Predicate Invention Based RDF Data Compression

Man Zhu[1], Weixin Wu[1], Jeff Z. Pan[2], Jingyu Han[1], Pengfei Huang[3], and Qian Liu[1]

[1] School of Computer Science, Nanjing University of Posts and Telecommunications, China
mzhu@njupt.edu.cn
[2] Department of Computing Science, University of Aberdeen, Aberdeen, UK
[3] College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, China

Abstract. RDF is a data representation format for schema-free structured information that is gaining speed in the context of semantic web, life science, and vice versa. 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 (triples), called PIC (Predicate Invention based Compression). By generating informative predicates and constructing effective mapping to original predicates, PIC only needs to store dramatically reduced number of triples with the newly created predicates, and restoring the original triples efficiently using the mapping. These predicates are automatically generated by a decomposable forward-backward procedure, which consequently supports very fast parallel bit computation. As a semantic compression method for structured data, besides the reduction of syntactic verbosity and data redundancy, we also invoke semantics in the RDF datasets. Experiments on various datasets show competitive results in terms of compression ratio.

1 Introduction

The Resource Description Framework (RDF) is gaining widespread momentum and acceptance among various fields, including science, bioinformatics, business intelligence and social networks, to mention a few [4]. For instance, Semantic-Web-style ontologies and knowledge bases with millions of facts from DBpedia [2], Probase [10], Wikidata [9] and Science Commons [11] are now publicly available. Studies like IDCs Digital Universe1 estimate that the size of the digital universe turned 1Zb (1 trillion Gb) for the first time in 2010, reached 1.8Zb just one year later in 2011 and will go beyond 35Zb in 2020. Combined with the growing size of the overall Linked Open Data cloud, with more than 30 billion triples, and of its individual datasets, with some of its hubs e.g. DBPedia exceeding 1,2 billion triples, the need of effective RDF data compression techniques is clear [8].

Current approaches to achieve lossless RDF document compression can be categorized into three categories, a.k.a., universal file compression techniques, RDF serialization approaches, and rule based compression methods. First of all, universal file compression techniques, such as bzip [4] and LZMA [5], can be applied on RDF document. Such approaches alter the file structure of RDF documents and can significantly

[4] http://www.bzip.org
[5] http://www.7-zip.org/

reduce file size [8] without reducing file content. In RDF-3X [7], all triples are sorted lexicographically in a clustered B^+-tree, and all literals are replaced by ids using a mapping dictionary. Alternative RDF serializations, such as HDT serialization, lean graphs [5] and K2-triples [1] can be used to reduce file size. Such techniques preserve the structured nature of RDF documents, but they fail to utilize the semantics when compressing documents. Other approaches are based on logical compression [6][8], which can be used to reduce the number of triples in an RDF document. Our approach lies in the third category who offers a more significant reduction.

	Person	Professor	Female
Joan	1	0	1
Mike	1	1	0
Teddy	1	1	1

Fig. 1. An example of a toy RDF dataset.

In this paper, we present PIC, a predicate invention based compression method for RDF data. PIC compresses RDF triples by inventing new informative predicates instead of isolated predicates. As a result, the amount of necessary triples (constructed with the newly created predicates) to losslessly decompress the original dataset is dramatically reduced. The newly invented predicates are generated by a forward-backward procedure where very fast parallel bit computation is supported. We also define a map function to locate original predicates and restore the original dataset. To speed-up compression, we propose to use divide and conquer strategy to divide large datasets into smaller ones.
Example. To intuitively illustrate the idea, let us consider Fig. 1 for an example. Fig. 1 depicts a toy RDF dataset containing 3 predicates, a.k.a. Person, Professor, and Female, and 3 instances (Joan, Mike, and Teddy). The element in the table is 1 iff the instance on its corresponding row is of the type (predicate) on the column, such as Person(Joan). There are 7 triples in this dataset, which corresponds to the number of 1s in the table. In this example, instance Joan belongs to both Person and Female, Mike belongs to both Person and Professor, and Teddy belongs to all predicates. By PIC, 3 integer labels are automatically generated after compression, whose corresponding DL (description logic, the logic foundation of semantic web) expressions are Person⊓Female (a concept of the intersection of Person and Female), Person⊓Professor, Person⊓Female⊓Professor. In order to restore the original dataset, only Person⊓Female⊓Professor(Teddy), Person⊓Female(Mike), and Person⊓Professor(Joan) are needed. PIC only saves the 3 integers and the indexes of the literal predicates and instances. In decompression, a mapping algorithm finds the original indexes according to these integer labels.

2 Preliminary

Resource Description Framework (RDF) is the most widely used data interchange format on the Semantic Web. Given a set of URI references $\mathcal{R}$, a set of literals $\mathcal{L}$ and a set

of blank nodes $\mathcal{B}$, an RDF statement is a triple $< s, p, o >$ on $(\mathcal{R} \cup \mathcal{B}) \times \mathcal{R} \times (\mathcal{R} \cup \mathcal{L} \cup \mathcal{B})$. , where s, p, o are the subject, predicate and object of the triple, respectively. An RDF document is a set of triples. An RDF document can be transformed into a 0-1 matrix M by corresponding rows to subjects, columns to predicate and object combinations, and set M_{ij} to 1 iff $< s, p, o >$ belongs to the RDF document, where s is the subject that corresponds to the i-th row, and $p - o$ combination (can roughly be understood as a predicate $\exists p.\{o\}$) corresponds to the j-th column. In this paper, we consider the matrix representation of RDF documents.

3 Approach

PIC generates predicates so that at most one triple is necessary for each subject, and by an effective mapping function the original dataset can be losslessly restored. In this section, we firstly introduce the compression and decompression algorithm, and then we discuss some speed-up techniques of PIC.

3.1 Compression

The compression algorithm is consist of a forward and a backward procedure as shown in Algorithm 1. The input of Algorithm 1 is the data matrix representation (c.f. Section 2) of the RDF document to be compressed. The output contains all necessary triples to be stored. In the forward procedure, we iteratively conjunct columns in bits, and cache the matrices D^i calculated in each outer iteration. The j-th column is conjunctions of columns $c_1, c_2, \ldots, c_i$ (c.f. line 3 of Algorithm 1). In D^i, an element D^i_{jk} is 1 when all elements are 1 on some set of columns of the j-th row in the original RDF data matrix, whose indexes are mapped from k by intoComp function:
intoComp. Given k the column index in D^i, and i the number of columns in the column conjunction, we iterate to find the index of columns $\{c_1, c_2, \ldots\}$ in D^i:

$$C^j_{c_i} \leq k - M < C^j_{c_i+1}$$
$$M = M + C^{j-1}_{c_{i-1}}$$

where c_{i-1} is the index found in the previous iteration. j initially is set to i, and is decreased by 1 in each iteration. M is set to 0 in the first iteration. When $k - M = 0$, the indexes of the rest columns are set from 0, and added by 1 each time, until all columns have been found. For example, intoComp(0, 2) = $\{0, 1\}$.

The backward procedure iteratively updates previously generated matrices D^i. The 1 elements $D^{i'}_{jk}$ in $D^{i'}$ mean that there are triples relate to subject s_j whose predicate and object combinations correspond to the set mapped from k by intoComp. Since intoComp deals with the mapping issues, we only need to store the following data in the compressed dataset:

- i: the matrix superscript, which indicates the number of columns involved in the conjunction;

- j: the row index which corresponds to the index in subjects list;
- k: the column index, together with i, encodes a set of predicate and object combinations.

Fig. 2 depicts an example of Algorithm 1. During forward, the second and third matrix contains values for conjunctions of 2 columns and 3 columns. Because there are 3 columns in the original matrix, there is no need to find D^4. In the right figure, we can find a 1 in the first matrix, namely D^3, which means that there are 3 triples relate to the third subject, so a triple $< 3, 2, 0 >$ is needed. Meanwhile, since the 1s on the 3rd row have been saved, there is no need to consider them in the former matrices. So D^2 is updated because intoComp(0, 3) returns $\{0, 1, 2\}$. In the same way, $< 2, 0, 1 >$ and $< 2, 1, 0 >$ are also stored. In the compressed dataset, only 3 triples are needed.

Algorithm 1: PIC Compression Algorithm

Input : data matrix D
Output: list $\{< i, j, k >\}$
/* forward procedure */
for *i from 2 to* N_C **do**
  for *j from 0 to* $C^i_{N_C} - 1$ **do**
    $D^i_{\cdot j} = D_{\cdot c_1} \wedge D_{\cdot c_2} \wedge \ldots \wedge D_{\cdot c_i}$, where intoComp$(j, i) = \{c_1, c_2 \ldots, c_i\}$;
  if D^i *all zero* **then**
    break;
initialize empty list L;
/* backward procedure */
foreach D^i **do**
  $D^{i'} = D^i$;
  for each $D^{i+1}_{jn} \neq 0$, set $D^{i'}_{jk} = 0$ where $k \in$ intoComp(n, i);
  add $< i, j, k >$ to$L$, where $D^{i'}_{jk} \neq 0$;
return L;

3.2 Decompression

We briefly introduce the decompression procedure here, c.f. Algorithm 2. We take a list $< i, j, k >$ as input. Each triple decompresses to a set of triples by locating their indexes through intoComp function. R is the set of predicate-object combinations, and $\mathcal{I}_i$ is the i-th subject. Suppose R_j is the combination of p_t and o_k, then RDF triple $< \mathcal{I}_i, a, R_j >$ is transformed to $< \mathcal{I}_i, p_t, o_k >$.

3.3 Speeding-up Strategy

In order to evidently reduce the column conjunctions, we adopt a partition and merge procedure. The objective of this procedure is to divide the RDF data matrix into a set of independent partitions, where the number of columns in each partition is decreased.

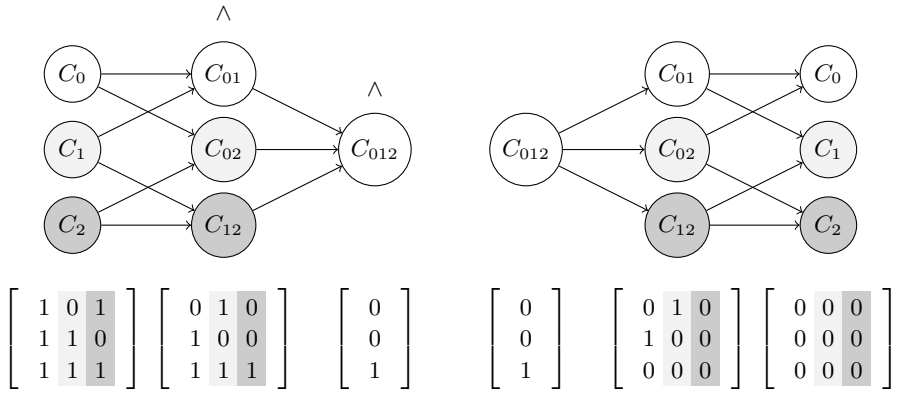

Fig. 2. An example of compression. Left and right correspond to forward and backward procedure respectively. Values of node in matrix are shown in the same color as the node.

Algorithm 2: PIC Decompression Algorithm

Input : list $\{< i, j, k >\}$
Output: D'
foreach $< i, j, k > \in \{< i, j, k >\}$ **do**
  $R = \{\mathcal{R}_j\}$, where j $\in$ intoComp(k, j);
  store RDF triple $< \mathcal{I}_i$, a, $R_j >$, where $R_j \in R$;

As depicted in figure 3, given an RDF data matrix, affixed with a first row indicating the original index of the property column, this procedure reorder its columns and recursively divide the matrix horizontally in the middle, until the size of the resulted matrices are small enough. The column reordering scheme is that first part are the columns containing both 1s and 0s in upper and blow, the second part containing 1s only in upper, and the third part containing 1s only in below. In the end, the repeated columns are merged, the repeat column indexes are recorded. The smaller matrix, the repeat column indexes, and a path string indicating its location is called a partition. For example, the path string of the left partition is "0" because the smaller matrix is the upper part of the previous one, and the right partition is "1" because it is the lower part.

When compressing the data partitions, the decompression slightly differs. Assume the matrix in the partition is M, $M-1$ is the matrix whose path string equals to M's eliminating the last character, last(M) is short for the last character of M's path string, and rows is the number of rows in the original matrix. Then the first row index of M is:

$$loc(M) = \begin{cases} loc(M-1) & \text{if last}(M) \text{ is } 0 \\ loc(M-1) + \lceil \frac{1}{2}\text{offset}(M-1) \rceil & \text{if last}(M) \text{ is } 1 \end{cases}$$

where

$$\text{offset}(M) = \begin{cases} \lceil \frac{1}{2}\text{offset}(M-1) \rceil & \text{if last}(M) \text{ is } 0 \\ \lfloor \frac{1}{2}\text{offset}(M-1) \rfloor & \text{if last}(M) \text{ is } 1 \end{cases}$$

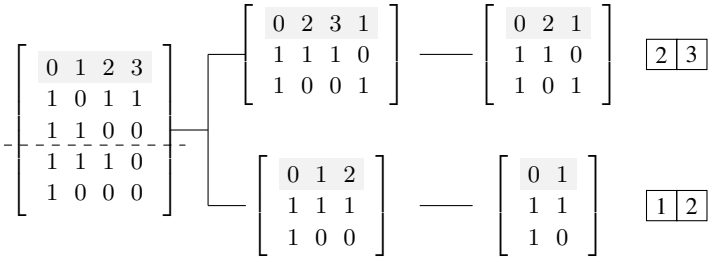

Fig. 3. An example of data partition.

and offset("") = #rows, loc("") = 0. Besides, here are some other speed-up strategies for compression:

- In some D^i, if the number of columns containing 1 is less than $i + 1$, the forward procedure can pre-terminate.
- In some D^i, if the numbers in a column are all 0, this column can be dropped out of consideration in the following-up conjunction calculations.
- We use different stop criterion in finding matrix partitions, since the calculations of bit conjunction are pretty cheap, there is no need to have trivial matrices.

4 Preliminary Evaluation

This section shows preliminary evaluations of the compression performed by our system. Our experiment is conducted on several linked open datasets of varying sizes (cf Tab. 1). The smallest dataset consists of 130K triples while the largest dataset consists of one million triples.

Table 1. Dataset statistics (the first 3 columns) and compression ratio γ (the last 2 columns). LOG stands for the logical linked data compression approach [6].

Dataset	#triples(K)	#predicate-object	size(M)	#triples (PIC)	#triples (LOG)	γ(PIC)	γ(LOG)
Dog Food	130	77,064	20.7	12.7K	106.6K	12.3%	82.0%
CN 2012	137	41,770	17.9	14.6K	58.9K	10.7%	43.0%
ArchiveHub	431	204,360	71.7	51.4K	306.0K	12.0%	71.0%
Jamendo	1047	485,443	143.0	335.9K	858.5K	32.1%	82.0%

The comparisons are mainly based on two metrics: compression ratio and running time. The compression ratio, γ is defined as the ratio of the number of triples in compressed dataset to that in uncompressed dataset [6]. Besides the compression ratio, we also measure time it takes to perform full compression and full decompression.

Now we firstly discuss the stop criterion of the data partition algorithm. Then, we report the experimental results in terms of compression ratio and running time compared

Table 2. Compressed size for various RDF datasets. LOG stands for the logical linked data compression approach [6].

Dataset	size (PIC-head)	size (PIC-triples)	size (PIC+bzip2)	size (LOG+bzip2)
Dog Food	4.4M	649K	1330.0K	1492K
CN 2012	1.1M	492K	226.6K	296K
ArchiveHub	11.7MB	1.81MB	2.3MB	1.9MB
Jamendo	26.7M	4.86M	5.6MB	5.6MB

to the logical linked data compression approach [6]. We compress the resulting datasets (in N-Triples format) by bzip2, which is one of the best universal compressors [3].

To find data partitions, we use 2 thresholds. One threshold controls the maximum number of rows each partition has, and the other dominates the number of columns of each partition. Specifically, on these 4 datasets, we first iterate to find partitions of no more than 400 rows. Then for each dataset, different threshold of columns are adopted according to the running time. The thresholds are 25, 30, 27, and 25 on DogFood, CN 2012, ArchiveHub, and Jamendo respectively.

From Tab. 2 we can find that: 1) the data reduction of PIC is significant, which is due to the discovery of expressive predicates; 2) PIC gets better results on DogFood and CN 2012, but not on ArchiveHub. In ArchiveHub, there are lots of isolated triples, which are not connected with others. However, the benefit of PIC is shrinking multiple triples related to the same subject into at most one triple, which cannot be done on ArchiveHub and Jamendo; 3) it is interesting that on DogFood and CN 2012, the numbers of triples after compression by LOG and PIC show reverse tendency. LOG needs significantly less triples for CN 2012 compared to DogFood, but PIC is the opposite. As mentioned above, PIC is not good at handle single isolated triples, which can be handled by LOG using subsumption axioms. 4) the size of head in PIC is large because we only draw namespaces in RDF data instead of striving to compress texts by lexical methods, since this is not our concern at the moment.

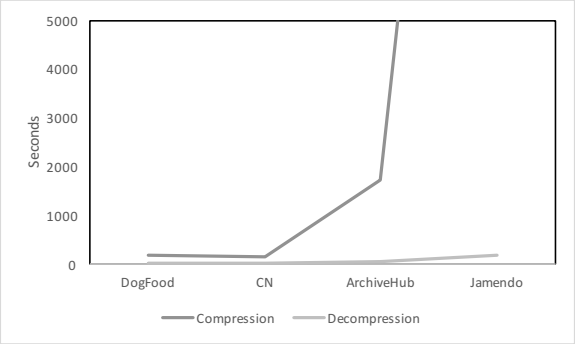

Fig. 4. Running time of compression and decompression by PIC.

Fig. 4 shows the comparison between total time required for compression and the full decompression. In general, the compression time increases with the increase in triple size. However, the compression time is influenced by number of predicate-object triples, which is shown by the compression time of Dog Food and CN 2012. Decompression is faster by several order of magnitudes compared to the compression.

5 Discussion

This paper has presented PIC, a predicate invention based RDF compression method. As our experiments have shown, PIC outperforms other methods on dense datasets in terms of compression ratio and compressed size. These datasets contain multiple triples with a same subject, which are transformed into one triple at most by PIC. As a result, we can find that the data reduction is significant from experimental results. For other datasets having sparse data, PIC is not effective enough. In terms of running time, PIC is efficient during decompression, but is still not efficient enough for compression.

Our future work includes two parts: 1) PIC needs further improvements on the running time of compression; 2) We will find more intelligent way to partition the RDF data matrix, and use PIC to compress dense partitions.

Acknowledgement.This work is partially funded by the National Science Foundation of China under grant 61602260 and 61702279.

References

1. S. Álvarez García, N. R. Brisaboa, J. D. Fernández, and M. A. Martínez-Prieto. Compressed k2-triples for full-in-memory RDF engines. In *AMCIS*, 2011.
2. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *Proc. ISWC'07/ASWC'07*, pages 722–735, 2007.
3. J. D. Fernández, C. Gutiérrez, and M. A. Martínez-Prieto. RDF compression: basic approaches. In *Proc. WWW'10*, pages 1091–1092, 2010.
4. M. Hammoud, D. A. Rabbou, R. Nouri, S. Beheshti, and S. Sakr. DREAM: distributed RDF engine with adaptive query planner and minimal communication. In *Proc. VLDB'15*, pages 654–665, 2015.
5. L. Iannone, I. Palmisano, and D. Redavid. Optimizing RDF storage removing redundancies: An algorithm. In *Innovations in Applied Artificial Intelligence*, volume 3533, pages 732–742. 2005.
6. A. K. Joshi, P. Hitzler, and G. Dong. Logical linked data compression. In *Proc. ESWC'13*, pages 170–184, 2013.
7. T. Neumann and G. Weikum. RDF-3X: A RISC-style engine for RDF. *Proc. VLDB Endow.*, 1(1):647–659, 2008.
8. J. Z. Pan, J. M. Gómez-Pérez, Y. Ren, H. Wu, H. Wang, and M. Zhu. Graph pattern based RDF data compression. In *Proc. JIST'14*, pages 239–256, 2014.
9. D. Vrandečić. Wikidata: A new platform for collaborative data collection. In *Proc. WWW'12*, pages 1063–1064, 2012.
10. W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *Proc. SIGMOD'12*, pages 481–492, 2012.
11. P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang, and L. Liu. TripleBit: a fast and compact system for large scale RDF data. *PVLDB*, 6(7):517–528, 2013.