 5. We compare the number of statements s in SG_{RDFS} to the number of triples n in the dataset G. Second, we count how many additional statements can be inferred using SG_{RDFS}. To his end, we compare the number of all additional properties and types that could be inferred to the number of triples in the dataset n to average the parameter g.

Datasets. We use two crawled datasets of the Web of Data with different characteristics. The TimBL-11M dataset contains about 11 Million triples [13]. The crawl was conducted with a breadth-first search starting from the FOAF profile of Tim Berners-Lee. Regular snapshots from the Web of Data are provided by the Dynamic Linked Data Observatory (DyLDO) [12]. We use their first snapshot containing about 127 Million triples crawled from about $95,000$ seed URIs. This crawl was done with a breadth-first search but limited to a crawling depth of two [12].

Empirical Evaluation Results. The schema graph has a size of about 2.0% of the TimBL-11M dataset and 0.6% of the DyLDO-127M dataset. In the TimBL-11M dataset on average 1.7 additional properties and 4.8 additional types were inferred. For the DyLDO-127M dataset, on average 2.1 additional properties and 13.5 additional types were inferred. Furthermore, for only about 15% of the triples in both datasets, the inferencing operation was necessary. For the remaining triples, there was no corresponding entry in the schema graph. Rather than having for each triple all possible triples inferred, as in the worst case suggests, we measured that on average it is only about 0.008. Thus, the factor g for the space and time complexity can be estimated as a small constant factor $g < 1.001$.

6.2 Towards a FLuID Prototype

We use FLuID to index the distributed graph data of the Web of Data. LODatio+ (`http://lodatio.informatik.uni-kiel.de/`) is a search engine to find relevant data sources given a structural query. However, the index LODatio+ currently uses is tailored for one specific application need. As depicted in Fig. 3, we are implementing FLuID in a generic processing pipeline and are updating LODatio+ to understand any defined index following the FLuID model. LODatio+ is already an extension of LODatio [10], but performing generic queries on any FLuID-index is ongoing work.

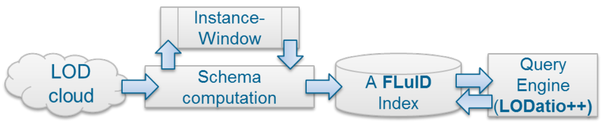

Figure 3: Approach to a generic computation pipeline for FLuID and a generic query engine LODatio++.

7. CONCLUSION

We have presented the novel, parameterized schema-level index model FLuID which is sufficient to express the functionalities of existing SLIs and beyond. We showed that the time and space complexity of any SLI developed with FLuID scales linear with respect to the number of triples indexed. Implementing FLuID in a single computation- and query-framework as well as qualitatively comparing existing and new approaches is ongoing work.

8. ACKNOWLEDGMENTS

This research was co-financed by the EU H2020 project MOVING (`http://www.moving-project.eu/`) under contract no 693092.

9. REFERENCES

[1] F. Benedetti, S. Bergamaschi, and L. Po. Online index extraction from linked open data sources. In *LD4IE*, 2014.

[2] F. Benedetti, S. Bergamaschi, and L. Po. Exposing the underlying schema of LOD sources. In *Joint IEEE/WIC/ACM WI and IAT*, 2015.

[3] K. Christodoulou, N. W. Paton, and A. A. A. Fernandes. Structure inference for Linked Data sources using clustering. In *Joint EDBT/ICDT*, 2013.

[4] M. Ciglan, K. Nørvåg, and L. Hluchý. The SemSets model for ad-hoc semantic list search. In *WWW*, 2012.

[5] G. De Giacomo and M. Lenzerini. TBox and ABox reasoning in expressive description logics. In *AAAI Technical Reports*, 1996.

[6] L. Ding, J. Shinavier, Z. Shangguan, and D. L. McGuinness. SameAs networks and beyond: Analyzing deployment status and implications of owl:sameAs in Linked Data. In *ISWC*, 2010.

[7] R. Q. Dividino, A. Scherp, G. Gröner, and T. Grotton. Change-a-lod: Does the schema on the linked data cloud change or not? In *COLD*, volume 1034 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[8] European Mathematical Society. Equivalence relation. `http://www.encyclopediaofmath.org/index.php?title=Equivalence_relation&oldid=35990`, 2014.

[9] R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *VLDB*, 1997.

[10] T. Gottron, A. Scherp, B. Krayer, and A. Peters. LODatio: using a schema-level index to support users infinding relevant sources of linked data. In *K-CAP*, 2013.

[11] K. Hose, R. Schenkel, M. Theobald, and G. Weikum. *Database Foundations for Scalable RDF Processing*, pages 202–249. Springer Berlin Heidelberg, 2011.

[12] T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, and A. Hogan. Observing linked data dynamics. In *ESWC*, volume 7882, 2013.

[13] M. Konrath, T. Gottron, S. Staab, and A. Scherp. SchemEX - efficient construction of a data catalogue by stream-based indexing of Linked Data. *J. Web Sem.*, 16:52–58, 2012.

[14] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: a database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[15] S. Nestorov, J. D. Ullman, J. L. Wiener, and S. S. Chawathe. Representative Objects: Concise representations of semistructured, hierarchial data. In *ICDE*, 1997.

[16] T. Neumann and G. Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *ICDE*, 2011.

[17] S. Sakr and G. Al-Naymat. Graph indexing and querying: a review. *Int. Journal of Web Information Systems*, 6(2):101–120, 2010.

[18] D. Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4):15:1–15:41, 2009.

[19] J. Schaible, T. Gottron, and A. Scherp. TermPicker: Enabling the reuse of vocabulary terms by exploiting data from the Linked Open Data cloud. In *ESWC*, 2016.

[20] B. Spahiu, R. Porrini, M. Palmonari, A. Rula, and A. Maurino. ABSTAT: ontology-driven Linked Data summaries with pattern minimalization. In *ESWC Satellite Events, Revised Selected Papers*, 2016.

[21] T. Tran, P. Haase, and R. Studer. Semantic search — using graph-structured semantic models for supporting the search process. In *Int. Conf. on Conceptual Structures*, pages 48–65. Springer, 2009.

[22] T. Tran, G. Ladwig, and S. Rudolph. Managing structured and semi-structured RDF data using structure indexes. *IEEE TKDE*, 25(9):2076–2089, 2013.