1-1-2013

Logical Linked Data Compression

Amit Krishna Joshi

Pascal Hitzler
pascal.hitzler@wright.edu

Guozhu Dong
guozhu.dong@wright.edu

Logical Linked Data Compression

Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong

Kno.e.sis Center, Wright State University, Dayton, OH, U.S.A.
{joshi35, pascal.hitzler, guozhu.dong}@wright.edu

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. Unlike other compression techniques, our approach not only takes advantage of syntactic verbosity and data redundancy but also utilizes semantic associations present 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 it's interlinking ability across disparate sources, made possible via machine processable non-proprietary RDF data [18]. Today, large number 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 [12]. As such, the need for compressing structured data is becoming increasingly important.

Earlier RDF compression studies [3, 6] have focused on generating a compact representation of RDF. [6] 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 a compact form representation rather than a reduction in the number of triples. [13] introduced the notion of a lean graph which is obtained by eliminating triples which contain blank nodes that specify redundant information. [19] proposed a user-specific redundancy elimination technique based on rules. Similarly, [21] studied RDF graph minimization based on rules, constraints and queries provided by users. The latter two approaches are application dependent and require human input, which makes them unsuitable for compressing the ever growing set of linked datasets.

In this paper, we introduce a 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 [9, 15]. We examine two possibilities for frequent mining - a) within each property (hence, intra-property) and b) among multiple properties(inter-property). Experiments reveal that RB compression performs better when inter-property transactions are used instead of intra-property transactions.

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 or 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.

A very preliminary and limited version of this paper appeared in [14].

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.”

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 itemsets to construct classifiers and clusters [7]. 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 $I = \{i_1, i_2, \ldots, i_n\}$ be a set of distinct items. A set $X = \{i_1, i_2, \ldots, i_k\} \subseteq I$ 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 containing the rdf:type[1]

[1] *rdf:type* is represented by *a*

property. 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 a member 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 member of Z is a pair (p_z, o_z) of property and object, i.e. (s, p_z, o_z) is a triple in graph G.

s1 a 125 s4 a 125.
s1 a 22. s4 a 22.
s1 a 225. s4 a 225.
s1 a 60. s4 a 60.
s6 a 90. s6 a 22.
s5 a 125. s5 a 22.
s2 a 225. s2 a 125.
s2 a 22. s3 a 81.
s3 a 22.

(a) Triples

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. Triples and corresponding transactions.

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 $\sigma(X) \geq \sigma_{min}$ (σ_{min} is a minimum support threshold).

Itemset Mining

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

A frequent itemset is often referred to as a *frequent pattern*. Numerous studies have been done and various algorithms [1, 2, 9, 22, 23] have been proposed to mine frequent itemsets. In this study, we use the *FP-Growth* [9] algorithm for generating frequent itemsets. We represent the output of FP-Growth as a set of pairs (k, F_k), where k is an item, and F_k, a set of frequent patterns corresponding to k. Each frequent pattern is a pair of the form (v, σ_v). v is an itemset of a frequent pattern and σ_v is a support of this frequent pattern.

Figure 2(a) shows several frequent patterns for *DBpedia Ontology Types* dataset containing only the rdf:type property.[2] To generate such frequent patterns, we first create a transaction database as shown in Figure 1 and then use

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

Item (k)	Frequent Patterns (F_k)
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

Item	Object
22	owl:Thing
227	dbp:Work
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. Sample frequent patterns generated for *DBpedia Ontology Types* dataset. Each item represents a numerically encoded object. An item can be associated with multiple frequent patterns as seen for item 70.

parallel FP-Growth to compute frequent patterns. Please refer to [9, 15] for details about the FP-Growth algorithm and its implementation. Figure 3 shows the list of inter-property frequent patterns for one of the linked open datasets.

Item	Frequent Patterns
6:114	{([1:101, 5:113, 6:114],748384), ([1:101, 11:8912626, 5:113, 6:114],230746)}
5:102	{([1:101, 5:102],1042692), ([1:101, 11:8912626, 5:102],225428)}
5:176	{([1:101, 5:176],1695814), ([1:101, 11:8912626, 5:176],1044079)}
6:109	{([1:101, 5:108, 6:109],2792865), ([1:101, 5:108, 6:109, 11:8912626],166815)}

Fig. 3. Frequent patterns generated for the Geonames dataset. Each item is a pair of property and object ($p : o$)

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 frequent patterns. Those transactions which contain one or more of the top N frequent patterns are inserted into dataset A while the other transactions are inserted into dataset B. Compression can be performed by creating a set of rules using top N frequent patterns and removing those triples from the dataset which can be inferred by applying rules to some other triples in the same dataset.

Multi-Dimensional Association Rules Although association mining was originally studied for mining transactions for only one attribute (ex:Product), much research has been performed to extend it across multiple attributes [16, 17, 28, 29]. In this study, RDF datasets are viewed as multi-dimensional transaction databases by treating each property as an attribute and a subject as an identifier. Similar to intra-transaction and inter-transaction associations [17], we define intra-property and inter-property associations for RDF datasets. Intra-property association refers to an association among different object values for a given property while inter-property association refers to association between multiple properties.

3 Rule Based Compression

In this section, we introduce two RB compression algorithms - one using intra-property transactions and the other using inter-property transactions. 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. 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 requirements of RB compression holds true.
- maximize the reduction in number of triples

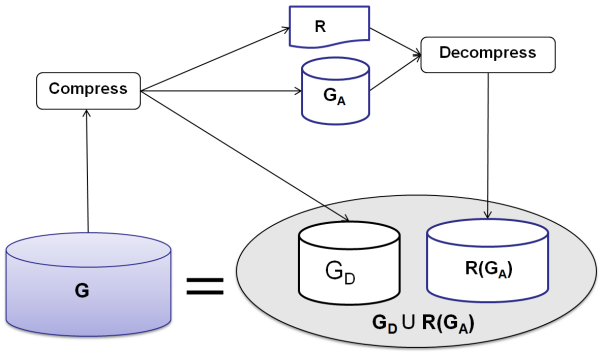

Fig. 4. Rule Based Compression, $G = G_D \cup R(G_A)$

Figure 4 depicts the high level overview of Rule Based Compression technique. 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 an *Active Graph,* G_A, and a *Dormant Graph,* G_D, such that: $G = R(G_A) \cup G_D$ where R represents the set of *decompression rules* to be applied to the active graph G_A during decompression. $R(G_A)$ is the graph resulting from this application.

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. An RB compression is a 3-tuple (G_A, G_D, R), where $G_D \subset G$ is a dormant graph containing some triples $T_D \subset T$, G_A is an active graph containing $T_A \subset T - T_D$ triples and R is a set of decompression rules that is applied to G_A (denoted by $R(G_A)$) producing a graph containing exactly the set $T - T_D$ of triples.*

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

3.1 Intra-property RB Compression

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 associated with k. This rule will be used to expand compressed data given in G_A as follows:

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

Algorithm 1 Intra-property RB compression

Require: G
1: $R \leftarrow \phi$, $G_D \leftarrow \phi$, $G_A \leftarrow \phi$
2: **for each** property, p that occurs in G **do**
3: 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 set of corresponding objects.
4: generate $\{(k, F_k)\}$ set of frequent patterns
5: **for all** (k, F_k) **do**
6: select v_k such that
7: $\sigma(v_k) = argmax_v\{\sigma(v) | v \text{ occurs in } F_k, |v| > 1\}$
8: $R \leftarrow R \cup (k \rightarrow v_k)$ ▷ add a new rule
9: **end for**
10: **for each** $(s,t) \in D$ **do**
11: **for each** $(k \rightarrow v_k) \in R$ **do**
12: **if** $t \cap v_k = v_k$ **then**
13: $G_A \leftarrow G_A \cup (s,p,k)$ ▷ add single triple
14: $t \leftarrow t - v_k$
15: **end if**
16: **end for**
17: **for each** $o \in t$ **do**
18: $G_D \leftarrow G_D \cup (s,p,o)$
19: **end for**
20: **end for**
21: **end for**

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

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

This triple is attached to the active graph G_A so that 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.

3.2 Inter-property RB Compression

In Algorithm 2, we mine frequent patterns across different properties. Transactions used in this algorithm are created by generating a list of all possible pairs of properties 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 $(p : o)$.

Algorithm 2 Inter-property RB compression

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 $\{(k, F_k)\}$ set of frequent patterns
for all (k, F_k) **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 $(k \rightarrow v_k) \in R$ **do**
        if $t \cap v_k = v_k$ **then**
            $G_A \leftarrow G_A \cup (s, p_k, o_k)$ ▷ add single triple
            $t \leftarrow t - v_k$
        end if
    end for
    for each $(p, o) \in t$ **do**
        $G_D \leftarrow G_D \cup (s, p, o))$
    end for
end for

The procedure is similar to one described in 3.1 once frequent patterns and rules are generated.

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

For illustration, here's one such decompression rule we obtained during an experiment on Geonames dataset:
$\forall x$.triple(x, geo:featureCode, geo:V.FRST) $\rightarrow$
triple(x, rdf:type, geo:Feature)
$\wedge$ triple(x, geo:featureClass, geo:V)

3.3 Optimal Frequent Patterns

In this section, we describe optimal rule generation strategy for achieving better compression. In Algorithm 1 and Algorithm 2, we generate frequent patterns and keep only one frequent pattern v per k. By selecting only one frequent pattern per item, it's guaranteed that no circular reference or recursion occurs during decompression. As such, for any given triple in a compressed graph, only one rule can be applied.

The choice of v for k is determined based on whether v has the maximum support. In this section, we present our findings for optimal v pattern selection based on both support value and itemset length. To illustrate this finding, please consider a sample FP-Growth output obtained by mining one of the datasets as shown in Figure 2(a) in section 2.1. If we look at frequent pattern sets for $k = 70$, we have:

1. $(v_1, \sigma_1) = ([22, 70], 56372)$
2. $(v_2, \sigma_2) = ([22, 103, 26, 304, 173, 70], 31084)$
3. $(v_3, \sigma_3) = ([22, 202, 42, 70], 25288)$

The following rule can be applied to select the optimal frequent pattern: select the pattern v_i that maximizes $(|v_i| - 1) \times \sigma_i)$. We call $(|v_i| - 1) \times \sigma_i)$, denoted by $\rho(v_i)$, the *Redundant Triple Density*, signifying the total number of triples that can be removed by using a rule: $(k \rightarrow v_k)$. It is apparent that selecting v_2 during rule generation leads to higher compression than selecting v_1 or v_3.

We call $(|v_i|) \times \sigma_i)$ the *Triple Density* signifying the total number of triples that are associated with this rule.

3.4 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 G_A active graph, G_D dormant graph and a 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. It should

Algorithm 3 Delta Compression

Require: G_A, G_D, R , ΔT_s
1: Extract all triples, T_D, corresponding to s subject from G_D
2: $T \leftarrow T_D \cup \Delta T_s$
3: **for all** $t \in T$ **do**
4: **if** $R(t) \subseteq T$ **then**
5: $G_A \leftarrow G_A \cup t$ ▷ insert into active graph
6: $T \leftarrow T - R(t)$
7: **end if**
8: **end for**
9: **for all** $t \in T$ **do**
10: $G_D \leftarrow G_D \cup t$ ▷ insert into dormant graph
11: **end for**

be noted that we do not create new rules for a new set of triples. As such, the compressed version might not be optimal. A full compression is recommended if a large number of new triples needs to be added or if large number of delta compression have already been performed.

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.

4 Decompression

Decompression can be performed either sequentially or in parallel. Sequential decompression requires applying R decompression rules to triples in G_A active graph and merging these inferred triples with the triples in G_D dormant graph. Since each triple in a compressed graph can belong to at most one rule, it's complexity is $O(|R|.|G_A|)$. The number of rules is negligible compared to the number of triples in the active graph.

For parallel decompression, an active graph can be split into multiple smaller graphs so that each small dataset can perform decompression. This allows generation of inferred triples in parallel. Since rules 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.

5 Experiments

This section shows experimental results of the compression performed by our system. Our experiment is conducted on several linked open datasets as well as synthetic benchmark datasets of varying sizes. The smallest dataset consists of 130K triples while the largest dataset consists of 119 million triples.

5.1 RB Compression - Triple Reduction

Table 1 shows a comparison between the outputs of the two algorithms we discussed in Section 3 for nine different linked open datasets. The compression ratio, r is defined as the ratio of the number of triples in compressed dataset to that in uncompressed dataset. 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 more than 50% of triples for the CN datasets and DBpedia rdftype dataset.

Dataset	**triples** (K)	**predicate**	**transaction** (K)	**compression ratio**	
				intra-property	inter-property
Dog Food	130	132	12	0.98	0.82
CN 2012	137	26	14	0.82	0.43
ArchiveHub	431	141	51	0.92	0.71
Jamendo	1047	25	336	0.99	0.82
LinkedMdb	6147	222	694	0.97	0.75
rdftypes	9237	1	9237	0.19	0.19
RDF About	17188	108	3132	0.97	0.84
DBLP	46597	27	2840	0.96	0.86
Geonames	119416	26	7711	0.97	0.71

Table 1. Compression ratio (based on triple counts) for various linked open datasets. Number of triples and transactions are shown in multiples of 1000.

5.2 RB Compression - Performance

In addition to the compression ratio, the following metrics are measured to evaluate the performance of the system: a) time it takes to perform RB compression and b) time it takes to perform full decompression.

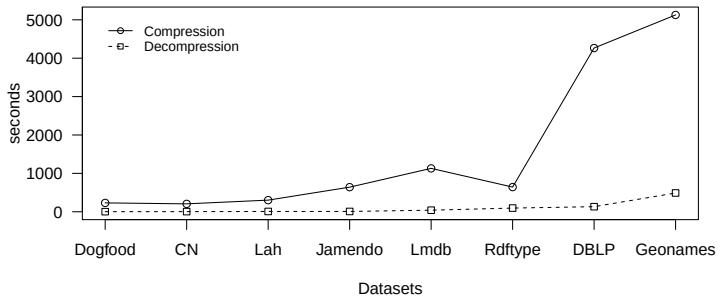

Fig. 5. Compression vs Decompression time for various linked open datasets

Figure 5 shows the comparison between total time required for compression and the total time required for the full decompression. In general, RB compression time increases with the increase in triple size. However, if the total number of predicates in a dataset is very low, as in the case of DBpedia rdftypes dataset, compression time could be significantly lower. Decompression is faster by several order of magnitudes compared to the compression. This can be attributed to the fact that each triple is associated with a maximum of one rule and the number of rules are very few compared to the triple size. In addition, we apply rules only to triples in the Active Graph.

5.3 RB Compression on Benchmark Dataset

In this experiment, we ran RB Compression against one of the mainstream benchmark datasets, LUBM [8]. LUBM consists of a university domain ontology and provides a method for generating synthetic data of varying size.

Table 2 provides details on various LUBM datasets[3] we used for the experiment. Not surprisingly, these results show that compression time on dataset increases with the increase in dataset size. However, the compression ratio remained nearly constant for all the synthetic dataset. Decompression time proved to be far lesser than the time required for compression as seen in Figure 6. It took only 200 seconds for the decompression of the LUBM 1000 dataset compared to 11029 second for the compression.

Dataset	**triples** (K)	**transaction** (K)	**compression** ratio	**Time** sec
LUBM 50	6654	1082	0.763	715
LUBM 100	13405	2179	0.757	1485
LUBM 200	26696	4341	0.757	2513
LUBM 500	66731	10847	0.757	6599
LUBM 1000	133573	21715	0.757	11029

Table 2. Compression ratio and time for various LUBM datasets. Number of triples and transactions are shown in multiples of 1000.

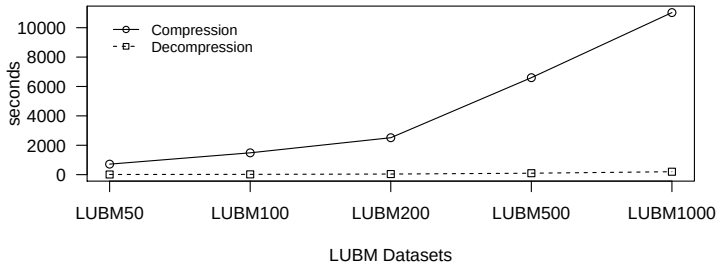

Fig. 6. Compression vs Decompression time for various LUBM datasets

[3] LUBM datasets created with index and seed set to 0.

5.4 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. [5] compared different universal compressors and found that bzip2[4] 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 3). An advantage of semantic compression such as RB Compression is that one can still apply syntactic compression (e.g. HDT) to the results. HDT [6] 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 ran HDT on the compressed dataset resulting from RB Compression. The experimental results are shown in Table 3. We see that this integration does not always lead to a better compression. This is due to the overhead of header and dictionary that HDT creates for both active and dormant dataset[5].

Dataset	Size	compressed	**compressed size using bzip2**		
			HDT	inter-property	HDT + inter-
DogFood	23.4 MB	1.5 MB	1088 K	1492 K	1106 K
CN 2012	17.9 MB	488 K	164 K	296 K	144 K
Archive Hub	71.8 MB	2.5MB	1.8 MB	1.9 MB	1.7MB
Jamendo	143.9 MB	6 MB	4.4MB	5.6 MB	4.6 MB
LinkedMdb	850.3 MB	22 MB	16 MB	22.6 MB	14.5MB
DBpedia rdftypes	1.2 GB	45 MB	11 MB	17.9 MB	10.1 MB
DBLP	7.5 GB	265 MB	201 MB	239 MB	205 MB
Geonames	13 GB	410 MB	304 MB	380 MB	303 MB

Table 3. Comparison of various compression techniques based on dataset size

6 Soundness and Completeness

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

[4] http://bzip2.org

[5] If both these graphs are merged and HDT is performed, the resulting size will be always lesser than that obtained when only HDT is used for compression.

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 [10] Web ontology Language. Indeed, a triple such as (x, p, k) can be expressed in OWL, e.g., in the form[6] $k(x)$ 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.

7 Related work

To the best of our knowledge, this is the first work that investigates practical rule based logical compression of RDF datasets which removes triples to achieve compression. Most of the existing compression techniques focus on compact representation of RDF data as a means of compression. Turtle, a sub-language of N3, is one such compact and natural text representation for RDF data. [5] has explored various compression techniques for RDF datasets and observed that most RDF datasets are highly compressible due to it's power-law distribution in term-frequencies, schemas and resources. [6] introduced a more compact representation format, HDT, by decomposing an RDF data source into Header, Dictionary and Triples. A specific compressed version of HDT, HDT-compressed, outperforms most of the universal compressors [6]. [19, 21] studied the problem of redundancy elimination on RDF graphs in the presence of rules, constraints and queries. [24] uses distributed dictionary encoding with MapReduce to compress large RDF datasets.
Work on frequent itemset mining [1, 9, 15, 26, 20, 27] provides a foundation for our algorithms. [4] explored pattern mining based compression schemes for web graphs specifically designed to accomodate community queries. [25] used association rule mining techniques for generating ontology based on rdf:type statements.

[6] We use description logic notation for convenience, see [11].

8 Conclusion

In this paper, we have introduced a novel lossless compression technique called Rule Based 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 dataset 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 more than half the triples without losing data integrity. This finding is promising and should be explored further for achieving better compression. In future work, we will investigate the use of RB Compression in instance alignment and automated schema generation.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on Management of data. pp. 207–216. SIGMOD '93, ACM (1993)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases. pp. 487–499. VLDB '94, Morgan Kaufmann Publishers Inc. (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. Buehrer, G., Chellapilla, K.: A scalable pattern mining approach to web graph compression with communities. In: Proceedings of the 2008 International Conference on Web Search and Data Mining. pp. 95–106. WSDM '08, ACM (2008)
5. Fernández, J.D., Gutierrez, C., Martínez-Prieto, M.A.: RDF compression: Basic approaches. In: Proceedings of the 19th international conference on World wide web. pp. 1091–1092. WWW '10, ACM (2010)
6. Fernández, J.D., Martínez-Prieto, M.A., Gutierrez, C.: Compact representation of large RDF data sets for publishing and exchange. In: Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I. pp. 193–208. ISWC'10, Springer-Verlag, Berlin, Heidelberg (2010)
7. Goethals, B.: Survey on frequent pattern mining. Tech. rep. (2003)
8. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. Journal of Web Semantics 3(2-3), 158–182 (Oct 2005)
9. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data. pp. 1–12. SIGMOD '00, ACM (2000)
10. 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/
11. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
12. Huang, J., Abadi, D.J., Ren, K.: Scalable SPARQL querying of large RDF graphs. PVLDB 4(11), 1123–1134 (2011)

13. Iannone, L., Palmisano, I., Redavid, D.: Optimizing RDF storage removing redundancies: An Algorithm. In: Proceedings of the 18th international conference on Innovations in Applied Artificial Intelligence. pp. 732–742. IEA/AIE'2005, Springer-Verlag (2005)
14. Joshi, A.K., Hitzler, P., Dong, G.: Towards logical linked data compression. In: Proceedings of the Joint Workshop on Large and Heterogeneous Data and Quantitative Formalization in the Semantic Web, LHD+SemQuant2012, at the 11th International Semantic Web Conference, ISWC2012 (2012)
15. Li, H., Wang, Y., Zhang, D., Zhang, M., Chang, E.Y.: PFP: Parallel FP-Growth for query recommendation. In: Proceedings of the 2008 ACM conference on Recommender systems. pp. 107–114. RecSys '08, ACM (2008)
16. Li, Q., Feng, L., Wong, A.K.Y.: From intra-transaction to generalized inter-transaction: Landscaping multidimensional contexts in association rule mining. Inf. Sci. 172(3-4), 361–395 (2005)
17. Lu, H., Feng, L., Han, J.: Beyond intratransaction association analysis: mining multidimensional intertransaction association rules. ACM Trans. Inf. Syst. 18(4), 423–454 (2000)
18. Manola, F., Miller, E., McBride, B.: RDF primer (2004), `http://www.w3.org/TR/rdf-primer/`
19. Meier, M.: Towards rule-based minimization of RDF graphs under constraints. In: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems. pp. 89–103. RR '08, Springer-Verlag, Berlin, Heidelberg (2008)
20. Özdogan, G.Ö., Abul, O.: Task-parallel fp-growth on cluster computers. In: International Symposium on Computer and Information Sciences. pp. 383–388. Lecture Notes in Electrical Engineering, Springer (2010)
21. Pichler, R., Polleres, A., Skritek, S., Woltran, S.: Redundancy elimination on RDF graphs in the presence of rules, constraints, and queries. In: Proceedings of the Fourth international conference on Web reasoning and rule systems. pp. 133–148. RR'10, Springer-Verlag, Berlin, Heidelberg (2010)
22. Savasere, A., Omiecinski, E., Navathe, S.B.: An efficient algorithm for mining association rules in large databases. In: Proceedings of the 21th International Conference on Very Large Data Bases. pp. 432–444. VLDB '95, Morgan Kaufmann Publishers Inc. (1995)
23. Srikant, R., Vu, Q., Agrawal, R.: Mining association rules with item constraints. In: KDD. pp. 67–73 (1997)
24. Urbani, J., Maassen, J., Drost, N., Seinstra, F.J., Bal, H.E.: Scalable RDF data compression with MapReduce. Concurrency and Computation: Practice and Experience 25(1), 24–39 (2013)
25. Vlker, J., Niepert, M.: Statistical schema induction. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., Leenheer, P., Pan, J. (eds.) The Semantic Web: Research and Applications, Lecture Notes in Computer Science, vol. 6643, pp. 124–138. Springer Berlin Heidelberg (2011)
26. Zaïane, O.R., El-Hajj, M., Lu, P.: Fast parallel association rule mining without candidacy generation. In: Proceedings of the 2001 IEEE International Conference on Data Mining. pp. 665–668. ICDM '01, IEEE Computer Society (2001)
27. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. In: KDD. pp. 283–286 (1997)
28. Zhang, H., Zhang, B.: Generalized association rule mining algorithms based on multidimensional data. In: CONFENIS (1). pp. 337–342 (2007)
29. Zhou, A., Zhou, S., Jin, W., Tian, Z.: Generalized multidimensional association rules. J. Comput. Sci. Technol. 15(4), 388–392 (2000)