Towards Logical Linked Data Compression

Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong

Kno.e.sis Center, Wright State University, Dayton, OH, U.S.A.

Abstract. Linked data has experienced accelerated growth in recent years. With the continuing proliferation of structured data, demand for RDF compression is becoming increasingly important. In this study, we introduce a novel lossless compression technique for RDF datasets, called Rule Based compression (RB compression) that compresses datasets by generating a set of new logical rules from the dataset and removing triples that can be inferred from these rules. We employ existing frequent pattern mining algorithms for generating new logical rules. Unlike other compression techniques, our approach not only takes advantage of syntactic verbosity and data redundancy but also utilizes intra- and inter-property associations in the RDF graph. Depending on the nature of the dataset, our system is able to prune more than 50% of the original triples without affecting data integrity.

1 Introduction

Linked Data has received much attention in recent years due to its interlinking ability across disparate sources, made possible via machine processable non-proprietary RDF data [14]. Today, large numbers of organizations, including governments, share data in RDF format for easy re-use and integration of data by multiple applications. This has led to accelerated growth in the amount of RDF data being published on the web. Although the growth of RDF data can be viewed as a positive sign for semantic web initiatives, it also causes performance bottlenecks for RDF data management systems that store and provide access to data [11]. As such, the need for compressing structured data is becoming increasingly important.

Earlier RDF compression studies [3, 5] have focused on generating a compact representation of RDF. [5] introduced a new compact format called *HDT* which takes advantage of the powerlaw distribution in term-frequencies, schema and resources in RDF datasets. The compression is achieved due to the compact form rather than a reduction in the number of triples. [12] introduced the notion of a lean graph which is obtained by eliminating triples which contain blank nodes that specify redundant information. [15] proposed a user-specific redundancy elimination technique based on rules. Similarly, [17] studied RDF graph minimization based on rules, constraints and queries provided by users. The latter two approaches are application dependant and require human input, which makes them unsuitable for compressing the ever growing set of linked datasets.

In this paper, we introduce scalable lossless compression of RDF datasets using automatic generation of *decompression rules*. We have devised an algorithm to automatically generate a set of rules and split the database into two smaller disjoint datasets, viz., an *Active* dataset and a *Dormant* dataset based on those rules. The dormant dataset contains list of triples which remain uncompressed and to which no rule can be applied during decompression. On the other hand, the active dataset contains list of compressed triples, to which rules are applied for inferring new triples during decompression.

In order to automatically generate a set of rules for compression, we employ frequent pattern mining techniques [8, 13]. We examine two possibilities for frequent mining - a) within each property (hence, intra-property) and b) among multiple properties(inter-property). Experiments reveal that compression based on inter-property frequent patterns are better than those done based on intra-property frequent patterns.

Specifically, the contribution of this work is a rule based compression technique with the following properties:

- The compression reduces the number of triples, without introducing any new subjects, properties and objects.
- The set of decompression rules, R, can be automatically generated using various algorithms.
- The compression can potentially aid in discovery of new interesting rules.
- It is highly scalable with the ability to perform incremental compression on the fly.

2 Preliminaries

2.1 Frequent Itemset Mining

The concept of frequent itemset mining [1] (FIM) was first introduced for mining transaction databases. Over the years, frequent itemset mining has played an important role in many data mining tasks that aim to find interesting patterns from databases, including association rules and correlations, or aim to use frequent itemset to construct classifiers and clusters [6]. In this study, we exploit frequent itemset mining techniques on RDF datasets for generating logical rules and subsequent compressing of RDF datasets.

Transaction Database Let $I = \{i_1, i_2, ..., i_n\}$ be a set of distinct items. A set $X = \{i_1, i_2 ..., i_k\} \subseteq I$ is called an itemset, or a k-itemset if it contains k items. Let D be a set of transactions where each transaction, $T = (tid, X)$, contains a unique transaction identifier, tid, and an itemset X. Figure 1 shows a list of transactions corresponding to a list of triples. Here, subjects represent identifiers and the set of corresponding objects represent transactions. In this study, we use the following definitions for intra- and inter-property transactions.

Intra-property transactions: For a graph G containing a set of triples, an intra-property transaction corresponding to a property p is a set $T = (s, X)$ such that s is a subject and X is a set of objects, i.e. (s, p, o_x) is a triple in graph G. o_x is an element of x.

Inter-property transactions: For a graph G containing a set of triples, an inter-property transaction is a set $T = (s, Z)$ such that s is a subject and each Z is a pair (p, o) of property and object, i.e. (s, p_z, o_z) is a triple in graph G.

Support and Frequent Itemset The support of an itemset X, denoted by $\sigma(X)$, is the number of transactions in D containing X. Itemset X is said to be frequent if $\sigma(X) \geq minSup$ (minSup is a minimum support threshold).

Itemset Mining

Definition 1. *Let D be a transaction database over a set of items I, and σ_{min} a minimum support threshold. The set of frequent itemsets in D with respect to σ_{min} is denoted by $F(D, \sigma_{min})\colon = X \subseteq I | \sigma(X) \geq \sigma_{min}$,*

A frequent itemset is often referred to as a *frequent pattern.* Numerous studies have been done and various algorithms [1, 2, 8, 18, 19] have been proposed to mine frequent itemsets. Among these algorithms, *Apriori* [1] was the first algorithm to generate frequent patterns. It consists of multiple mining iterations, each mining frequent patterns of a given length. However, it requires a large number of database scans to generate frequent patterns. An alternative *FP-Growth* [8] algorithm was introduced about seven years after *Apriori*; it is much faster than the Apriori algorithm. Parallelized versions of FP-Growth [13, 20, 16] have also been explored. In this study, we use FP-Growth for generating frequent patterns.

s1 rdf:type 125
s1 rdf:type 22.
s1 rdf:type 225.
s1 rdf:type 60.
s6 rdf:type 90.
s5 rdf:type 125.
s2 rdf:type 225.
s2 rdf:type 22.
s3 rdf:type 22.
s4 rdf:type 125.
s4 rdf:type 22.
s4 rdf:type 225.
s4 rdf:type 60.
s4 rdf:type 22.
s5 rdf:type 22.
s2 rdf:type 125.
s3 rdf:type 81.

(a) Triples for rdf:type property

TID	rdf:type
S1	125,22,225,60
S2	125,22,225
S3	81,22
S4	125,22,225,60
S5	125,22
S6	90,22

(b) Transactions

Fig. 1. RDF Triples and Corresponding Transactions.

FP-Growth FP-Growth uses a recursive divide-and-conquer approach to decompose both the mining tasks and the database [7]. It requires only two scans on the database. For a given input dataset, it scans the data set to compute a list of frequent items sorted in frequency descending order. Then, the database is compressed into a frequent pattern tree (FP-tree). Then it starts to mine the FP-tree for each item whose support is larger than the support(σ_{min}) by recursively constructing conditional FP-tree for the item [7]. It transforms the problem of finding frequent patterns to identifying frequent items and constructing trees recursively [13].

Key	Value
225	([22, 225],525786)
60	([22, 225,60],525786)
189	([22, 227, 83, 189],60194)
213	([22, 227, 83, 189, 213],60194)
173	([22, 103, 26, 304, 173],57772)
70	([22, 70],56372), ([22, 103, 26,304,173,70],31084), ([22,202,42,70],25288)
13	([22, 225, 60, 174, 13],53120)
235	([22, 225, 60, 174, 235],52305), ([22, 225, 60, 202, 42, 174, 235],480)
126	([22, 191, 97, 222, 126],49252)

(a) Frequent patterns with support

ID	Object
22	owl:Thing
227	dbp:Work
83	schema:CreativeWork
189	dbp:Film
213	schema:Movie
103	dbp:Person
26	schema:Person
304	foaf:Person
173	dbp:Artist
225	dbp:Place
60	schema:Place

(b) object mappings

Fig. 2. Frequent patterns generated for 'DBpedia Ontology Types' dataset and several object mappings

Figure 2 shows several frequent patterns for one of the core DBpedia datasets containing only rdf:type property[1]. To generate such frequent patterns, we first create a transaction database as shown in Figure 1 and then use FP-Growth. Please refer to [8, 13] for the details about the FP-Growth algorithm and implementation. In this paper, we represent the output of FP-Growth as a set of pairs $\langle k, F_k \rangle$, where k is an item, and F_k, a frequent pattern corresponding to k, is in turn a set of pairs of the form $\langle v, \sigma_v \rangle$. v is an itemset of a frequent pattern and σ_v is a support of this frequent pattern.

2.2 Association Rule Mining

Frequent itemset mining is often associated with *association rule mining*, which involves generating association rules from the frequent itemset with constraints of minimal confidence (to determine if a rule is interesting or not). However, in this study, we do not require mining association rules using confidence values. Instead, we split the given database into two disjoint databases, say A and B, based on the most frequent patterns. Those transactions which contain one or more of the top k frequent patterns are inserted into dataset A while the other transactions are inserted into database B. Compression can be performed by creating a set of rules using top k frequent patterns and removing those triples from the dataset which can be inferred by applying rules to some other triples in the same dataset. Algorithmic details are provided in Section 4.

[1] `http://downloads.dbpedia.org/preview.php?file=3.7_sl_en_sl_instance_types_en.nt.bz2`

3 Rule based Compression (RB Compression)

We consider an RDF Graph G containing $|G|$ non-duplicate triples. Lossless compression on graph G can be obtained by splitting the given graph G into *Active Graph, G_A,* and a *Dormant Graph, G_D,* such that: $G \equiv R(G_A) \cup G_D$ where R represents the set of *decompression rules* to be applied to an active graph (G_A) during decompression.

G_A together with G_D represents the compressed graph and the ratio of compressed triple size to uncompressed triple size is the compression ratio, denoted by r. In equation,
$r = \frac{|G_A|+|G_D|}{|G|}$

Since the compression is lossless, we have $|G| = |R(G_A)| + |G_D|$.

Definition 2. *Let G be an RDF graph containing a set T of triples. A dormant graph, $G_D \subset G$ is a graph containing some $T_D \subset T$ triples. An active graph, denoted by G_A, is a graph containing $T_A \subset \{T - T_D\}$ triples such that when a set of R decompression rules is applied to G_A (denoted by $R(G_A)$), it produces a graph containing exactly the $\{T - T_D\}$ set of triples.*

G_D is referred to as dormant since it remains unchanged during decompression (no rule can be applied to it during decompression).

In the next section, we provide an algorithm to automatically generate the compressed version of an RDF graph G. Specifically, we investigate how to

- generate a set of decompression rules, R.
- decompose the graph G to G_A and G_D, such that the definition of RB compression holds true.
- maximize the reduction in number of triples.

4 Algorithms

In this section, we introduce two rule based compression algorithms using intra- and inter-property frequent patterns respectively. In addition, we provide an algorithm for *delta compression* to deal with incremental compression when a set of triples needs to be added to existing compressed graphs.

For both algorithms, the following holds true. Given an RDF graph G, RB compression outputs two graphs: G_A (Active) and G_D (Dormant), and a set of R decompression rules. We represent the output of FP-growth as a set of pairs $\langle k, F_k \rangle$, where k is an item, and F_k, a frequent pattern corresponding to k, is in turn a set of pairs of the form $\langle v, \sigma(v) \rangle$. v is an itemset of frequent pattern and $\sigma(v)$ is a support of this frequent pattern. For the sake of simplicity, we choose only one frequent pattern such that for a given k, v_k has the maximum support and a length of greater than one. In Algorithms, a rule resulting from this frequent pattern is written as $k \rightarrow v_k$.

A decompression can be performed either sequentially or in parallel. Sequential decompression is trivial and requires merging of inferred triples into the resulting uncompressed graph produced by decompression using previous rules. For parallel decompression, an active graph can be scanned in parallel for each rule. This allows generation of inferred triples in parallel. Since triples are not ordered, inferred triples can be added to an uncompressed graph whenever they are generated. Finally, all triples of the dormant graph are merged into this uncompressed graph. Storage needed for the rules is negligible in comparison with the storage of the active and dormant graphs.

4.1 RB compression using intra-property associations

Algorithm 1 RB compression using intra-property association

```
Require: $G$
$R \leftarrow \phi$, $G_D \leftarrow \phi$ , $G_A \leftarrow \phi$
for each property, $p$ that occurs in $G$ do
    create a transaction database $D$ from a set of intra-property transactions. Each transaction $(s, t)$ contains a subject $s$ as identifier and $t$ a list of corresponding objects.
    generate a set of frequent patterns $\langle k, F_k \rangle$ using FP-Growth
    for all $k$ that occurs in $\langle k, F_k \rangle$ do
        select $v_k$ such that $\sigma(v_k) = argmax_v \sigma(v) | v$ occurs in $F_k, |v| > 1$
        $R \leftarrow R \cup (k \rightarrow v_k)$        ▷ add a new rule
    end for
    for each $(s, t) \in D$ do
        for each $k \in R$ do
            if $t \cap v_k = v_k$ then
                $G_A \leftarrow G_A \cup (s, p, k)$        ▷ add this triple to active graph
                $t \leftarrow t - v_k$
            end if
        end for
        for each $o \in t$ do
            $G_D \leftarrow G_D \cup (s, p, o)$        ▷ add to dormant graph.
        end for
    end for
end for
```

Algorithm 1 follows a divide and conquer approach. For each property in a graph G, we create a new dataset and mine frequent patterns on this dataset. Transactions are created per subject within this dataset. Each transaction is a list of objects corresponding to a subject as shown in Figure 1. Using frequent patterns, a set of rules is generated for each property and later aggregated. Each rule contains a property p, an object item k, and a frequent pattern itemset v corresponding to k. A frequent pattern itemset, v, is a set of items including k.

The outcome of Algorithm 1 is the following logical rule that can be attached to an active graph G_A:

$$\forall x.triple(x,p,k) \rightarrow \bigwedge_{i=1}^{n} triple(x,p,v_i) \qquad \text{where, } v = v_1, v_2, ..., v_n$$

For illustration, here's one such decompression rule we obtained during an experiment on one core DBpedia dataset:

$\forall x$.triple(x, rdf:type, foaf:Person) → triple(x, rdf:type, schema:Person)
∧ triple(x, rdf:type, dbp:Person)
∧ triple(x, rdf:type, owl:Thing)

This triple is added to the active graph GA while all triples that can be inferred from it are removed. Other triples which cannot be inferred, are placed in dormant graph G_D. The process is repeated for all properties, appending results to already existing rules R, active graph G_A and dormant graph G_D.

4.2 RB compression using inter-property associations

Algorithm 2 RB compression using inter-property associations

```
Require: $G$
$R \leftarrow \phi$, $G_D \leftarrow \phi$ , $G_A \leftarrow \phi$
create a transaction database $D$ from a set of inter-property transactions. Each
  transaction, $(s,t)$ contains a subject $s$ as identifier and $t$ a set of $(p,o)$ items.
generate a set of frequent patterns $\langle k, F_k \rangle$ using FP-Growth
for all $k$ that occurs in $\langle k, F_k \rangle$ do
    select $v_k$ such that $\sigma(v_k) = argmax_v \sigma(v) | v$ occurs in $F_k, |v| > 1$
    $R \leftarrow R \cup (k \rightarrow v_k)$                ▷ add a new rule
end for
for each $(s,t) \in D$ do
    for each $k \in R$ do
        if $t \cap v_k = v_k$ then        ▷ both $t$ and $v$ contain a set $(p,o)$ of items
            $G_A \leftarrow G_A \cup (s, p_k, o_k)$        ▷ add single triple to active graph
            $t \leftarrow t - v_k$
        end if
    end for
    for each $(p,o) \in t$ do
        $G_D \leftarrow G_D \cup (s,p,o))$        ▷ add triple to dormant graph.
    end for
end for
```

In Algorithm 2, we try to mine frequent patterns across different properties. Transactions used in this algorithm are created by generating a list of all possible pairs of property and objects for each subject. Thus, each item of a transaction is a pair $(p : o)$. We follow similar approach as before for generating frequent patterns and rules. Each rule contains a key pair (p_k, o_k) and a corresponding frequent pattern v as a list of items $(p : o)$. The procedure is similar to 4.2 once frequent patterns and rules are generated.

$$\forall x.triple(x, p_k, o_k) \rightarrow \bigwedge_{i=1}^{n} triple(x, p_i, o_i) \quad [\ v_i = (p_i, o_i)\ ]$$

4.3 RB-Delta Compression

One of the important properties of RB compression is that incremental compression can be achieved on the fly without much computation. Let's say, we consider an RDF graph G, which has undergone RB-Compression resulting in active graph G_A, dormant graph G_D and set R of decompression rules. If a new set of triples corresponding to a subject s, denoted by ΔT_s, needs to be added to Graph G, delta compression can be achieved by using the results from the last compression. Each delta compression updates the existing active and dormant graphs. Hence, there is no need for full RB-Compression every time a set of triples is added. Algorithm 3 provides a delta compression algorithm when ΔT_s needs to be added. The algorithm can be extended to include a set of subjects, S. If a triple needs to be removed, an extra check needs to be performed to see if the removal violates any existing rules. Such removal might require moving some of the inferred triples from the active graph to the dormant graph.

Algorithm 3 Delta Compression

```
Require: $G_A$, $G_D$, $R$ , $\Delta T_s$
$S \leftarrow \phi$
for all $t \in \Delta T_s$ do
    if $R(t) \subseteq \Delta T_s$ then
        $G_A \leftarrow G_A \cup t$                ▷ insert into active graph
    else
        $G_D \leftarrow G_D \cup t$                ▷ insert into dormant graph
    end if
end for
```

5 Evaluation

This section shows experimental results of the compression performed by our system. Our experiment is conducted on several linked open datasets, of varying sizes. The smallest dataset consists of 130K triples while the largest dataset consists of 119 million triples. Readers can download the original and compressed datasets with additional experimental details from the website[2]. The main purpose of this test is to validate the working of Rule Based compression techniques and test algorithm performance. We study both of these in detail.

[2] http://dl.dropbox.com/u/65933145/rbc_download

5.1 On intra- and inter-property association and compression

Compression ratio, r is defined as the ratio of the compressed size to the uncompressed size. Table 5.1 shows a comparison between the outputs of the two algorithms we discussed in Section 4 for nine different linked datasets. It is evident from the results that compression based on inter-property frequent patterns is far better than compression using intra-property frequent patterns. Details including the number of predicates and transactions derived during experiments are also included in the table. It can be seen that the best RB compression (inter-property) can remove around 50% of triples for the Geonames, DBpedia rdftypes and CN datasets.

Data Set	**#triples**	**#predicates**	**#transactions**	**compression ratio**	
				intra-property	inter-property
Dog Food	130,178	132	12,695	0.99	0.93
CN 2012	137,484	26	14,553	0.82	0.51
ArchiveHub	431,088	141	51,411	0.92	0.77
Jamendo	1,047,950	25	335,925	0.99	0.83
LinkedMdb	6,147,996	222	694,400	0.97	0.77
DBpedia rdftypes	9,237,320	1	9,237,320	0.49	0.49
RDF About	17,188,323	108	3,132,667	0.97	0.86
DBLP	46,597,620	27	2,840,639	0.96	0.88
Geonames	119,416,854	26	7,711,126	0.97	0.52

Table 1. Compression ratio (based on triple counts) for various linked open datasets

Data Set	**#predicates**		**frequent predicates**
	Total	Frequent	
Dog Food	132	16	rdf:type, dc:creator
CN 2012	26	2	skos:closeMatch, pscs:relatedMatch
ArchiveHub	141	7	rdf:type, geo:locatedIn
Jamendo	25	3	rdf:type, foaf:made
LinkedMdb	222	21	rdf:type, lmdb:genre
DBpedia rdftypes	1	1	rdf:type
RDF About	108	9	rdf:type, dc:subject
DBLP	27	4	rdf:type, dc:identifier
Geonames	26	3	foaf:page, geo:alternateName

Table 2. Properties with frequent patterns

During experiments on intra-property transactions, only few predicates exhibited frequent patterns. For most properties, a set of transactions don't result in any frequent pattern even for a low support of 3. Table 5.1 shows the number of predicates in the original dataset and the total number of predicates that resulted in frequent patterns for experiments involving intra-property transactions. For Geonames dataset, only 3 (foaf:page, geo:alternateName, geo:officialName) out of 26 properties exhibited frequent patterns. In most cases, *rdf:type* is shown to contain frequent patterns, which is expected since it supports a hierarchical structure (and hence associations). It is apparent that the DBpedia rdftype dataset, which contains only the *rdf:type* property, has the best compression ratios. The experimental results indicate that RDF datasets exhibit strong associations among different properties resulting in a greater reduction of triples.

5.2 Comparison using compressed dataset size

In addition to evaluating our system based on triple count, we examine the compression based on the storage size of the compressed datasets and compare it against other compression systems. This is important since none of the existing compression systems has the ability to compress RDF datasets by removing triples. [4] compared different universal compressors and found that bzip2 is one of the best universal compressors. For this study, we compress the input dataset (in N-Triples format) and the resulting dataset using bzip2 and provide a quantitative comparison (see Table 5.2). An advantage of semantic compression such as RB Compression is that one can still apply syntactic compression (e.g. HDT) to the results. HDT [5] achieves a greater compression for most of the datasets we experimented on. Such high performance can be attributed to its ability to take advantage of the highly skewed RDF data. Since any generic RDF dataset can be converted to HDT compact form, we did a test by converting the output of RB compression to HDT for the linkedMdb dataset. The resulting dataset size is only 9MB which is better than the individual compression. Performing the experiment with a few other datasets exhibited similar behavior and we observed that converting to HDT after RB compression results in the best compression.

Data Set	Size	compressed	**compressed size using bzip2**	
			HDT-Plain	inter-property
DogFood	23.4 MB	1.5MB	1.1 MB	904K
CN 2012	17.9 MB	488K	168K	532K
Archive Hub	71.8 MB	2.5MB	1.79 MB	1.73MB
Jamendo	143.9 MB	6MB	4.3M	5.6MB
LinkedMdb	850.3 MB	22MB	16 MB	27MB
DBpedia rdftypes	1.2 GB	45MB	44 MB	39 MB
RDFabout	4.3 GB	79MB	45 MB	75M
DBLP	10.9 GB	265 MB	201 MB	239 MB
Geonames	13G	410 MB	274 MB	380 MB

Table 3. Comparison of compression ratio based on dataset size

5.3 Soundness and Completeness of RB compression

Although it should already be rather clear from our definitions and algorithms that our compression is *lossless* in the sense that we can recover all erased triples by using the newly introduced rules—let us dwell on this point for a little while.

First of all, it is worth mentioning that we cannot only recreate all erased triples by exhaustive forward-application of the rules—a fact that we could reasonable refer to as *completeness* of our approach. Rather, our approach is also *sound* in the sense that *only* previously erased triples are created by application of the rules. I.e., our approach does *not* include an inductive component, but is rather restricted to *detecting patterns which are explicitly and exactly represented in the dataset.* Needless to say, the recreation of erased triples using a forward-chaining application of rules can be rephrased as using a deductive reasoning system as decompressor.

It is also worth noting that the rules which we introduce, which are essentially of the form $\text{triple}(x, p, k) \rightarrow \text{triple}(x, p, v)$, can also be expressed in the OWL [9] Web ontology Language. Indeed, a triple such as (x, p, k) can be expressed in OWL, e.g., in the form[3] $k(x)$ if p is `rdf:type`, or in the form $p(x, k)$ if p is a newly introduced property. The rule above then becomes $k \sqsubseteq v$ for p being `rdf:type`, and it becomes $\exists p.\{k\} \sqsubseteq \exists p.\{v\}$ in the case of the second example.

The observation just made that our compression rules are expressible in OWL. From this perspective, our approach to lossless compression amounts to the creation of schema knowledge which is completely faithful (in the sound and complete sense) to the underlying data. I.e., it amounts to the introduction of *uncontroversial* schema knowledge to Linked Data sets. It is rather clear that this line of thinking opens up a plethora of exciting follow-up work, which we intend to pursue.

6 Conclusion

In this paper, we have introduced a novel lossless compression technique called rule based compression (RB compression) that efficiently compresses RDF datasets using logical rules. The key idea is to split the original dataset into two disjoint datasets A and B, such that A adheres to certain logical rules while B does not. Dataset A can be compressed since we can prune those triples that can be inferred by applying rules on some other triples in the same dataset. We have provided two algorithms based on frequent pattern mining to demonstrate the compression capability of our rule based compression. Experimental results show that in some datasets, RB Compression can remove almost half the triples without losing data integrity. The approach is highly scalable due to the dynamic compression capability exhibited by RB-Delta compression. Rules generated by RB compression can be used to study instance alignment and automated schema generation. In future work, we will explore more efficient algorithms for better compression and will explore the effect of RB-Compression in the querying of linked open datasets.

[3] We use description logic notation for convenience, see [10].

Acknowledgements

This work was supported by the National Science Foundation under award 1143717 "III: EAGER – Expressive Scalable Querying over Linked Open Data" and 1017225 "III: Small: TROn – Tractable Reasoning with Ontologies."

References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: SIGMOD Conference. pp. 207–216 (1993)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB. pp. 487–499 (1994)
3. Álvarez-García, S., Brisaboa, N.R., Fernández, J.D., Martínez-Prieto, M.A.: Compressed k2-triples for full-in-memory rdf engines. In: AMCIS (2011)
4. Fernández, J.D., Gutierrez, C., Martínez-Prieto, M.A.: Rdf compression: basic approaches. In: WWW. pp. 1091–1092 (2010)
5. Fernández, J.D., Martínez-Prieto, M.A., Gutierrez, C.: Compact representation of large rdf data sets for publishing and exchange. In: ISWC. pp. 193–208 (2010)
6. Goethals, B.: Survey on frequent pattern mining pp. 1–43 (2003), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.2405
7. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov. 15(1), 55–86 (2007)
8. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Min. Knowl. Discov. 8(1), 53–87 (2004)
9. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S. (eds.): OWL 2 Web Ontology Language: Primer. W3C Recommendation 27 October 2009 (2009), available from http://www.w3.org/TR/owl2-primer/
10. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
11. Huang, J., Abadi, D.J., Ren, K.: Scalable sparql querying of large rdf graphs. PVLDB 4(11), 1123–1134 (2011)
12. Iannone, L., Palmisano, I., Redavid, D.: Optimizing rdf storage removing redundancies: An algorithm. In: IEA/AIE, pp. 732–742 (2005)
13. Li, H., Wang, Y., Zhang, D., Zhang, M., Chang, E.Y.: Pfp: parallel fp-growth for query recommendation. In: RecSys. pp. 107–114 (2008)
14. Manola, F., Miller, E., McBride, B.: Rdf primer (2004), http://www.w3.org/TR/rdf-primer/
15. Meier, M.: Towards rule-based minimization of rdf graphs under constraints. In: RR. pp. 89–103 (2008)
16. Özdogan, G.Ö., Abul, O.: Task-parallel fp-growth on cluster computers. In: ISCIS. pp. 383–388 (2010)
17. Pichler, R., Polleres, A., Skritek, S., Woltran, S.: Redundancy elimination on rdf graphs in the presence of rules, constraints, and queries. In: RR (2010)
18. Savasere, A., Omiecinski, E., Navathe, S.B.: An efficient algorithm for mining association rules in large databases. In: VLDB. pp. 432–444 (1995)
19. Srikant, R., Vu, Q., Agrawal, R.: Mining association rules with item constraints. In: KDD. pp. 67–73 (1997)
20. Zaïane, O.R., El-Hajj, M., Lu, P.: Fast parallel association rule mining without candidacy generation. In: ICDM. pp. 665–668 (2001)