Join Size Estimation on Boolean Tensors of RDF Data

Saskia Metzler
Max-Planck-Institut für Informatik
Saarbrücken, Germany
saskia.metzler@mpi-inf.mpg.de

Pauli Miettinen
Max-Planck-Institut für Informatik
Saarbrücken, Germany
pauli.miettinen@mpi-inf.mpg.de

ABSTRACT

The Resource Description Framework (RDF) represents information as subject–predicate–object triples. These triples are commonly interpreted as a directed labelled graph. We instead interpret the data as a 3-way Boolean tensor. Standard SPARQL queries then can be expressed using elementary Boolean algebra operations. We show how this representation helps to estimate the size of joins. Such estimates are valuable for query handling and our approach might yield more efficient implementations of SPARQL query processors.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

Keywords

Tensors; Khatri–Rao product; RDF; SPARQL

1. INTRODUCTION

RDF data are commonly treated as a directed labelled graph. While this interpretation is often intuitive, other interpretations are also possible. One alternative is to consider each predicate a relation in standard relational database. This interpretation facilitates the use of existing techniques for faster query evaluation and efficient storage, among others. On the other hand, it is too restrictive to naturally express many SPARQL queries.

We propose an alternative interpretation, where RDF data is considered as a 3-way Boolean tensor. This allows us to present SPARQL queries using tensor slicing and specific types of matrix multiplications. Furthermore, certain optimization techniques will lend themselves very naturally to this setting. In our on-going work, we study what benefits can be gained from this tensor interpretation. In this short paper, we concentrate on one of those benefits: the techniques to estimate the sizes of the join operations. Before that, however, we give a short introduction on tensors and RDF data as a Boolean tensor (please see [4] for a full explanation).

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2742738.

2. RDF DATA AS A BOOLEAN TENSOR

For a 3-way Boolean tensor $\boldsymbol{\mathcal{X}} \in \{0,1\}^{n\times m\times k}$, $x_{:jk} \in \{0,1\}^n$ is the *mode-1 (row) fibre*, $x_{i:k} \in \{0,1\}^m$ is the *mode-2 (column) fibre*, and $x_{ij:} \in \{0,1\}^k$ is the *mode-3 (tube) fibre*. A tensor can also be sliced, e.g. $\boldsymbol{X}_{::k} \in \{0,1\}^{n\times m}$ is the kth *frontal slice*.

Let $\boldsymbol{X} \in \{0,1\}^{n_1\times m_1}$ and $\boldsymbol{Y} \in \{0,1\}^{n_2\times m_2}$. Their *Kronecker product* $\boldsymbol{X} \otimes \boldsymbol{Y} \in \{0,1\}^{n_1 n_2\times m_1 m_2}$ is defined as

$$\boldsymbol{X} \otimes \boldsymbol{Y} = \begin{pmatrix} x_{11}\boldsymbol{Y} & x_{12}\boldsymbol{Y} & \cdots & x_{1m_1}\boldsymbol{Y} \\ x_{21}\boldsymbol{Y} & x_{22}\boldsymbol{Y} & \cdots & x_{2m_1}\boldsymbol{Y} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n_1 1}\boldsymbol{Y} & x_{n_1 2}\boldsymbol{Y} & \cdots & x_{n_1 m_1}\boldsymbol{Y} \end{pmatrix} . \quad (1)$$

The *Khatri–Rao product* of $\boldsymbol{X}$ and $\boldsymbol{Y}$ is defined as "column-wise Kronecker". That is, $\boldsymbol{X}$ and $\boldsymbol{Y}$ must have the same number of columns ($m_1 = m_2 = m$), and their Khatri–Rao product $\boldsymbol{X} \odot \boldsymbol{Y} \in \{0,1\}^{n_1 n_2\times m}$ is defined as

$$\boldsymbol{X} \odot \boldsymbol{Y} = \left(\boldsymbol{x}_1 \otimes \boldsymbol{y}_1, \boldsymbol{x}_2 \otimes \boldsymbol{y}_2, \ldots, \boldsymbol{x}_m \otimes \boldsymbol{y}_m\right) . \quad (2)$$

Given an RDF graph T, let $S(T)$, $P(T)$, and $O(T)$ denote the sets of distinct subjects, predicates and objects, respectively. The number of distinct subjects is denoted by $|S(T)|$, or shorthand $|S|$. Correspondingly $|P(T)| = |P|$ and $|O(T)| = |O|$ denote the number of predicates and objects.

To represent RDF data by a binary tensor, we enumerate all subjects, predicates, and objects to obtain mappings from the items in $S(T)$, $P(T)$, and $O(T)$ to indices. Let s_i denote the ith subject with the corresponding index $i = 1,\ldots,|S|$ and respectively p_j the jth predicate with $j = 1,\ldots,|P|$ and o_k the kth object with $k = 1,\ldots,|O|$.

With this mapping we can represent any RDF graph T as a 3-way binary $|S|$-by-$|P|$-by-$|O|$ tensor $\boldsymbol{\mathcal{T}}$. An element (i,j,k) of $\boldsymbol{\mathcal{T}}$ is 1 if and only if the respective subject-predicate-object triple (s_i, p_j, o_k) is present in the RDF graph T.

A join operation on such data can be expressed by means of a Khatri–Rao product. Consider for example a basic graph pattern consisting of two triple patterns, `{?a` T`:p`$_i$ `?b} . {?c` U`:p`$_j$ `?b}`. This queries for all RDF triples where the object `?b` is linked to a subject by predicate `p`$_i$ of RDF graph T as well as by predicate `p`$_j$ of RDF graph U, where $i \in \{1,\ldots,|P(T)|\}$ and $j \in \{1,\ldots,|P(U)|\}$.

In Boolean tensor algebra these triple patterns resemble the slices $\boldsymbol{T}_{:i:}$ and $\boldsymbol{U}_{:j:}$ of RDF tensors $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\mathcal{U}}$. A join operation on equal objects is equivalent to the Khatri–Rao product of $\boldsymbol{T}_{:i:}$ and $\boldsymbol{U}_{:j:}$. We assume the length of the columns of both slices to match and the labels to be in the same order (this assumption can be lifted using standard techniques). The result to the basic graph pattern (i.e. the join) is a matrix of size $|S(T)|\,|S(U)|$ -by- $|O|$,

$$\boldsymbol{T}_{:i:} \odot \boldsymbol{U}_{:j:}$$
$$= \begin{pmatrix} t_{1i1}\boldsymbol{u}_{:j1} & {}_{1i2}\boldsymbol{u}_{:j2} & \cdots & t_{1i|O|}\boldsymbol{u}_{:j|O|} \\ t_{2i1}\boldsymbol{u}_{:j1} & t_{2i2}\boldsymbol{u}_{:j2} & \cdots & t_{2i|O|}\boldsymbol{u}_{:j|O|} \\ \vdots & \vdots & \ddots & \vdots \\ t_{|S|i1}\boldsymbol{u}_{:j1} & t_{|S|i2}\boldsymbol{u}_{:j2} & \cdots & t_{|S|i|O|}\boldsymbol{u}_{:j|O|} \end{pmatrix} .$$

This matrix has non-zeroes where objects have corresponding subjects when the predicate is $\mathtt{p}_i$ as well as $\mathtt{p}_j$. It can be regarded as $|S(T)|$ blocks of $|S(U)|$ -by- $|O|$ matrices stacked on top of each other. Each block then corresponds to a subject `?a` from T and each row per block corresponds to a subject `?c` from U.

Similarly we can express join queries where another variable than the object is bound. For a detailed analysis on SPARQL queries in the Boolean tensor setting, see the aforementioned technical report.

3. ESTIMATING THE JOIN SIZE

An important problem in query processing is to estimate the cardinality of join operations. While the problem has attracted a significant amount of research in relational databases over the years, much less work addresses the specifics of SPARQL joins, although [5] presents *characteristic sets* to estimate the cardinalities of SPARQL joins. The tensor framework, however, allows us to use techniques similar to relational databases natively with SPARQL queries, as we shall demonstrate below.

Let us assume we have stored the marginal sums along each mode of the $|S|$ -by- $|P|$ -by- $|O|$ data tensor $\boldsymbol{\mathcal{T}}$. That is, we have three matrices, $\boldsymbol{P}$ ($|P|$ -by- $|O|$), $\boldsymbol{Q}$ ($|S|$ -by- $|O|$), and $\boldsymbol{R}$ ($|S|$ -by- $|P|$), for column, row, and tube marginal sums, respectively (for example, $q_{ij} = \sum_{k=1}^{|P|} t_{ijk}$).

The number of triples returned by a join (with no projection or DISTINCT keyword) is the number of non-zeroes in the result. When a join can be expressed as a Khatri–Rao product between two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, the number of non-zeroes in $\boldsymbol{A} \odot \boldsymbol{B}$ can be determined exactly using the column marginal sums of $\boldsymbol{A}$ and $\boldsymbol{B}$. Specifically, if $\boldsymbol{A}$ and $\boldsymbol{B}$ both have n columns, let $\boldsymbol{\sigma}^{\boldsymbol{A}} = (\sigma_i^{\boldsymbol{A}})_{i=1}^n$ and $\boldsymbol{\sigma}^{\boldsymbol{A}} = (\sigma_i^{\boldsymbol{B}})_{i=1}^n$ be row vectors that contain the column marginals of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively (e.g. $\sigma_i^{\boldsymbol{A}} = \sum_j a_{ji}$).

PROPOSITION 3.1. *Let $\boldsymbol{\sigma}^{\boldsymbol{A}}$ and $\boldsymbol{\sigma}^{\boldsymbol{B}}$ be as above. The number of non-zeroes in $\boldsymbol{A} \odot \boldsymbol{B}$ is*

$$|\boldsymbol{A} \odot \boldsymbol{B}| = \sum_{i=1}^{n} \sigma_i^{\boldsymbol{A}} \sigma_i^{\boldsymbol{B}} = \boldsymbol{\sigma}^{\boldsymbol{A}} (\boldsymbol{\sigma}^{\boldsymbol{B}})^T . \qquad (3)$$

The column marginal vectors $\boldsymbol{\sigma}$ are appropriate rows or columns of the tensor marginal sum matrices $\boldsymbol{P}$, $\boldsymbol{Q}$, or $\boldsymbol{R}$. If they are stored in a sparse format, the size of the join can be computed exactly in time $\Theta(\alpha + \beta)$, where α and β are the number of non-empty columns of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively.

We can also obtain an upper bound for the size of the join in constant time if we in addition store the l_2-norms for each row and column of the marginal sum matrices (that is, $\|\boldsymbol{\sigma}\|$ for every possible $\boldsymbol{\sigma}$):

PROPOSITION 3.2. *Let $\boldsymbol{\sigma}^{\boldsymbol{A}}$ and $\boldsymbol{\sigma}^{\boldsymbol{B}}$ be as above. Then*

$$|\boldsymbol{A} \odot \boldsymbol{B}| \leq \|\boldsymbol{\sigma}^{\boldsymbol{A}}\| \|\boldsymbol{\sigma}^{\boldsymbol{B}}\| . \qquad (4)$$

PROOF. Noticing that $\boldsymbol{\sigma}^{\boldsymbol{A}}(\boldsymbol{\sigma}^{\boldsymbol{B}})^T = \left\|\boldsymbol{\sigma}^{\boldsymbol{A}}\right\| \left\|\boldsymbol{\sigma}^{\boldsymbol{B}}\right\| \cos\theta$ together with (3) gives the result as $\cos\theta \leq 1$. $\square$

This estimate can be improved if we can estimate the angle θ between vectors $\boldsymbol{\sigma}^{\boldsymbol{A}}$ and $\boldsymbol{\sigma}^{\boldsymbol{B}}$. To that end, Charikar [3] gives a locality sensitive hashing scheme where the collision probability for vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ hashed by a function $h_{\boldsymbol{r}}$ is

$$\mathbf{Pr}\left[h_{\boldsymbol{r}}(\boldsymbol{u}) = h_{\boldsymbol{r}}(\boldsymbol{v})\right] = 1 - \frac{\theta(\boldsymbol{u}, \boldsymbol{v})}{\pi} . \qquad (5)$$

The hash function $h_{\boldsymbol{r}}(\boldsymbol{x}) = 1$ if $\boldsymbol{r}\boldsymbol{x}^T \geq 0$, and $h_{\boldsymbol{r}}(\boldsymbol{x}) = 0$ otherwise. The random vector $\boldsymbol{r}$ is drawn from a multidimensional Gaussian distribution.

To estimate θ, we evaluate k hash functions $h_{\boldsymbol{r}_1}, \ldots, h_{\boldsymbol{r}_k}$ for each row and column marginal of the tensor slices and store the results as k-dimensional bit vectors $\boldsymbol{h}(\boldsymbol{\sigma})$ for every possible $\boldsymbol{\sigma}$ (in total $2(|S| + |P| + |O|)$ vectors). Then, for two vectors $\boldsymbol{\sigma}^{\boldsymbol{A}}$ and $\boldsymbol{\sigma}^{\boldsymbol{B}}$, the join cardinality can be estimated in $O(k)$ time by

$$\theta = \pi \cdot \left(1 - \mathbf{Pr}\left[\boldsymbol{h}(\boldsymbol{\sigma}^{\boldsymbol{A}}) = \boldsymbol{h}(\boldsymbol{\sigma}^{\boldsymbol{B}})\right]\right) . \qquad (6)$$

The larger k is, the more accurate the estimate is. Charikar [3] points out that in order to hash n vectors, it is enough to evaluate $O(\log^2 n)$ randomly chosen hash functions.

This approach is most valuable for systems where many queries are made but the data are rarely edited. The initial computation of the hashes is costly but then the estimate of θ and hence the cardinality of the join can be computed fast. If the data are edited more often, we should either store the full marginal vectors $\boldsymbol{\sigma}$ or use techniques similar to [1] for dynamically tracking the join sizes.

Finally, it is worth noting that in case of the DISTINCT keyword, the problem of estimating the join size returns to the problem of estimating the size of a Boolean matrix product, for which there are existing methods [2] (see the aforementioned technical report for details).

4. CONCLUSIONS

We interpreted RDF data as a Boolean tensor instead of a graph. This view admits for defining SPARQL operations by means of Boolean algebra. This facilitates the application of methods designed for relational data and matrices to RDF data and SPARQL queries. As an example of that, we presented methods to estimate the join sizes, but improvements on other aspects of SPARQL query processing should also be possible.

5. REFERENCES

[1] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. *PODS '99*, pages 10–20, 1999.

[2] R. Amossen, A. Campagna, and R. Pagh. Better size estimation for sparse matrix products. *Algorithmica*, 69(3):741–757, 2014.

[3] M. S. Charikar. Similarity estimation techniques from rounding algorithms. *STOC '02*, pages 380–388, 2002.

[4] S. Metzler and P. Miettinen. On defining SPARQL with Boolean tensor algebra, 2015. arXiv:1503.00301 [cs.DB].

[5] T. Neumann and G. Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *ICDE '11*, pages 984–994, 2011.