SYNTHG: Mimicking RDF Graphs Using Tensor Factorization

Abdelmoneim Amer Desouki
Felix Conrads
Michael Röder
and Axel-Cyrille Ngonga Ngomo
DICE group, Department of Computer Science, Paderborn University
Email: desouki@mail.upb.de, michael.roeder@upb.de, axel.ngonga@upb.de

***Abstract*—There is a need for synthetic graphs to help benchmarking efforts. Synthetic graphs that mimic real-world graphs can be used to avoid sending sensitive information to third parties while preserving topological characteristics of the input original graph. They can also be used to evaluate the scalability of different algorithms since the size of synthetic graphs can be scaled. In view of these applications, we introduce a novel approach to mimik RDF graphs. Our approach introduces a random rotation in the tensor factorization of the input RDF graph. By combining this matrix with the core tensor computed by the factorization, our approach can generate a graph which maintains the querying characteristics of the input graph, while not permitting a reconstruction of the input graph. We use Semantic Web Dog Food and DBpedia 2016 to evaluate our approach and compare the original, reconstructed and synthetic graphs by using them to benchmark five triple stores. The results show that the Pearson correlation between the performance of the triple stores under original and synthetic graphs is 0.91, 0.64 for Semantic Web Dog Food and DBpedia respectively. Our results also suggest that the synthetic graphs inherit the main graph characteristics of the original graphs. SYNTHG is open-source and is available at: https://github.com/dice-group/SynthG**

I. INTRODUCTION

There is a growing interest in using RDF *knowledge graph* [1] to represent data (See Section II). Therefore, there are a lot of algorithms to deal with such data representations. Synthetic graphs can be used as means of testing algorithms on different scales or on anonymised data [1], [2]. In the latter use case, synthetic graphs facilitate sending sensitive information to third parties. However, to make the usage of synthetic graphs attractive they should opt similar features as real-world graphs. Synthetic graphs are usually used for benchmarking purposes to test performance of triplestores as well as different query strategies [3], [4]. In this case synthetic graphs should give similar performance metrics to the real ones with the advantage to give different scales and different instances of real graphs.

We introduce a novel approach to generate synthetic KGs which mimic real-world KGs. Our approach, dubbed SYNTHG, employs tensor factorization to capture the main features of a given real-world dataset. Then, we apply a distance-preserving rotation to the factors. Finally, we generate the synthetic graph from the rotated factors. Our use of a rotation ensures that our approach maintains the latent topology of the original graph while ensuring that the original data cannot be reconstructed. Our results include an open-source and scalable implementation of the RESCAL tensor factorization [5] and a scalable generation approach for synthetic KGs able to process graphs of the scale of DBpedia.

The structure of the rest of the paper is as follows: we begin by presenting the preliminaries to our approach in Section II. Then, we present the details of our approach in Section III. We evaluate SYNTHG in Section IV on two different datasets. An overview of the work related to SYNTHG is given in Section V. Finally the conclusion is given in Section VI.

II. PRELIMINARIES

A. Tensor Factorization

Let the following sets be mutually disjoint: F is the set of URI references, B is the set of blank nodes, L is the set of literals [6]. Further, let $P \subseteq F$ be the set of all properties. An RDF $\mathcal{G}$ is a set of triples. A triple $t = (s, p, o) \in (F \cup B) \times P \times (F \cup B \cup L)$ displays the statement that the subject s is related to the object o via the predicate p [6]. Three-mode tensors can be used to represent RDF KGs [7], [8], [9]. For each known triple in a KG $\mathcal{G}$, the entry in the tensor will be 1 and 0 otherwise. So, An element x_{ijk} of a three-way tensor $\mathcal{X}$ representing $\mathcal{G}$ is

$$x_{ijk} = \begin{cases} 1, & \text{if } (e_i, e_j, e_k) \in \mathcal{G}, \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where e_i, e_j and e_k are the URIs (also known as *resources*) with indices i, j, and k respectively. One advantage of this tensor representation is that it can be folded laterally. This means that each predicate (relation) can be represented by a square adjacency matrix of dimension $(N \times N)$ where N is the number of resources. Tensor factorization then can be used to study the properties of such tensors [9], [7].

In particular, the RESCAL tensor factorization learns an embedding of resources of dimension r (r is the rank of the factorization) and a core tensor R (dimensions: $r \times r \times m$, where m is the number of predicates) [7], [8], [5]. So RESCAL tries to find a matrix A (dimensions: $N \times r$) and a core tensor R such that $\mathcal{X}_k \approx AR_kA^T$, for $k = 1, ..., m$. A can be seen as array of embedding.

[1]Resource Description Framework, see https://www.w3.org/RDF/

B. Matrix Decomposition

Our approach relies on Singular Value Decomposition (SVD) of matrices A (Section III). Given any $m \times n$ matrix A, the aim of SVD is to compute the three matrices U, D, and V such that A=UDV^T where U is a $m \times m$-unitary matrix, D is a $m \times n$ diagonal matrix with non-negative values in the diagonal and V is a $n \times n$-unitary matrix. When A is real, U and V^T are orthonormal matrices.

The QR decomposition of a matrix A is a decomposition that finds two matrices Q and W such that $A = QW$ where Q is an orthogonal matrix (i.e. $QQ^T = Q^TQ = I$ or columns are orthogonal unit vectors) and W is an upper triangle matrix.

III. SYNTHG

The outline of SYNTHG is as follows:

1) Materialize the dataset and represent it as a binary trimodal tensor $\mathcal{X}$.
2) Calculate $\mathcal{X}$'s RESCAL factorization (A and R).
3) Apply a random rotation to A to get A'.
4) Generate the synthetic graph by getting top values of $A'RA'^T$.

A. Materialising the dataset

The algorithm starts by preprocessing the dataset. To ensure that all triples are made explicit, forward chaining is applied to add missing triples, e.g., in case of symmetric or inverse relations. Then, we represent the graph as a set of lateral slices (one slice per predicate). Given that most pairs of resources are not connected by a given predicate each slice of the tensor $\mathcal{X}$ is represented as a sparse matrix.

B. Tensor Factorization using RESCAL

The next step is to factorize the tensor representing the dataset. We use RESCAL for this task. To make RESCAL able to handle tensors at scale we modified several parts of the original algorithm implementation. Four of these modifications were suggested in [8]. First, we modified the step of calculating the error to test convergence. Instead of calculating the error between the approximation and the original tensor, we check the significance the changes made per iteration in A and R. The second modification suggested in [8] is to change the implementation of the update step of A and R to make use of parallelisation. The third modification is to initialize A with the eigenvectors of the matrix that sums the adjacency matrices of all predicates (lateral slices in the tensor). This initialization makes the convergence much faster. The fourth is to add the *TBox* triples to the graph (i.e the triples identifying the Ontology) as the authors of [8] proposed that this guides the factorization to the real concepts of the dataset. In addition to that, a major speedup ($\approx$4 times faster) is gained by using more compact matrices in representing the tensor. These compact matrices store the indices of rows (I_c) and columns (J_c) that contain at least one nonzero element. That allows the matrix to shrink to dimensions $|I_c| \times |J_c|$ instead of $N \times N$. These matrices avoid a lot of zero by zero multiplications which were done when multiplying sparse matrices.

C. Random Rotation

To apply a transformation to A that preserves the distance between columns (i.e. factors), we first decompose A using the SVD decomposition. Then we replace the rotation matrix U in the result by a random rotation U'. We get the random rotation in two steps: first, we generate a random matrix Rm of the same size as U (Rm is random but not rotation matrix). Then, we apply QR decomposition to Rm to get the rotation matrix. We use the matrix Q of the decomposition result as U'. Finally A' is calculated by multiplying matrices U', D, and V^T as shown in Algorithm 1.

Algorithm 1: Apply a random rotation to matrix A

Data: A: embedding matrix
Result: A': rotated embedding
$[U, D, V] \leftarrow svd(A)$ ▷ get SVD of A
Rm $\leftarrow$ get random matrix of dimensions as U
$U' \leftarrow qr.Q(qr(Rm))$ ▷ find QR decomposition
$A' \leftarrow U'DV^T$ ▷ get the randomly rotated A into A'
return A';

The random rotation via substituting U with U' preserves inter-column Euclidean distances.

D. Generating triples from RESCAL factorization

After A' has been generated, the new, synthetic tensor $\mathcal{X}'$ can be calculated. To this end, we take the top N_p scores of $A'R_pA'^T$ for each predicate p. We provide two algorithms to find the elements with the top scores—one algorithm for small graphs and a second algorithm for large graphs. Both of them are based on partitioning the matrix A' into chunks. Then we use the *quantile* function to give a threshold of top N_p (quantile $= 1 - N_p/Number\_of\_values\_in\_the\_chunk$). At the end of the procedure we apply the same function to all top values yielded from all chunks to select from them N_p triples. To avoid multiplications which will result in too small values we modified the algorithm as follows: by using a constraint on the maximum possible result of the multiplication of two rows without doing the multiplication. Also we sort the columns and rows descending according to the maximum possible value.

IV. EVALUATION

We evaluate SYNTHG on two different datasets by measuring the correlation between the performance of triple stores on real and synthetic KGs and by comparing graph characteristics of the original graphs and the synthetic ones.

A. Experimental setup

The first RDF graph we use to evaluate SYNTHG, is Semantic Web Dogfood (SWDF) [2]. SWDF contains linked data about conferences of the Semantic Web community, papers published at these conferences and authors of the papers. We used the already materialised version. The second dataset is DBpedia 2016-04 (DBP).[3] We materialised the graph using

[2]SWDF: http://www.scholarlydata.org/dumps/conferences/alignments/
[3]DBpedia 2016-04: http://downloads.dbpedia.org/2016-04/core-i18n/en

the graphDB [10] reasoner with the `rdfsplus-optimised` ruleset. The resulting DBP dataset has 116 417 207 triples, 825 predicates and 9 212 306 entities. We calculate RESCAL factorizations of the two datasets with $r = 100$. We denote by *reconstructed graph* the graph corresponding to the approximate tensor of the original one which is calculated by generating triples using A and R without any rotations. For each dataset we generated five different synthetic graphs.

B. Query performance on generated graphs

With the first experiment, we evaluate whether the synthetic graphs created by SYNTHG lead to similar results as the original graph in a typical use case. To this end, we use all the graphs to benchmark five different triple stores. We use IGUANA [11] to compare the performance of five triple stores: Fuseki[4], Blazegraph[5], HDT [12], GraphDB [10][6] and Virtuoso [13][7]. Each triple store is benchmarked on the original, the reconstructed and five synthetic (i.e. rotated) graphs. The queries used for the benchmarking are based on real-world query logs of SPARQL endpoints and are gathered using FEASIBLE [4] and LSQ [14]. Except property URIs, all bound URI's within the queries are replaced by template variables. Finally, we remove duplicate queries leading to 8 (template) queries for SWDF. For DBpedia, after applying the query preprocessing described above, we gain 24 queries. During the benchmark execution, IGUANA measures the queries per second (QPS) for each single query, i.e., how often the benchmarked triple store can answer the query in a single second. To determine the similarity between the original, the reconstructed and the rotated graphs, we measure the Pearson correlation of these values. The correlation is calculated between graph-pairs selected from three different categories which represent combinations of types of graphs (original (org), reconstructed (rec) or rotated (i.e synthetic) (rot)). We end up with four category-pairs. The number of graph-pairs in each category-pair is (1,5,5,10) for (`org_rec`, `org_rot`, `rec_rot`, `rot_rot`) as there are five synthetic graphs.

Figure 1 and 2 show the correlations for all queries executed on the two datasets. For the SWDF dataset, Figure 1 shows that the QPS values of the single triple stores on the synthetic graphs are strongly correlated with the values achieved on the original and reconstructed graphs. The overall correlation between original and rotated graphs is 91%. Figure 2 shows the 24 queries of DBP benchmark. For the DBP dataset, results of 18 queries show strong correlations between the original and the synthetic graphs (more than 75% correlation). We investigate the possible reasons that caused differences in the runtimes of some of the queries. For the Queries 7, 19 and 21, the result size of their instance queries differs significantly between the synthetic graphs and the original graph.

[4]https://jena.apache.org/documentation/fuseki2/, version 3.11.0
[5]https://www.blazegraph.com/, version 2.1.4
[6]version 8.6.1
[7]version 7.2.5.1

TABLE I
MEAN OF CORRELATIONS OF QPS (PERCENTILES), THE NUMBERS IN HEADER ARE THE NUMBER OF CORRELATIONS

Graph	org_rot (5)	rec_org (1)	rec_rot (5)	rot_rot (10)
SWDF	90.88	93.10	89.37	93.79
DBpedia	64.45	82.83	61.90	70.31

Fig. 1. SWDF: Correlations between QPS of eight queries on five triplestores. org, rec, and rot stands for original, reconstructed and rotated (i.e. synthetic) graphs. Four combinations of types of graphs (org,rec), (rec,rot), (org,rot), and (rot,rot) having counts 1, 5, 5, and 10 respectively.

C. Graph characteristics

To check whether the synthetic graphs attain graph characteristics of the original graphs, we use different graph metrics for a comparison of the original, the reconstructed and the synthetic graphs. These characteristics include the following:

- The number of vertices (#Vert) of the graph.
- The number of vertex triangles (vTr). A vertex triangle comprises three vertices, where each vertex is directly connected to the other two vertices.
- Structuredness (Strctrdnss) is an RDF metric that measures how regularly instances of a class are. Its value is between 0 (low structuredness) and 1 (highly structured) [15], [3].

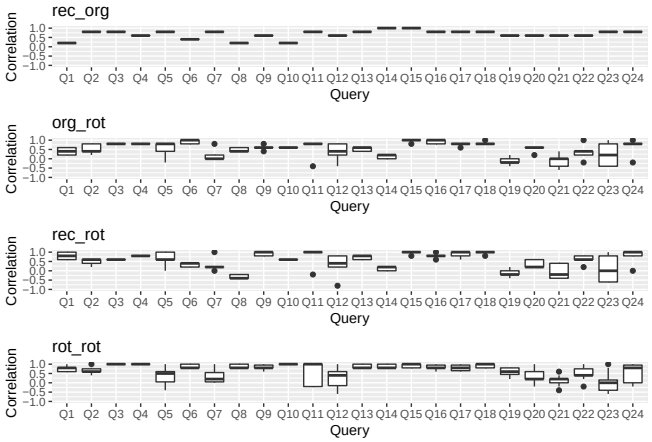

Fig. 2. DBP: Correlations between QPS on five triplestores of 24 queries. For abbreviations as in Figure 1.

The results for these metrics are listed in Table II. The number of vertices of the reconstructed graph of SWDF is about 77% of the original graph. This ratio drops to only 58% for the DBP dataset. This effect could be connected to the fact that the amount of entities with a type definition is about 77% and 59% in SWDF and DBP, respectively. The number of vertex triangles is higher in the reconstructed graphs but slightly decreased in the synthetic graphs of SWDF and similar in the synthetic graphs of DBP. Recently, it was shown in [15], [3] that synthetic graphs tend to have too high values in *Structuredness* metric. This may due to the unrealistic perfection in the synthetic graphs by other methods. This is not the case in our generated graphs as they have lower values than the original graph.

TABLE II
GRAPH METRICS OF ORIGINAL (ORG), RECONSTRUCTED (REC), AND SYNTHETIC (SYNTH) GRAPHS. #VERT IS THE NUMBER OF VERTICES. AVGDG IS THE AVERAGE DEGREE. VTR IS THE NUMBER OF VERTEX TRIANGLES IN MILLIONS. STRCTRDNSS IS THE STRUCTUREDNESS [15], [3]. AVG AND STD : MEAN AND STANDARD DEVIATION RESPECTIVELY.

Graph	#Vert	avgDg	vTr$\times 10^6$	Strctrdnss
SWDF Org	104 601	11.1	2.04	0.5614
SWDF Rec	80 746	14.4	4.11	0.2450
SWDF Synth avg	52 849	21.9	1.38	0.1068
SWDF Synth std	462	0.2	0.12	0.0021
DBP Org	9 212 306	12.6	204.40	0.1230
DBP Rec	5 364 555	21.7	804.61	0.0520
DBP Synth avg	2 143 935	54.3	207.60	0.0085
DBP Synth std	72 723	1.8	29.34	0.0004

V. RELATED WORK

Generating graphs of a given size is a known task. As an example, [16] defines an algorithm to generate a scale-free graph with a given size. However, general graph generation algorithms are not able to mimic the complexity of RDF graphs. This is mainly caused by the fact that these approaches make use of one type of nodes and one type of edges while RDF graphs can comprise different node and edge types. Within the Semantic Web community, the Lehigh University Benchmark (LUBM) generator [17] is a frequently used generator for synthetic RDF graphs. It generates RDF data based on a relational database model of virtual universities. The program takes the number of universities as a parameter and samples the number of professors, employees, students and other parts of the university. The Berlin SPARQL Benchmark (BSBM) offers a data generator for an e-commerce use case [18]. It mainly comprises products, vendors, consumers and their reviews for products. Another RDF graph generator is PoDiGG [2]. Based on a population distribution, it creates a transit network and public transport routes as well as a schedule for trips. All these data generators have two main drawbacks. First, the generated datasets come with a nearly perfect completeness and, hence, are very different to real-world RDF datasets [15]. Second, all the data generators above cover one single domain. Although most of them offer several parameters to configure the generation process, the generation is limited by using a certain, pre-defined ontology. In contrast to that, SYNTHG is aiming at mimicking a given graph—regardless of its domain and including its level of completeness.

VI. CONCLUSION AND FUTURE WORK

We presented and evaluated an approach to generate synthetic graphs that mimic a given real-world graph. SYNTHG is based on RESCAL tensor factorization. In our evaluation, we applied SYNTHG to two real graphs. Our results show that triple stores evaluated on the synthetic graphs achieve a very similar performance as on the original graph. As a future work, we want to extend SYNTHG to include Literals.

ACKNOWLEDGMENT

This work has been supported by the German Federal Ministry of Transport and Digital Infrastructure (BMVI), for Economic Affairs and Energy (BMWi), and for Education and Research (BMBF) within the projects within the projects OPAL (no. 19F2028A), RAKI (no. 01MD19012D) and DAIKIRI (no. 01IS19085B). We thank the HPC at Paderborn University for their support.

REFERENCES

[1] O. Erling, A. Averbuch, J. Larriba-Pey *et al.*, "The ldbc social network benchmark: Interactive workload," in *Proceedings of the 2015 ACM SIGMOD*. ACM, 2015, pp. 619–630.
[2] R. Taelman, P. Colpaert, E. Mannens, and R. Verborgh, "Generating public transport data based on population distributions for rdf benchmarking," *Semantic Web Journal*, Jul. 2018.
[3] M. Saleem, G. Szárnyas, F. Conrads *et al.*, "How representative is a sparql benchmark? an analysis of rdf triplestore benchmarks," in *WWW Conference*, NY, USA, 2019, p. 1623–1633.
[4] M. Saleem, Q. Mehmood, and A.-C. N. Ngomo, "Feasible: A feature-based sparql benchmark generation framework," in *ISWC*. Springer, 2015, pp. 52–69.
[5] M. Nickel and V. Tresp, "Tensor factorization for multi-relational learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2013, pp. 617–621.
[6] P. Hitzler, M. Krotzsch, and S. Rudolph, *Foundations of semantic web technologies*. CRC press, 2009.
[7] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in *ICML*. Omnipress, 2011.
[8] ——, "Factorizing yago: scalable machine learning for linked data," in *Proc. of the 21st international conference on World Wide Web*, 2012.
[9] T. Franz, A. Schultz, S. Sizov, and S. Staab, "Triplerank: Ranking semantic web data by tensor decomposition," in *ISWC*. Springer, 2009.
[10] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov, "Owlim: A family of scalable semantic repositories," *Semantic Web*, vol. 2, no. 1, pp. 33–42, 2011.
[11] F. Conrads, J. Lehmann, M. Saleem, M. Morsey, and A.-C. N. Ngomo, "Iguana: a generic framework for benchmarking the read-write performance of triple stores," in *ISWC*. Springer, 2017, pp. 48–65.
[12] J. D. Fernández, M. A. Martínez-Prieto *et al.*, "Binary rdf representation for publication and exchange (hdt)," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 19, p. 22–41, 2013.
[13] O. Erling and I. Mikhailov, "Rdf support in the virtuoso dbms," in *Networked Knowledge-Networked Media*. Springer, 2009, pp. 7–24.
[14] M. Saleem, M. I. Ali, A. Hogan *et al.*, "Lsq: the linked sparql queries dataset," in *ISWC*. Springer, 2015, pp. 261–269.
[15] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea, "Apples and oranges: a comparison of rdf benchmarks and real rdf datasets," in *Proceedings of the 2011 ACM SIGMOD*, 2011, pp. 145–156.
[16] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.
[17] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *J. Web Sem.*, vol. 3, pp. 158–182, 2005.
[18] C. Bizer and A. Schultz, "The berlin sparql benchmark," *IJSWIS*, vol. 5, pp. 1–24, 2009.