Modelling Dynamics in Semantic Web Knowledge Graphs with Formal Concept Analysis

Larry González
Center for Advancing Electronics Dresden (cfaed)
TU Dresden, Germany
larry.gonzalez@tu-dresden.de

Aidan Hogan
Center for Semantic Web Research
DCC, Universidad de Chile
ahogan@dcc.uchile.cl

ABSTRACT

In this paper, we propose a novel data-driven schema for large-scale heterogeneous knowledge graphs inspired by Formal Concept Analysis (FCA). We first extract the sets of properties associated with individual entities; these property sets (aka. *characteristic sets*) are annotated with cardinalities and used to induce a lattice based on set-containment relations, forming a natural hierarchical structure describing the knowledge graph. We then propose an algebra over such schema lattices, which allows to compute diffs between lattices (for example, to summarise the changes from one version of a knowledge graph to another), to add diffs to lattices (for example, to project future changes), and so forth. While we argue that this lattice structure (and associated algebra) may have various applications, we currently focus on the use-case of modelling and predicting the dynamic behaviour of knowledge graphs. Along those lines, we instantiate and evaluate our methods for analysing how versions of the Wikidata knowledge graph have changed over a period of 11 weeks. We propose algorithms for constructing the lattice-based schema from Wikidata, and evaluate their efficiency and scalability. We then evaluate use of the resulting schema(ta) for predicting how the knowledge graph will evolve in future versions.

CCS CONCEPTS

• **Information systems → Semantic web languages**; *Graph-based database models*;

KEYWORDS

Semantic Web, Schema, Knowledge Graph, Dynamics, FCA

ACM Reference Format:
Larry González and Aidan Hogan. 2018. Modelling Dynamics in Semantic Web Knowledge Graphs with Formal Concept Analysis. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.3186016

1 INTRODUCTION

Graph-based data models [5] have become increasingly common in data management scenarios that require flexibility beyond what is offered by traditional relational databases. Such flexibility is particularly important in Web scenarios, where potentially many users may be involved (either directly or indirectly) in the creation, management and curation of data, where data may be incomplete, properties may have multiple values, and the data schema may be subject to frequent change. This need for flexibility has given rise to the adoption of graph-based models for various applications, including Facebooks's Open Graph Protocol, Google's Knowledge Graph, schema.org, and so forth. In other applications, users may further have control over the schema, allowing not only to edit nodes and edges in the graph, but also to define new *types* of nodes and edges; an example of such a scenario is the Wikidata knowledge graph [39] – hosted by the Wikimedia Foundation and seen as a source of data to compliment Wikipedia – where users can add new properties and types that can be used to define further data.

While graphs enable increased levels of flexibility in terms of how a given data collection is managed and curated, on the flip-side, this flexibility comes with the inevitable cost of higher levels of heterogeneity, where involved entities may be defined in diverse ways, data may have various levels of (in)completeness, etc. Conceptually understanding the current state of a knowledge graph – in terms of what data it contains, what it is missing, how it can be effectively queried, what has changed recently, etc. – is thus a major challenge: it is unclear how to distil an adequate, high-level description that captures an actionable overview of knowledge graphs.

We thus need well-founded methodologies to make sense of knowledge graphs, where an obvious approach is to define some notion(s) of *schema* for such graphs. The traditional approach in the Semantic Web has been what Pham and Boncz [31] call the *schema first* approach: define the schema that the data should follow. The most established language for specifying *semantic schemata* is RDF Schema (RDFS) [9], which allows for defining the semantics of terms used in the RDF [37] graph-based model; however, such an approach does not help to understand the data that an RDF graph contains since defined terms need not be used and further undefined terms may be used in such data. More recently, *validating schemata* – such as the Shapes Constraint Language (SHACL) [27] – have been proposed that allow for defining various constraints that compliant RDF graphs must follow; however, the purpose of such schemata is to constrain and validate graphs rather than to gain an understanding of the legacy data contained in a given graph.

An alternative to the *schema first* approach is the *schema last* approach [31], which foregoes an upfront schema and rather lets the data evolve naturally; thereafter, the goal is to understand what the legacy graph data contain by extracting high-level summaries that characterise the graph, resulting in a *data-driven schema*. Due to a growing realisation that traditional notions of schema are not enough, various works have emerged on this topic, trying to extract implicit structure from – and ultimately make sense of – diverse RDF graphs [1, 2, 10–14, 19, 20, 25, 31, 32, 36]. Such works consider various applications, be it to help users write queries, to build browsing interfaces, to optimise query processing, to identify abstract topics covered, to model topological changes, etc.

In this paper, we propose yet another approach to compute a data-driven schema from such graphs; more specifically, our approach is inspired by formal concept analysis (FCA) and produces a lattice of "concepts" based on the properties (outgoing edge labels) for all entities in the graph (also known as *characteristic sets*). A key novelty of our approach is to propose an FCA-style framework that can be applied to very large, diverse, graph-structured knowledge-bases. To validate the utility of the FCA-based schema extracted by this framework, as our use-case, we study the problem of summarising the dynamics of a dataset and of predicting future high-level changes. To address this use-case, we propose a novel abstract algebra over FCA-style lattices that allows for computing diffs between two such schemas (through a subtraction operator) and adding such diffs to given schemata in order to project future schema-level changes (through an additional operator).

We apply this framework to compute lattices for 11 versions of the Wikidata knowledge graph, evaluating their suitability for the use-case of predicting future, high-level changes. We select Wikidata as: (1) it provides a history of weekly versions that we can use for evaluating predictions, (2) it is edited by thousands of users, meaning that significant changes are observed week-to-week, (3) the scale and diversity of the dataset offer (to the best of our knowledge) an unprecedented challenge for FCA-style techniques, requiring novel methods. Our results show that the proposed framework can scale to datasets like Wikidata and that it can provide better predictions than a baseline method using a linear model.

Contributions: Our main contributions are as follows: (1) We propose a notion of formal context and concepts for applying FCA-style techniques to RDF graphs. (2) To improve scalability, we propose using an intermediary lattice that does not materialise the full lattice but rather allows for the concepts to be lazily computed (as needed). (3) We propose an algebra for (a) computing a high-level diff between two versions of an RDF graph based on our lattice structures, and (b) adding lattices to predict future changes. (4) We evaluate our methods by extracting the lattices for 11 weekly versions of the Wikidata knowledge graph, presenting performance and scalability results, and assessing the quality of predictions.

Paper outline: Section 2 presents related work in the areas of data-driven schemata, FCA techniques and Semantic Web dynamics. Section 3 presents preliminaries relating to RDF and FCA. Section 4 presents our framework for extracting lattices from RDF graphs, for which Section 5 discusses concrete algorithms. Section 6 describes an algebra for computing diffs and predicting future changes in lattices. Section 7 presents our evaluation before Section 8 concludes.

2 RELATED WORKS

We now provide an overview of the most pertinent related works in the areas of data-driven schemata for RDF, FCA on the Semantic Web, and modelling dynamics in knowledge graphs.

Data-driven RDF schemata: A variety of works have proposed methods to summarise, profile and/or compute schemata from RDF graphs (as opposed to defining an *upfront* schema for RDF graphs, per the RDFS [9] and SHACL [27] standards). A common approach is to compute a *graph summary* based on various notions of *quotient graphs* [12], which first define an equivalence relation on nodes in the input graph, where each node partition induced by the relation is then considered a node in the quotient graph; such equivalence relations can be defined in terms of, e.g., bisimulations [10, 14, 32, 36], node types [11, 19, 20, 25], isomorphism [12], and so forth. An interesting property of such quotient graphs is that they can (often) preserve some notion of the connectivity of the original graph.

Further approaches rather consider extracting a meta-data summary – such as a *VoID description* [4] – from the graph [8, 23, 29, 34]; however, such approaches tend to extract statistical descriptions rather than inherent structures from the data (though VoID's dataset partitions [4] do capture some notion of structure).

Other approaches for computing inherent structures from an RDF dataset are based on clustering [1], latent topic analysis [7], association rule mining [2], *n*-ary relations [31], prototypes [13], formal concept analysis [6, 16, 22], and more besides. The approach we propose falls into the latter category, applying formal concept analysis to RDF graphs; we now discuss such works in more detail.

FCA on the Semantic Web: Our proposal is inspired by methods proposed in the Formal Concept Analysis (FCA) community [33, 40]. In fact, we are far from the first authors to consider applying FCA techniques to a Semantic Web context, where amongst such works we can mention the proposal by Rouane-Hacene et al. [35] for Relational Concept Analysis (RCA), where FCA is applied individually to entities of different types to create a concept lattice for each type; the work by Alam et al. [3] on applying FCA to help explore and assess the completeness of Linked Datasets; the evaluation of Kirchberg et al. [26] for the performance of FCA algorithms applied to Linked Datasets; as well as works by Formica [21] and d'Aquin and Motta [15] for facilitating search and question answering applications over Semantic Web datasets. However, while some of these papers do deal with datasets similar to our own (e.g., DBpedia), all of the papers we have observed apply FCA over closed subsets of datasets, typically including a subset of entities of a particular type. For example, in the performance-focussed paper of Kirchberg et al. [26], the largest datasets considered contain in the order of 35,000 entities, whereas we consider an FCA-style analysis over full (truthy) Wikidata, which describes tens of millions of entities.

Broadening the search to more general FCA methods at large scale, we could find works by Xu et al. [41] and Krajca and Vychodil [28] that (like us) propose to use the distributed MapReduce framework to enhance the scalability of the FCA process; however, the largest dataset considered by Xu et al. [41] contains in the order of 100,000 entities, while the largest considered by Krajca and Vychodil [28] contains in the order of 33,000 entities—still orders of magnitude below our target scale. Hence, at least to the best of our knowledge, no work has considered applying FCA over a dataset as diverse and large as Wikidata; in fact, as we will discuss later, typical FCA methods require adaptations to scale to such levels.

Modelling Dynamics on the Semantic Web: Our main use-case for applying FCA over Wikidata is to model the dynamic behaviour of the dataset and predict future changes. Thus within our related works, we can consider works relating to the modelling of changes in Semantic Web knowledge graphs. Within this area, we can consider, for example, the work by Umbrich et al. [38], who define various types of entity- and document-level changes in Linked Data, looking to see if such changes follow a Poisson distribution. Later work by Käfer et al. [24] proposed the Dynamic Linked Data Observatory to collect weekly snapshots of Linked Data crawled from the Web; analysing various aspects of the dynamics of datasets, they classify websites by the types of changes observed, be they bulk changes, continuous changes, or simply static datasets. The data collected by Käfer et al.'s observatory was later used in follow-up work by, e.g., Dividino et al. [18] for improving cache maintenance. To the best of our knowledge, however, no work has attempted to *predict* high-level changes in such datasets; rather the focus of such work has been on modelling and analysing historical dynamics.

3 PRELIMINARIES

In order to present a formal framework for the paper, we focus on the RDF data model. However, the techniques and results developed herein generalise to other graph-structured data models [5].

RDF terms and graphs: RDF is a graph-structured model based on three disjoint sets of terms: IRIs ($\mathbf{I}$), literals ($\mathbf{L}$) and blank nodes ($\mathbf{B}$). Claims involving these terms can be organised into *RDF triples* $(s, p, o) \in \mathbf{IB} \times \mathbf{I} \times \mathbf{IBL}$,[1] where s is called *subject*, p is called *predicate*, and o is called *object*. An *RDF graph* G is then a finite set of RDF triples, where a triple $(s, p, o) \in G$ can be viewed as an edge of the form $s \xrightarrow{p} o$ in a directed edge-labelled graph. The terms used in the predicate position are referred to as *properties*. We use the term *entity* to refer to the real-world objects referred to by the subjects of the graph. Given an RDF graph G, for $\bullet \in \{\text{S}, \text{P}, \text{O}\}$, we denote by $\pi_\bullet(G)$ the projection of the set of terms appearing in a particular triple position in G; e.g., $\pi_\text{S}(G) := \{s \mid \exists p, o : (s, p, o) \in G\}$.

Formal contexts and concepts: Formal concept analysis (FCA) is a methodology for extracting a concept hierarchy from sets of entities and their properties [40]. More specifically, the methodology is based on extracting *formal concepts* from *formal contexts*. A *formal context* is a triple $X = (E, A, I)$, where E is a set of entities,[2] A is a set of attributes, and $I \subseteq E \times A$ is the *incidence*: a set of pairs such that $(e, a) \in I$ if and only if the attribute a is defined for entity e.

Towards defining *formal concepts*, we give some initial definitions. Given a formal context $X = (E, A, I)$, for a subset of entities $F \subseteq E$, let $[\![F]\!]_X := \{a \in A \mid \forall f \in F : (f, a) \in I\}$; conversely, for a subset of attributes $B \subseteq A$, let $[\![B]\!]_X := \{e \in E \mid \forall b \in B : (e, b) \in I\}$. Thus, for a set of entities, $[\![\cdot]\!]$ takes the set of attributes they all share in common, while for a set of attributes, $[\![\cdot]\!]$ takes the set of entities they all share in common. A formal concept is then a pair (F, B) where: (1) $F \subseteq E$, (2) $B \subseteq A$, (3) $[\![F]\!]_X = B$, *and* (4) $F = [\![B]\!]_X$. In the formal concept (F, B), the set F is called the *extent* of the concept while the set B is called the *intent* of the concept.

In terms of inducing a concept hierarchy, let (F_1, B_1) and (F_2, B_2) be two formal concepts for the formal context $X = (E, A, I)$. We define the partial order $\leq$ based on set containment of intent such that $(F_1, B_1) \leq (F_2, B_2)$ iff $B_1 \subseteq B_2$. Letting C denote the set of all formal concepts in X, then $(E, [\![E]\!]_X)$ serves as the bottom context denoting the attributes that all entities share, while $([\![A]\!]_X, A)$ serves as the top concept ($\top$) denoting the entities using all attributes; since for any $c \in C$ it holds that $\bot \leq c \leq \top$, we can say that $(C, \leq)$ forms a complete lattice, known as the *concept lattice*. We remark that $[\![E]\!]_X$ and $[\![A]\!]_X$ can be the empty set in practice, and that $[\![A]\!]_X$, in particular, will very often be empty. Also we note that the same characteristics could be achieved by considering a dual partial order based on set containment of the entities in the extent; however, herein we will be concerned with the attribute-based order. Furthermore, it will be useful to consider a non-transitive version of the $\leq$ order wrt. C, which we denote by $\lessdot$, such that $c \lessdot c''$ iff $c < c''$ and there does not exist $c' \in C$ such that $c < c' < c''$.

Characteristic sets: In the section that follows, we will outline a (rather natural) notion of formal context for RDF graphs based on *characteristic sets*, which were first proposed by Neumann and Moerkotte [30] in the context of query optimisation (more specifically, for cardinality estimation). The characteristic set of an RDF term $s \in \mathbf{IBL}$ in an RDF graph G is defined as the set of properties associated with that subject s in G; more formally, $\text{cs}(G, s) := \{p \mid \exists o : (s, p, o) \in G\}$. The characteristic sets of the graph G are then defined as the set of characteristic sets for all subjects in G; more formally, overloading notation, $\text{cs}(G) := \{\text{cs}(G, s) \mid s \in \pi_\text{S}(G)\}$.

4 FCA FOR RDF GRAPHS

We now discuss a general method by which FCA can be used to extract a data-driven schema – in the form of a formal concept hierarchy – from an RDF graph. We begin with a general definition that instantiates a formal context in a natural way from an RDF graph. However, the concept lattice resulting from such a definition is not practical to compute at scale and hence we propose increasingly minimal structures that should be more feasible to compute.

4.1 RDF FC-Lattice

An intuitive instantiation of FCA for RDF is given by constructing a formal context $X = (E, A, I)$ from an RDF graph G considering the subject terms in G to be the entities ($E := \pi_\text{S}(G)$), the properties in G to be the attributes ($A := \pi_\text{P}(G)$), and the incidence to be given by the use of that property as a predicate on the given subject ($I := \{(s, p) \mid \exists o : (s, p, o) \in G\}$). The notion of a formal concept in such a setting then follows naturally from the definition of X.

Example 4.1. Consider the following example RDF graph G (in Turtle syntax) containing five subjects and four properties.

```
ex:UT ex:name "U Thurman"; ex:star ex:Gattaca .
ex:GO ex:name "G Orwell"; ex:writer ex:1984 .
ex:AK ex:name "A Kurosawa"; ex:director ex:Ikiru, ex:Ran .
ex:PD ex:name "PK Dick"; ex:writer ex:Ubik, ex:Valis .
ex:CE ex:name "C Eastwood"; ex:director ex:Sully;
   ex:star ex:Unforgiven, ex:Tightrope ;
```

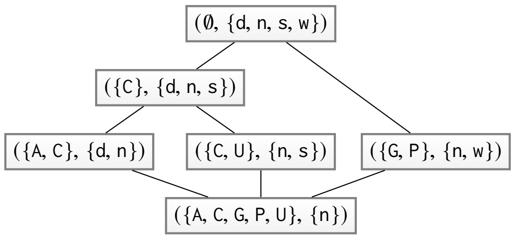
We can consider the formal context $X = (E, A, I)$ of this RDF graph as the following matrix (often known as a *cross table* in the FCA literature) with the row leader denoting A, the column header denoting E , and the matrix ticks denoting I:

[1] We use, e.g., $\mathbf{IB}$ as a shortcut for $\mathbf{I} \cup \mathbf{B}$.
[2] In the FCA literature, it is more typical to refer to a set of *objects*; we avoid this nomenclature since it clashes with the notion of an object in an RDF triple.

	ex:AK	ex:CE	ex:GO	ex:PD	ex:UT
ex:director	✓	✓			
ex:name	✓	✓	✓	✓	✓
ex:star		✓			✓
ex:writer			✓	✓	

Let d, n, s and w denote properties by their initial and A, C, G, P and U denote subjects likewise by their initial. Within this matrix, the maximal projections of incidence sub-matrices filled with ✓ are then considered to be formal concepts. For example, ({A}, {d, n}) is not considered a formal concept since it can be extended by the C column maintaining a dense sub-matrix; on the other hand ({A, C}, {d, n}) is a formal concept since it cannot be extended by any row or column while keeping the projected sub-matrix full. Likewise ({G, P}, {w}) is not considered a formal concept since it can be extended by row n to create the formal concept ({G, P}, {n, w}).

Along these lines, one can verify that the formal context representing G has six formal concepts C. We can draw the corresponding lattice $(C, \leq)$ as the following *Hasse diagram*, where lines denote only direct inclusions (i.e., $(C, \preceq)$) and the top concept $\top$ is drawn at the top of the diagram with lesser concepts then descending:



Here we maintain attribute subsets with the same cardinality on the same "level" where direct inclusions may skip levels as shown for the inclusion between {n, w} and {d, n, s, w}.

Intuitively, the idea is that this lattice represents a concept hierarchy distinguishing sets of entities based on the properties by which they are defined; for example, we can see concepts in the lattice relating to directors, actors, writers, and director–actors. We call this the *formal concept lattice* or *FC lattice* for short. □

While the previously defined notion of a formal context and formal concepts for RDF are quite intuitive, there are a variety of potential practical problems to address with the FC lattice.

To start with, for a formal context $X = (E, A, I)$, the upper bound on the number of formal concepts is $\min(2^{|E|}, 2^{|A|})$, bounded by the cardinality of the powerset of entities and attributes (whichever is smaller since the same subset of attributes or entities cannot appear twice). The bound is tight considering, for example, a context where $E = A = \{1, ..., n\}$ and where $I = \{(e, a) \mid e \neq a\}$; now each pair (F, B) such that $F \cap B = \emptyset$, $F \cup B = \{1, ..., n\}$ is a formal concept, generating the 2^n powerset of concepts (both in extent and intent). However, under the hypothesis that many combinations of properties – such as ex:capital and ex:director – are unlikely to ever occur on a single subject in practice, we can speculate that such exponentiality is unlikely to be encountered in real RDF graphs (though this will require empirical support).

More problematically in practice, the size of individual formal concepts can be prohibitively large, especially with respect to the inclusion of subjects in each such concept: in most RDF graphs the number of unique subjects will far surpass the number of unique properties. In the Wikidata knowledge graph, for example, there are millions of subjects, with each concept being potentially of length $|\pi_S(G)| + |\pi_P(G)|$ (e.g, measured in bits) and where each subject s in the graph G can be contained in potentially $2^{|\mathrm{cs}(G,s)|}$ concepts.

For such reasons, given a large-scale dataset as input, from even an initial inspection, it may not be practical to materialise the FC concept lattice. A number of approaches have been developed to deal with this issue by reducing the dimensionality of the concept lattice (creating what is sometimes called an *iceberg lattice*) by pruning attributes or entities that are rare, or grouping attributes or entities that frequently coincide, and so forth (we refer to Section 5.5 of the survey by Poelmans et al. [33] for further details). We take a rather simpler strategy as described in the following section.

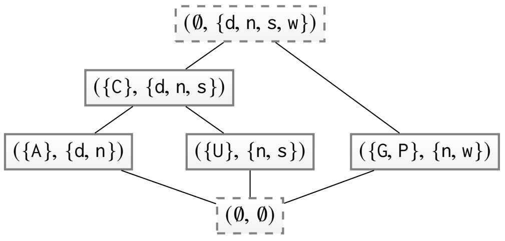
4.2 RDF CS-Lattice

To avoid materialising the entire FC lattice, we rather propose to materialise an intermediary structure from which the concept lattice, or parts there of, can be lazily materialised, as needed. The core intuition is to represent a non-transitive version of the FC lattice such that, for each concept, the intent corresponds precisely to the extent. We call this the characteristic set (CS) lattice since each concept refers to a characteristic set and its extension.

More specifically, given a formal context $X = (E, A, I)$, let $I(e) := \{a \in A \mid (e, a) \in I\}$ denote the attributes of entity $e \in E$. We say that (F, B) is a CS concept of X if (1) $F \subseteq E$, (2) $B \subseteq A$, (3) for all $e \in F$, it holds that $I(e) = B$, and (4) for all $e \in E \setminus F$, it holds that $I(e) \neq B$. Equivalently, (F, B) is a CS concept of G iff B is a characteristic set of G and F is the set of all subjects in G with characteristic set B.

However, letting C denote the set of all CS concepts of X and considering the intent-based ordering $\leq$ as before, we must be careful: namely the partially ordered set $(C, \leq)$ is no longer a lattice since the previous top formal concept may not be a CS concept (if no subject uses all properties) while the bottom formal concept will never be a CS concept (since no subject has no properties). Hence to return to a complete lattice, we can create new top and bottom CS concepts to return to a complete CS lattice.

Example 4.2. Let us return to the FC lattice depicted in Example 4.1. The corresponding CS lattice is then:



We draw with a dashed line the virtual top and bottom CS concepts introduced to ensure the result is a lattice. Note that now, for example, the extent of {n, s} no longer contains C even though that entity is incident with both attributes: instead, each extent refers to the set of entities with *precisely* that intent. The CS lattice is thus, intuitively speaking, a "non-transitive" version of the FC lattice. □

The CS lattice has a number of practical benefits when compared with the previously defined FC lattice.

First, in many use-cases, it may be useful to group subjects by the exact set of properties that they are incident with. To give an intuition of such a case, we will later use these lattices to compute probabilistic predictions of how a particular subject will evolve in a future version of the RDF graph in terms of what properties are most likely to be added/deleted for that subject; here we need to analyse the evolution of other subjects with *precisely* that set of properties in observable historical data. For this, the CS lattice will be a better alternative than the corresponding FC lattice.

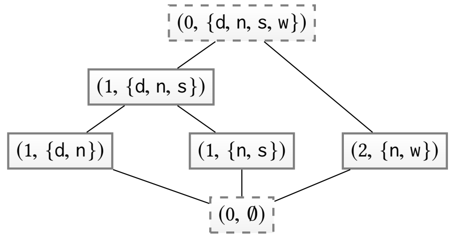
Second, the size of this CS lattice is now bounded by the number of subjects $|\pi_S(G)| + 2$ since only one extent can contain a particular subject (with 2 referring to the top and bottom concepts). This bound is tight if each subject is associated with a different characteristic set and no subject contains all properties. Intuitively, the CS lattice no longer contains "intermediary" concepts; for instance, in the previous example, while there was a formal concept associated with the intent {n}, there is no "strict" CS concept with that intent since no subject has precisely the set of properties {n}; if it were not needed as the bottom concept of the CS lattice, such a concept (and other such intermediary concepts) would not be included. Taking perhaps a better example, if we consider again the formal context $E = A = \{1, ..., n\}$, and $I = \{(e, a) \mid e \neq a\}$, the CS lattice will contain $n + 2$ concepts encoding precisely I and the required top $(\emptyset, A)$ and bottom $(\emptyset, \emptyset)$ concepts. Likewise, given that each subject appears in one extent, the average length of the CS concepts is greatly reduced (though the upper bound is not).

We highlight that the CS lattice directly encodes the incidence I of the formal context and (assuming all entities and attributes appear in the incidence) thus contains sufficient information to recompute the FC lattice, allowing to materialise formal concepts in a lazy manner—hence why we referred to the CS lattice as an "intermediary structure" at the outset of the section.

4.3 RDF #-Lattices

The number of subjects described by large-scale knowledge graphs such as Wikidata or DBpedia is often in the order of millions, while the number of properties rather tends to be in the order of thousands. Hence we can greatly reduce the overall (e.g., in-memory) size of the lattice by replacing the extents in each concept with their cardinality. In other words, given a lattice $(C, \leq)$, we define its #-lattice as $(C^\#, \leq)$ where $C^\# := \{(|F|, B) \mid (F, B) \in C\}$. This may be sufficient for a number of use-cases, such as for estimating the cardinalities of conjunctive queries [30]. We refer to such lattices as #-lattices, where the definition applies to either FC #-lattices or CS #-lattices; in the following, we exemplify the latter.

Example 4.3. The CS #-lattice corresponding to Example 4.2 is as follows (replacing the extent with its cardinality):



The dashed concepts represent the top and bottom concepts included to ensure the result is a complete lattice (other concepts with count 0 are excluded). The hierarchy remains the same. □

We highlight that #-lattices contain the same number of concepts as their full-extent versions; furthermore, the CS #-lattice contains sufficient information to recreate the FC #-lattice as needed.

4.4 Alternatives

We remark that the previous notions of lattices form a natural "base" for describing an RDF graph as part of a data-driven schema. However, one can consider a number of variations on this theme:

1. One could consider the properties on *objects* as forming a separate "inverse" lattice based on labels for inward edges, or potentially even combining both subject and object lattices into one by considering virtual inverse properties.
2. One could consider encoding the number of values that a given subject takes for a given property into the lattice, which would distinguish, for instance, `ex:GO` (with one value for `ex:writer`) from `ex:PD` (with two values).
3. One could consider including the *values* of certain (categorical) properties into the lattice, such as to capture the type of a particular entity, or its occupation, gender, etc.; this would lead towards the notion of a *many-valued context* [33].

While such variations and extensions would be interesting to investigate, we consider them as part of future work, with a particular challenge being to keep the size of the resulting lattice manageable.

5 COMPUTING LATTICES

We now present an overview of the methods we propose for computing the concept lattices previously described. Given a (potentially very large) RDF graph G, our strategy is as follows: (1) We first compute the CS concepts; given that here we must process the entire graph, we propose an algorithm based on the MapReduce framework to enable horizontal scaling. (2) We then compute the hierarchy over these CS concepts to generate the CS lattice; more precisely, we compute the direct containments and add the top and bottom elements, giving us the CS lattice for the RDF graph. (3) We do not directly materialise the FC lattices; rather these will be materialised as needed for a particular use-case.

5.1 Computing the CS concepts

We compute the characteristic sets from the RDF graph using an algorithm for the distributed *MapReduce framework* [17], which consists of two main phases: a *map* phase where sets of key–value pairs are assigned to machines based on their key, and a *reduce* phase, where values with the same key are grouped, aggregated and processed to produce an output on the local machine. Given an input RDF graph as a set of triples, the algorithm for computing CS concepts then consists of two high-level MapReduce tasks:

Task$_1$ takes as input the set of triples from G and runs:
- **Map$_1$** Each input triple (s, p, o) is mapped with key s and value p, thus emitting pairs of the form (s, p).
- **Reduce$_1$** For each key s, the pair $(s, \{p_1, \ldots, p_n\})$ is output where $\{p_1, \ldots, p_n\}$ is the set of all properties on s.

Task$_2$ takes as input the set of pairs from Task$_1$ and runs:

Map$_2$ Each input pair $(s, \{p_1, \ldots, p_n\})$ is mapped with key $[p_1, \ldots, p_n]$ and value s. In practice, we apply a lexical order on the properties in the key and concatenate them to produce a canonical key for each such set.

Reduce$_2$ For each key $[p_1, \ldots, p_n]$, all subjects are collected and the pair $(\{s_1, \ldots, s_m\}, [p_1, \ldots, p_n])$ is output, corresponding to a CS concept.

While conceptually straightforward, in practice we encountered a litany of errors in trying to run these tasks over Wikidata on rented clusters; in particular, we frequently encountered out-of-disk errors, expensively slow runtimes, load issues, and so forth. Hence we implemented and tested a number of improvements:

- Subjects and properties are compressed using numeric ids. This greatly reduces space and improves performance by allowing MapReduce to sort more data in memory, also producing much more succinct keys for the second task.
- We tested a variety of *combiners* – local reducers that take advantage of the commutativity of processing to reduce the number of key–value pairs that need to be sent over the network and processed on the reduce machines – for the first task that also boosted performance.
- We also experimented with varying number of machines.

Some brief details on performance will be provided in Section 7.

Algorithm 1 Computing the CS partial order $(C, \prec)$

```
1:  function POSET($C$)                                  ▷ $C$ a set of CS concepts
2:    $\mathcal{B} \leftarrow \{B \mid \exists F : (F, B) \in C\}$  ▷ we only need the CS intents (sets of props.)
3:    initialise $\olessthan$                    ▷ will store the direct containments: $\olessthan \subseteq \mathcal{B} \times \mathcal{B}$
4:    let $\mathcal{B}.n := \{B \in \mathcal{B} : |B| = n\}$            ▷ returns CSs on level $n$
5:    m $\leftarrow \max\{|B| : B \in \mathcal{B}\}$        ▷ the size of the largest characteristic set
6:    for $j = 2; j \leq$ m; $j$++ do
7:      for $i = j - 1; i > 0; i$−− do
8:        if $i = j - 1$ then
9:          $\olessthan \leftarrow \olessthan \cup$ REMOVEONE($\mathcal{B}.i$, $\mathcal{B}.j$)
10:       else
11:         $\olessthan \leftarrow \olessthan \cup$ RAREJOIN($\mathcal{B}.i$, $\mathcal{B}.j$, $\olessthan$)
12:   let $(F, B) \prec (F', B')$ if and only if $(B, B') \in \olessthan$
13:   return $(C, \prec)$
14: function REMOVEONE($\mathcal{B}_i$,$\mathcal{B}_j$)  ▷ for $B_j \in \mathcal{B}_j$, remove $p \in B_j$, see if set in $\mathcal{B}_i$
15:   initialise $\olessthan_{i,j}$
16:   for $B_j \in \mathcal{B}_j$ do
17:     for $p \in B_j$ do
18:       $B'_j \leftarrow B_j \setminus \{p\}$
19:       if $B'_j \in \mathcal{B}_i$ then
20:         $\olessthan_{i,j} \leftarrow \olessthan_{i,j} \cup \{(B_i, B_j)\}$
21:   return $\olessthan_{i,j}$
22: function RAREJOIN($\mathcal{B}_i$,$\mathcal{B}_j$,$\olessthan$)   ▷ check $\subset$ only for sets sharing rare property
23:   initialise $\olessthan_{i,j}$
24:   for $B_i \in \mathcal{B}_i$ do
25:     for $B_j \in \mathcal{B}_j : B_i[0] \in B_j$ do   ▷ for all $B_j$ containing rarest prop. of $B_i$
26:       if $B_i \subset B_j \wedge (B_i, B_j) \notin \olessthan^+$ then▷ if subset and not reachable in $\olessthan$
27:         $\olessthan_{i,j} \leftarrow \olessthan_{i,j} \cup \{(B_i, B_j)\}$
28:   return $\olessthan_{i,j}$
```

5.2 Computing the CS partial order

The next challenge is to compute the CS lattice based on the subset partial order of the set of CS concepts C computed in the previous stage; more specifically, we compute direct containments within C. We can then (trivially) add a top and bottom concept, as previously described, to compute the final CS lattice. In this phase, we assume that the *intents* (i.e., the characteristic sets themselves, not the lists of subjects) fit in memory since the partial order underlying the CS lattice only relies on the intents of the CS concepts. Indeed, as described later in the experimental section, although over 2 million unique characteristics sets are computed for Wikidata, with numeric compression, these fit in 16GB of memory without issue.

In order to compute the CS lattice from n characteristic sets, the simplest algorithm we could consider is to perform $\binom{n}{2}$ pairwise subset comparisons, but clearly this would not be practical for n >2,000,000, and likewise we would compute $(C, \leq)$ (i.e., all transitive containments) rather than $(C, \lessdot)$ (the direct containments).

Instead we adopt the approach outlined in Algorithm 1. Here we only consider the intents of the concepts: the characteristic sets themselves, denoted $\mathcal{B}$. We stratify these characteristic sets into levels based on their cardinality, where level i is the set of all characteristic sets with cardinality i (denoted $\mathcal{B}.i$). Note that as per Example 4.2, a direct containment may "skip" a level; hence we must check all pairs of levels. Starting with $j = 2$ and ending when $j = m$ (for m the max number of levels), we compare all characteristic sets on level j to all on levels $j-1$ to 1; in other words, we compare levels in the order $(\mathcal{B}.2, \mathcal{B}.1), (\mathcal{B}.3, \mathcal{B}.2), (\mathcal{B}.3, \mathcal{B}.1), \ldots, (\mathcal{B}.m, \mathcal{B}.1)$, which helps avoid returning indirect containments. For comparing two levels i and j (for $i < j$); we have two algorithms to choose from:

(1) When $i + 1 = j$, we invoke REMOVEONE, where from each characteristic set in $\mathcal{B}.j$, we remove a property and check if the result is in $\mathcal{B}.i$. We use an index to check membership in $\mathcal{B}.i$ where we then require $|\mathcal{B}.j| \times j$ lookups on that index.

(2) Otherwise we apply RAREJOIN where, for each characteristic set $B_i \in \mathcal{B}.i$, we find the rarest property $p \in B_i$ in terms of appearing in the fewest sets of $\mathcal{B}$ (choosing arbitrarily based on lexical order if tied), retrieve each $B_j \in \mathcal{B}.j$ that also contains p, and then check if $B_i \subset B_j$ and (B_i, B_j) is not already reachable in the current partial order; if so, we add the pair (B_i, B_j) to the partial order. Note that in a preprocessing step, all properties in each input characteristic set are ordered by rarest first, and we create an inverted index from properties to characteristic sets by level; hence finding all B_j matching the condition on Line 25 requires one lookup on the inverted index. While the upper-bound remains $|\mathcal{B}.i| \times |\mathcal{B}.j|$ set-containment checks, in practice, comparing only pairs of sets that share their rarest property should greatly reduce the number of comparisons from a brute-force method.

In terms of the condition for choosing one algorithm or the other, note that if we considered a generalised method REMOVEN for $n = j - i \leq N$, we would end up having to perform $\binom{j}{n}$ lookups on the $\mathcal{B}.i$ index, which would be problematic for $n \approx \frac{j}{2}$. Empirically we found that REMOVEONE was the only case faster than RAREJOIN.

5.3 Computing the lattices

Once we have the partially ordered set returned by Algorithm 1, to derive the final CS lattice, we need to compute the extent and the top and bottom concepts. Given that the extent (computed by the MapReduce framework) does not fit in-memory, we simply leave it indexed on-disk. To complete the CS lattice, we add the top concept $([\![A]\!]_X, A)$ for A the set of all properties and $[\![A]\!]_X$ the set of subjects with all properties; and the bottom concept $(\emptyset, \emptyset)$.

6 LATTICE DIFF-ALGEBRA

We could intuitively consider the FC/CS lattice as encoding the possible paths of evolution of entities in a knowledge graph: in a monotonic knowledge graph where properties are continuously added to entities (often the case for incomplete knowledge-graphs where new information is constantly being added), we could consider new entities as beginning at the bottom of the lattice and evolving towards the top of the lattice. Referring back to Example 4.2, for instance, we could consider new entities as first having `ex:name` defined, where they can then take a path towards being a director, an actor, or a writer; if already an actor or director, they may become an actor–director, and so forth. The cardinality of the extent likewise encodes information about the popularity of certain paths along which entities evolve. Of course, if the knowledge graph is not monotonic, entities may also descend the lattice as properties are removed. In any case, we can see the lattice as somehow encoding possible evolutions of an entity.

Taking this one step further, if we have the lattices for two different versions of a knowledge graph, we can apply a diff to see high-level changes between both versions of the data. Furthermore, given such a diff between two versions, we could further consider adding that diff to the most recent version to try predict future changes. We now capture precisely these intuitions with an algebra for computing diffs between lattices and adding diffs to lattices.

6.1 Defining CS-lattice diffs

Let $X_i = (E_i, A_i, I_i)$ and $X_j = (E_j, A_j, I_j)$ be formal contexts for two versions of an RDF graph (i being some version before j), and let $\mathcal{L}_i := (C_i, \leq)$ and $\mathcal{L}_j := (C_j, \leq)$ be the two corresponding CS lattices, where we remark that $\leq$ is defined for $C_i \cup C_j$ (being based on a general notion of set containment). Further let $E := E_i \cup E_j$ and $A := A_i \cup A_j$. We can define a lattice diff $\Delta_{j,i} \subseteq 2^A \times E \times 2^A$ as a set of triples denoting for each entity in E its intent in C_j and in C_i. More specifically, we say that $\Delta_{j,i} = \mathcal{L}_j - \mathcal{L}_i$ iff

$$\begin{aligned}\Delta_{j,i} = \{\, (B_j, e, B_i) \mid & (e \in E_i \cap E_j \text{ and } I_i(e) = B_i \text{ and } I_j(e) = B_j) \\ & \textit{or}\, (e \in E_i \setminus E_j \text{ and } I_i(e) = B_i \text{ and } B_j = \emptyset) \\ & \textit{or}\, (e \in E_j \setminus E_i \text{ and } B_i = \emptyset \text{ and } I_j(e) = B_j) \,\}\end{aligned}$$

Note that if e is a new entity ($e \in E_j \setminus E_i$), we mark it as *coming from* the bottom CS concept $(\emptyset, \emptyset)$ of $\Delta_{j,i}$, whereas if e is removed ($e \in E_i \setminus E_j$), we mark it as *going to* the bottom CS concept.

Example 6.1. At the top of Figure 1, we provide an example of the diff computed between two lattices, where $\mathcal{L}_1$ is the CS lattice previously introduced in Example 4.2 and $\mathcal{L}_2$ is taken as an example of how the lattice evolves in the next version of the dataset. The diff is then a directed edge-labelled graph where the nodes are the sets of characteristic sets and the edges are labelled according to the entities that move between the sets from version 1 to 2. □

As before, we can also consider a cardinality version of a diff $\Delta^{\#}_{j,i} \subseteq 2^A \times \mathbb{N} \times 2^A$ where instead of computing the entities that move between characteristic sets, we simply count the number of entities that move. Thus given $\Delta_{j,i}$, let $\Delta_{j,i}(B_j, B_i) := \{e : (B_j, e, B_i) \in \Delta_{j,i}\}$; now we can define $\Delta^{\#}_{j,i} := \{(B_j, n, B_i) : n = |\Delta_{j,i}(B_j, B_i)|\}$.

6.2 Predicting future #-lattices

Given two CS lattices $\mathcal{L}_1$ and $\mathcal{L}_2$ referring to two versions of an RDF graph, we could consider using the diff $\Delta_{2,1} = \mathcal{L}_2 - \mathcal{L}_1$ to predict a future version of the dataset through an operation such as $\mathcal{L}_{[3]} = \mathcal{L}_2 + \Delta_{2,1}$. However, such an operation would not make much sense since specific entities in $\Delta_{2,1}$ have already reached their destination. Instead we can consider predicting the CS #-lattice $\mathcal{L}^{\#}_{[3]}$ by defining the following algebraic operation: $\mathcal{L}^{\#}_{[3]} = \mathcal{L}^{\#}_2 + \Delta^{\#}_{2,1}$ (or in other words, $\mathcal{L}^{\#}_{[3]} = \mathcal{L}^{\#}_2 + (\mathcal{L}^{\#}_2 - \mathcal{L}^{\#}_1)$). More generally, given $\Delta^{\#}_{j,i} = \mathcal{L}^{\#}_j - \mathcal{L}^{\#}_i$, let $\mathcal{L}^{\#}_k$ be derived from a third version of the graph; we now wish to "add" the changes between the i^{th} and j^{th} versions to the k^{th} version to predict the $(k + j - i)^{\text{th}}$ version.[3] We will thus define the operation $\mathcal{L}^{\#}_k + \Delta^{\#}_{j,i}$ as producing a #-lattice $\mathcal{L}^{\#}_{k,j,i} := (C^{\#}_{k,j,i}, \leq)$ predicting $\mathcal{L}^{\#}_{[k+j-i]}$; we are left to define $C^{\#}_{k,j,i}$.

A natural idea is to sum the incoming entities and subtract the outgoing entities for each characteristic set between versions i and j and add that total to version k; for example, let us say that n entities move from some characteristic set $\{p, q\}$ in version i to $\{p, q, r\}$ in version j; then starting with $C^{\#}_k$, we could add n to the number of entities for $\{p, q, r\}$ and remove n from $\{p, q\}$ when computing $C^{\#}_{k,j,i}$. But what if $C^{\#}_k$ does not have n entities in the source characteristic set $\{p, q\}$ to "move" to $\{p, q, r\}$? Furthermore, what if more entities should move from $\{p, q\}$ to another set $\{p, q, s\}$?

To resolve such issues, rather than apply transitions in terms of absolute numbers of entities, we apply them in terms of the ratio of entities that move from the source characteristic set. Formally, first let $\mathcal{B}_i$, $\mathcal{B}_j$, $\mathcal{B}_k$ denote the characteristic sets in $C^{\#}_i$, $C^{\#}_j$, $C^{\#}_k$, let $C^{\#}(B)$ denote m such that $(m, B) \in C^{\#}$ (or 0 if no such value for m exists), and let $\Delta^{\#}_{j,i}(B_j, B_i)$ denote n such that $(B_j, n, B_i) \in \Delta^{\#}_{j,i}$ (or 0 if no such value for n exists). Next we define the ratio of entities of B_i moving to B_j as $\rho_{j,i}(B_j, B_i) := \frac{\Delta^{\#}_{j,i}(B_j, B_i)}{C^{\#}_i(B_i)}$ if $C^{\#}_i(B_i) \neq 0$; for convenience, we also define the ratio for characteristic sets not in $\mathcal{B}_i$ to indicate no change where, in such a case (i.e, where $C^{\#}_i(B_i) = 0$), if $B_i = B_j$ then $\rho_{j,i}(B_j, B_i) := 1$, otherwise $\rho_{j,i}(B_j, B_i) := 0$. Finally, we define $C^{*}_{k,j,i} := \{(B, \sigma(B)) \mid B \in \mathcal{B}_j \cup \mathcal{B}_k, B \neq \emptyset \text{ and } \sigma(B) > 0\}$, where $\sigma(B)$, in turn, is defined as:[4]

$$\sigma(B) := \text{round}\left(\sum_{B' \in \mathcal{B}_k \setminus \{\emptyset\}} \rho_{j,i}(B, B') \times C^{\#}_k(B') \right) + \Delta^{\#}_{j,i}(B, \emptyset)\,.$$

The summand $\Delta^{\#}_{j,i}(B, \emptyset)$ adds the absolute number of fresh entities (nowhere in version i) added to B in version j. Finally, we add top and bottom concepts to $C^{*}_{k,j,i}$ to generate a lattice $\mathcal{L}^{\#}_{k,j,i} = (C^{\#}_{k,j,i}, \leq)$: let $\mathcal{B}^{*}_{k,j,i}$ denote all characteristic sets in $C^{*}_{k,j,i}$ and $A^{*}_{k,j,i}$ their union; if $A^{*}_{k,j,i} \in \mathcal{B}^{*}_{k,j,i}$, we add only the bottom concept $(0, \emptyset)$; otherwise we add $(0, \emptyset)$ and the top concept $(0, A^{*}_{k,j,i})$.

Example 6.2. At the bottom of Figure 1, we provide an example of adding a #-diff to a #-lattice to predict the next #-lattice (with $\mathcal{L}^{\#}_2$

[3]In practice, this assumes versions with regular periodicity, e.g., weekly versions; often $k = j$ with both referring to the latest version from which predictions are made.
[4]round(·) denotes rounding towards positive infinity (applying ceiling for $\frac{2n+1}{2}$).

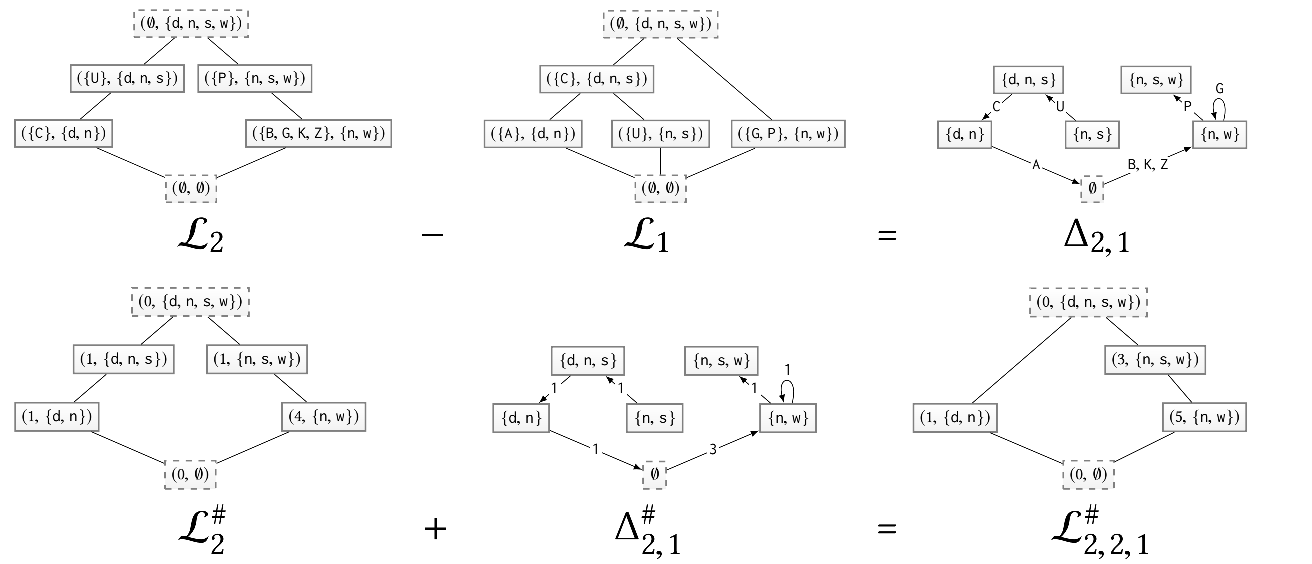


Figure 1: Computing a diff between two CS lattices and adding it to the most recent CS #-lattice to predict the next CS #-lattice

and $\Delta^{\#}_{2,1}$ based on the top of the figure). Take the case of {n, w}: in $\mathcal{L}^{\#}_2$ this characteristic set has 4 entities, of which, $\Delta^{\#}_{2,1}$ states that half (2) should stay in {n, w} while half (2) should go to {n, s, w}; furthermore, 3 fresh entities are defined for {n, w}; hence the predicted value for {n, w} is 5. Consider on the other hand {n, s, w}: in $\Delta^{\#}_{2,1}$ it has no outgoing edges since it was not present in $\mathcal{L}_1$, hence the one entity in $\mathcal{L}^{\#}_2$ remains and 2 are added from {n, w} as aforementioned; thus the predicted value is 3. Finally, we highlight that {d, n, s} is predicted empty: though $\Delta^{\#}_{2,1}$ suggests that entities should be added from {n, s}, no such entities are available in $\mathcal{L}^{\#}_2$, and the entity previously in {d, n, s} moves to {d, n} (while the previous entity in {d, n} is deleted, leaving one entity in {d, n}). □

These algebraic operations then allow to predict future high-level changes in the RDF graph where, in particular, the #-diff encodes a prediction on how entities will evolve and move between characteristic sets. This has various concrete use-cases: e.g., given a particular subject in G_2 – the second version of the dataset – we may wish to know the probability that it will change characteristic sets – either adding or removing unique incident properties – in the next version G_3, which we can compute based on $\Delta_{2,1}$ as described.

A natural generalisation of this idea is to consider the "transitive counts" of the ancestors of a characteristic set, where rather than considering a fixed subject, we consider the evolution of all subjects with (at least) a given characteristic set (in line with the original FC lattice). This is useful, for example, to predict how the results for a query on those properties might change in the next version. To compute such a prediction, we can simply take the predicted $\mathcal{L}^{\#}$ lattice and sum the non-overlapping counts of its ancestors.

Finally, note that where $n > 2$ past versions of the dataset are available, we may consider computing a *mean #-diff* by simply computing the $n - 1$ #-diffs possible and then taking the average of their transition values; the intuition here is to take the "mean" transition of entities across several pairs of versions, which may smooth the effect of bulk edits between a given pair of versions.

7 EVALUATION

The prior discussion raises a number of questions that can only be validated empirically; in particular, we are interested in addressing the following primary questions: (1) Can we compute the CS concepts at scale? (2) Can we efficiently compute the CS lattice? (3) How large is the CS lattice produced? (4) How accurately can our #-diffs predict future changes? Along these lines, we now present the results of experiments for the Wikidata knowledge graph.

Data. We consider the "truthy" RDF dumps of Wikidata – without qualifier information – spanning 11 weeks from 2017-04-18 to 2017-06-27. The first version has 1,102,242,331 triples, 54,236,592 unique subjects and 3,276 unique properties, while the final version has 1,293,099,057 triples (+17%), 57,197,406 unique subjects (+5%) and 3,492 unique properties (+6%). Hence we see that the dataset is growing, particularly in the volume of triples (with new triples often using existing properties on existing subjects).

Computing CS concepts. We use a Hadoop cluster with a single namenode and a varying number of datanodes. All machines had a 2.20GHz Xeon E5-2650 v4 CPU, 8GB of RAM and a 500G SSD. We used JDK 1.8.0_121, Apache Hadoop 2.7.3 and Apache Jena 3.2.0 for parsing. We ran a variety of experiments testing different combiner strategies, compression techniques, varying number of reducers, and so forth. For reasons of space, we do not present the full details of these experiments except to note that the fastest configuration involved processing data with numeric ID compression (more than halving the processing time including compression time on a single machine) and with a concatenation-based combiner, we save an additional 12.5% of computational time. In experiments with 4, 8, 16 and 32 machines, we found that after 8 machines, little gain in wall-clock computation time was observed, perhaps due to skew in the characteristic set distribution. For the largest dataset, numeric compression took 01:21:05 (HH:MM:SS), while **Task**$_1$ took 01:06:38 and **Task**$_2$ took 00:07:15; the total (wall-clock) time for computing the CS concepts was thus 02:34:58.

The total number of characteristic sets varied from 2,004,910–2,118,109 between the earliest and latest versions of Wikidata considered. The smallest characteristic sets contained one property, with the largest containing 148–154 properties across the versions; the median number of properties was 18 for all versions.

Computing CS lattices. We use a single machine for computing the CS lattice with a 2.5GHz Intel Core i7-6500U CPU, 16GB RAM and a 256 GB SSD. Using the strategy outlined in Algorithm 1, the runtimes for computing the lattice varied from 06:57:59–07:51:07 for the least-to-most recent version, with the number of edges varying from 78,046,423–86,848,506. This corresponds to a mean indegree (or equivalently outdegree) of ~39–41 edges in the CS lattice.

Quality of predicted #-lattices. Finally, we turn to testing the quality of the future #-lattices we predict. For this, we run experiments where we train on w previous weekly versions of the dataset to predict the next version of the #-lattice. Given that we have 11 versions, we train on $1 \leq w \leq 6$ versions to ensure at least 5 ($11 - w$) predicted lattices for each experiment. To measure the quality of the prediction, we compute the Root Mean Square Error (RMSE) and the Mean Average Error (MAE) between the predicted #-lattice and the real lattice. Note that RMSE = MAE indicates that prediction errors have consistent magnitude (e.g., each prediction is out by a constant factor $\pm n$), while RMSE $\gg$ MAE indicates that some errors have much larger magnitude than the average case (which we would expect given that some characteristic sets have much higher cardinality and much more dynamic behaviour than others).

In each case, we consider two algorithms: (1) a baseline algorithm that, independently for each CS in the (union of) the w previous #-lattices, applies a linear model (LM) – more specifically, using linear regression with least squares fitting – over the previous counts for that CS to predict the count in the subsequent version; and (2) using our diff algebra (Δ) averaged over the w previous diffs and added to the latest version to derive the prediction.

We then apply two experiments. The first experiment considers the counts of subjects with an exact characteristic set, evaluating the quality of prediction given an exact CS, for example, to predict how a particular subject might change. The results are shown in Table 1, where we see that our diff algebra (Δ) outperforms the baseline method (LM) in all cases, with smaller error by a considerable margin. We attribute this to the fact that Δ considers where entities come from, whereas LM does not: for example, if we consider two weeks of training data where a bulk edit is made between the two weeks adding a property p to each entity with CS $\{q, r\}$, LM will predict the same increase again in $\{p, q, r\}$ for the next week whereas Δ will recognise that there are no "source" entities left in $\{q, r\}$ and will not predict such an increase again. We also see that considering more weeks improves the quality of prediction for Δ: considering further training data allows to smooth out the effect of certain bursty (e.g., bot) edits between recent versions. In both cases, RMSE $\gg$ MAE, indicating that most predictions of CS cardinalities are accurate, but a few predictions have large errors.

The second experiment we run considers the counts of subjects with at least a given characteristic set (but that may have further properties); a concrete use-case would be to predict how the results for a query with those properties may change. First, we note that the overall error rises considerably, which is to be expected as the absolute (transitive) counts likewise increase considerably. As before, we see that the Δ-based predictions considerably outperform the LM baseline, and that the errors decrease for Δ as further weeks of training data are considered for the prediction.

Table 1: Quality of predicted #-lattice for exact intent

w	LM (RMSE)	LM (MAE)	Δ (RMSE)	Δ (MAE)
2	167.2	0.5697	25.26	0.1286
3	173.9	0.5595	19.15	0.1134
4	186.6	0.6051	17.71	0.1078
5	196.0	0.6624	17.31	0.1020
6	202.9	0.6842	15.62	0.0941

Table 2: Quality of predicted #-lattice for transitive intent

w	LM (RMSE)	LM (MAE)	Δ (RMSE)	Δ (MAE)
2	1477.8	177.0	264.2	6.19
3	1458.9	162.4	209.1	5.09
4	1535.6	178.8	185.7	4.50
5	1398.8	123.4	176.7	4.15
6	1357.8	59.6	145.8	3.67

Evaluation material: Source code and other evaluation materials are available at: https://github.com/larryjgonzalez/rdf_dynamics.

8 CONCLUSION

In this paper, we have presented a framework for computing a data-driven schema from large-scale knowledge graphs based on Formal Concept Analysis. Given that FCA is challenging to apply at scale, we proposed more lightweight structures that similarly provide a concept hierarchy based on a lattice of characteristic sets. We then discussed algorithms for extracting these characteristic sets and building the resulting lattices in a scalable and efficient manner. As a concrete use-case, we presented an algebraic method by which these lattices can be used to predict high-level changes in the dataset. Our evaluation over 11 weeks of Wikidata versions – each with more than 1 billion triples, 50 million subjects and 3 thousand properties – demonstrates the feasibility of our approach. Furthermore, we validated the quality of predictions made by our algebraic approach against a linear-model baseline.

There are a number of future directions for follow-up work. Aside from Wikidata, it would be interesting to conduct further experiments on other knowledge graphs with different scales and different types of dynamic behaviour. We also wish to investigate other applications for our proposed schema, including query processing, user interfaces, etc. Other variations of schema could also be explored, including, for example, concepts that encode type values or multiplicity, or quotient graphs based on characteristic sets. In general, we foresee much potential in the area of deriving data-driven schema from emergent knowledge graphs.

Acknowledgements: We thank Pablo Barceló and Pablo Muñoz for discussions that inspired this topic. We also thank Romana Pernischova and the anonymous reviewers for their comments. This work was supported by the Millennium Nucleus Center for Semantic Web Research, Grant No. NC120004; by Fondecyt Grant No. 1181896; by the German Research Foundation (DFG) within the Collaborative Research Center SFB 912 (HAEC); and by Emmy Noether grant KR 4381/1-1 (DIAMOND).

REFERENCES

[1] Ziawasch Abedjan, Toni Grütze, Anja Jentzsch, and Felix Naumann. 2014. Profiling and mining RDF data with ProLOD++. In *International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 1198–1201.

[2] Ziawasch Abedjan and Felix Naumann. 2013. Improving RDF Data Through Association Rule Mining. *Datenbank-Spektrum* 13, 2 (2013), 111–120.

[3] Mehwish Alam, Aleksey Buzmakov, Víctor Codocedo, and Amedeo Napoli. 2015. Mining Definitions from RDF Annotations Using Formal Concept Analysis. In *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 823–829.

[4] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. 2009. Describing Linked Datasets. In *Workshop on Linked Data on the Web (LDOW)*. CEUR-WS.org.

[5] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoc Vrgoč. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Computing Surveys* 50, 5 (2017). https://doi.org/10.1145/3104031

[6] Franz Baader, Bernhard Ganter, Baris Sertkaya, and Ulrike Sattler. 2007. Completing Description Logic Knowledge Bases Using Formal Concept Analysis. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 230–235.

[7] Christoph Böhm, Gjergji Kasneci, and Felix Naumann. 2012. Latent topics in graph-structured data. In *ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, 2663–2666.

[8] Christoph Böhm, Johannes Lorey, and Felix Naumann. 2011. Creating voiD descriptions for Web-scale data. *J. Web Sem.* 9, 3 (2011), 339–345.

[9] Dan Brickley, R.V. Guha, and Brian McBride. 2014. RDF Schema 1.1. W3C Recommendation. (25 Feb. 2014). http://www.w3.org/TR/rdf-schema/.

[10] Peter Buneman and Slawek Staworko. 2016. RDF Graph Alignment with Bisimulation. *PVLDB* 9, 12 (2016), 1149–1160.

[11] Stéphane Campinas, Thomas Perry, Diego Ceccarelli, Renaud Delbru, and Giovanni Tummarello. 2012. Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation. In *Database and Expert Systems Applications Workshop (DEXA)*. IEEE Computer Society, 261–266.

[12] Šejla Čebirić, François Goasdoué, and Ioana Manolescu. 2015. Query-Oriented Summarization of RDF Graphs. *PVLDB* 8, 12 (2015), 2012–2015. http://www.vldb.org/pvldb/vol8/p2012-cebiric.pdf

[13] Michael Cochez, Stefan Decker, and Eric Prud'hommeaux. 2016. Knowledge Representation on the Web Revisited: The Case for Prototypes. In *International Semantic Web Conference (ISWC) (Lecture Notes in Computer Science)*. Springer, 151–166.

[14] Mariano P. Consens, Valeria Fionda, Shahan Khatchadourian, and Giuseppe Pirrò. 2015. S+EPPs: Construct and Explore Bisimulation Summaries, plus Optimize Navigational Queries; all on Existing SPARQL Systems. *PVLDB* 8, 12 (2015), 2028–2031. http://www.vldb.org/pvldb/vol8/p2028-consens.pdf

[15] Mathieu d'Aquin and Enrico Motta. 2011. Extracting relevant questions to an RDF dataset using formal concept analysis. In *International Conference on Knowledge Capture (K-CAP)*. ACM, 121–128.

[16] Frithjof Dau and Baris Sertkaya. 2011. Formal Concept Analysis for Qualitative Data Analysis over Triple Stores. In *Advances in Conceptual Modeling (ER)*. Springer, 45–54.

[17] Jeffrey Dean and Sanjay Ghemawat. 2010. MapReduce: a flexible data processing tool. *Commun. ACM* 53, 1 (2010), 72–77.

[18] Renata Queiroz Dividino, Thomas Gottron, and Ansgar Scherp. 2015. Strategies for Efficiently Keeping Local Linked Open Data Caches Up-To-Date. In *International Semantic Web Conference (ISWC)*. Springer, 356–373.

[19] Marek Dudás, Vojtech Svátek, and Jindrich Mynarz. 2015. Dataset Summary Visualization with LODSight. In *Extended Semantic Web Conference (ESWC) – Demo*. Springer, 36–40.

[20] Fernando Florenzano, Denis Parra, Juan L. Reutter, and Freddie Venegas. 2016. A Visual Aide for Understanding Endpoint Data. In *International Workshop on Visualization and Interaction for Ontologies and Linked Data (VOILA@ISWC)*. CEUR-WS.org, 102–113.

[21] Anna Formica. 2012. Semantic Web search based on rough sets and Fuzzy Formal Concept Analysis. *Knowl.-Based Syst.* 26 (2012), 40–47. https://doi.org/10.1016/j.knosys.2011.06.018

[22] Mohamed Rouane Hacene, Marianne Huchard, Amedeo Napoli, and Petko Valtchev. 2007. A Proposal for Combining Formal Concept Analysis and Description Logics for Mining Relational Data. In *International Conference on Formal Concept Analysis (ICFCA)*. Springer, 51–65.

[23] Ali Hasnain, Qaiser Mehmood, Syeda Sana e Zainab, and Aidan Hogan. 2016. SPORTAL: Profiling the Content of Public SPARQL Endpoints. *Int. J. Semantic Web Inf. Syst.* 12, 3 (2016), 134–163.

[24] Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne, and Aidan Hogan. 2013. Observing Linked Data Dynamics. In *Extended Semantic Web Conference (ESWC)*. Springer, 213–227.

[25] Sheila Kinsella, Uldis Bojars, Andreas Harth, John G. Breslin, and Stefan Decker. 2008. An Interactive Map of Semantic Web Ontology Usage. In *International Conference on Information Visualisation*. 179–184.

[26] Markus Kirchberg, Erwin Leonardi, Yu Shyang Tan, Sebastian Link, Ryan K. L. Ko, and Bu-Sung Lee. 2012. Formal Concept Discovery in Semantic Web Data. In *International Conference on Formal Concept Analysis (ICFCA)*. Springer, 164–179.

[27] Holger Knublauch and Dimitris Kontokostas. 2014. Shapes Constraint Language (SHACL). W3C Working Group Note. (24 June 2014). http://www.w3.org/TR/rdf11-primer/.

[28] Petr Krajca and Vilém Vychodil. 2009. Distributed Algorithm for Computing Formal Concepts Using Map-Reduce Framework. In *International Symposium on Intelligent Data Analysis (IDA)*. Springer, 333–344.

[29] Nandana Mihindukulasooriya, María Poveda-Villalón, Raúl García-Castro, and Asunción Gómez-Pérez. 2015. Loupe – An Online Tool for Inspecting Datasets in the Linked Data Cloud. In *International Semantic Web Conference (ISWC) Posters & Demos*. CEUR-WS.org.

[30] Thomas Neumann and Guido Moerkotte. 2011. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 984–994.

[31] Minh-Duc Pham and Peter A. Boncz. 2016. Exploiting Emergent Schemas to Make RDF Systems More Efficient. In *International Semantic Web Conference (ISWC) (Lecture Notes in Computer Science)*. Springer, 463–479.

[32] François Picalausa, George H. L. Fletcher, Jan Hidders, and Stijn Vansummeren. 2014. Principles of Guarded Structural Indexing. In *International Conference on Database Theory (ICDT)*. OpenProceedings.org, 245–256.

[33] Jonas Poelmans, Sergei O. Kuznetsov, Dmitry I. Ignatov, and Guido Dedene. 2013. Formal Concept Analysis in knowledge processing: A survey on models and techniques. *Expert Syst. Appl.* 40, 16 (2013), 6601–6623.

[34] Laurens Rietveld, Wouter Beek, Rinke Hoekstra, and Stefan Schlobach. 2017. Meta-data for a lot of LOD. *Semantic Web* 8, 6 (2017), 1067–1080.

[35] Mohamed Rouane-Hacene, Marianne Huchard, Amedeo Napoli, and Petko Valtchev. 2013. Relational Concept Analysis: mining concept lattices from multi-relational data. *Ann. Math. Artif. Intell.* 67, 1 (2013), 81–108. https://doi.org/10.1007/s10472-012-9329-3

[36] Alexander Schätzle, Antony Neu, Georg Lausen, and Martin Przyjaciel-Zablocki. 2013. Large-scale bisimulation of RDF graphs. In *Semantic Web Information Management (SWIM) Workshop*.

[37] Guus Schreiber and Yves Raimond. 2014. RDF 1.1 Primer. W3C Working Group Note. (24 June 2014). http://www.w3.org/TR/rdf11-primer/.

[38] Jürgen Umbrich, Michael Hausenblas, Aidan Hogan, Axel Polleres, and Stefan Decker. 2010. Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources. In *Workshop on Linked Data on the Web (LDOW)*. CEUR-WS.org.

[39] Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85. https://doi.org/10.1145/2629489

[40] Rudolf Wille. 2009. Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts. In *International Conference on Formal Concept Analysis (ICFCA) (Lecture Notes in Computer Science)*. Springer, 314–339. (Reprint).

[41] Biao Xu, Ruairí de Fréin, Eric Robson, and Mícheál Ó Foghlú. 2012. Distributed Formal Concept Analysis Algorithms Based on an Iterative MapReduce Framework. In *International Conference on Formal Concept Analysis (ICFCA) (Lecture Notes in Computer Science)*. Springer, 292–308.