Ranking on Very Large Knowledge Graphs

Abdelmoneim Amer Desouki
desouki@mail.upb.de
Data Science Group, Paderborn University
Paderborn, Germany

Michael Röder
michael.roeder@upb.de
Data Science Group, Paderborn University
Paderborn, Germany

Axel-Cyrille Ngonga Ngomo
axel.ngonga@upb.de
Data Science Group, Paderborn University
Paderborn, Germany

ABSTRACT

Ranking plays a central role in a large number of applications driven by RDF knowledge graphs. Over the last years, many popular RDF knowledge graphs have grown so large that rankings for the facts they contain cannot be computed directly using the currently common 64-bit platforms. In this paper, we tackle two problems: Computing ranks on such large knowledge bases efficiently and incrementally. First, we present D-HARE, a distributed approach for computing ranks on very large knowledge graphs. D-HARE assumes the random surfer model and relies on data partitioning to compute matrix multiplications and transpositions on disk for matrices of arbitrary size. Moreover, the data partitioning underlying D-HARE allows the execution of most of its steps in parallel. As very large knowledge graphs are often updated periodically, we tackle the incremental computation of ranks on large knowledge bases as a second problem. We address this problem by presenting I-HARE, an approximation technique for calculating the overall ranking scores of a knowledge without the need to recalculate the ranking from scratch at each new revision. We evaluate our approaches by calculating ranks on the 3×10^9 and 2.4×10^9 triples from *Wikidata* resp. *LinkedGeoData*. Our evaluation demonstrates that D-HARE is the first holistic approach for computing ranks on very large RDF knowledge graphs. In addition, our incremental approach achieves a root mean squared error of less than 10^{-7} in the best case. Both D-HARE and I-HARE are open-source and are available at: https://github.com/dice-group/incrementalHARE.

CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**; • **Information systems → Resource Description Framework (RDF)**; Page and site ranking.

KEYWORDS

Knowledge Graphs; Ranking RDF; Random Surfer Model

ACM Reference Format:
Abdelmoneim Amer Desouki, Michael Röder, and Axel-Cyrille Ngonga Ngomo. 2019. Ranking on Very Large Knowledge Graphs. In *30th ACM Conference on Hypertext and Social Media (HT '19), September 17–20, 2019, Hof, Germany.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3342220.3343660

1 INTRODUCTION

Over the last years, the number and size of RDF knowledge graphs (short: KGs), which are also called RDF knowledge bases (short: KBs) have grown steadily. For example, the Linked Open Data now contains roughly 10,000 KBs distributed over the Web.[1] These KBs tend to grow continuously over time and several of them reach magnitudes of several billion triples. For example, the *Wikidata* dump grew from 58×10^6 triples in May 2015 to more than 3×10^9 triples in June 2018.[2] The concurrent computation of ranks for triples, properties and resources (called *holistic ranking* [19]) on large, continuously growing knowledge graphs is a task of central importance to devise indexes that can be used by a large number of applications—e.g., question answering, chat bots, semantic search engines, semantic browsers and entity summarization approaches [4, 6, 8, 16, 19]. For example, question answering engines [24] use ranks for resources and properties when preprocessing natural-language questions. They then employ ranks for triples when presenting their results to the end user [4]. While there is consequently an increasing need for ranking approaches to scale to the billions of triples available on the Web, none of the existing approaches for computing ranks on RDF triples scales up to very large knowledge graphs (VLKGs), i.e., knowledge graphs which describe more than $2^{31} - 1$ triples [6]. This is due to three main reasons.

(1) None of the existing approaches has been designed for a distributed environment. Therefore, the existing solutions are bound to be executed on a single machine.
(2) The existing implementations come with additional limitations like the size of the integer address space that can limit the number of triples an implementation is able to handle even on modern 64-bit machines (see, e.g., [4, 6, 8, 19]).
(3) None of the current ranking approaches for RDF provides means for the incremental computation of ranks on VLKGs, i.e., updating existing rankings for a growing graph is not possible.

We address these challenges by extending the only existing holistic ranking approach for RDF knowledge bases, HARE [19].[3] Our contributions can be summarized as follows:

(1) With DISTRIBUTED RANKING FOR KNOWLEDGE GRAPHS (D-HARE), we address the problem of computing rankings on

[1]http://stats.lod2.eu
[2]https://stats.wikimedia.org
[3]We based our works on the version of December 2nd, 2017 at https://github.com/dice-group/HARE as provided to us by the authors of [19].

VLKGs by presenting the first approach able to compute stationary probability distributions on such KGs. D-HARE achieves this goal by (1) representing RDF graphs as bi-partite graphs and (2) partitioning the resulting matrix representation of RDF graphs into chunks upon which the corresponding matrix multiplications and transpositions can be carried out.

(2) As computing ranks on VLKGs is a costly endeavor, we extend D-HARE with Incremental Ranking for Knowledge Graphs (I-HARE) and show as our second contribution that the scores computed by our distributed algorithm can be updated incrementally.

(3) We evaluate our approach on two VLKGs with 2.4×10^9 and 3×10^9 triples and show that both D-HARE and I-HARE scale to billions of triples. In addition, we show that while D-HARE preserves the ranking results of Hare, I-HARE achieves an root mean squared error (RMSE) of under 10^{-7} in the best case.

2 RELATED WORK

Table 1: List of approaches and their features

Approach	Rank resources	Rank triples	Rank properties
PageRank [3]	Yes	No	No
HITS [15]	Yes	No	No
EntityAuthority [22]	Yes	No	No
TripleRank [8]	Yes	No	No
RELIN [4]	No	Yes	No
Dessi et al. [5]	No	No	Yes
Hare [19]	Yes	Yes	Yes

Ranking approaches for RDF can be subdivided into four main categories of approaches, namely ranking for RDF triples, RDF properties, RDF resources and holistic ranking approaches.

Approaches to *ranking triples* have been developed in the area of entity summarization. For example, RELIN [4] uses a graph-based model to detect top-n triples based on the top-m relations of a given resource. TripleRank [8] is a CP-tensor-decomposition-based approach which emulates the HITS algorithm [15]. The decomposition consists of three matrices (U_1, U_2, and U_3). The largest entry of the first row of U_1 corresponds to the best hub for the property group described by the first row of U_3 and the largest entry of first row of U_2 corresponds to the best authority for the same property group.

Ranking properties has also been studied in previous works. RELIN [4] rank properties to the k most relevant triples for a particular resource. The work of Dessi et al. [5] uses a machine learning approach to rank RDF properties based on a number of specifically designed numerical features that measure different aspects of each property. The authors apply existing learning-to-rank algorithms to a number of classified properties, thereby automatically constructing ranking models that reflect a given classification. DBtrends [17] is a framework designed for comparing and evaluating different ranking functions for RDF data. It allows the combination of these rankings by means of an extension of the Spearman's footrule estimation.

The largest body of work related to our approach pertains to *ranking entities*. An extension to the PageRank algorithm for Linked Data is described in [22]. This approach extracts facts from Web documents and combines these facts with an underlying ontology to improve the ranking quality. Swoogle [7], one of the first Semantic Web search engine, relies on OntoRank, an approach based on the rational surfer model at the document level. The ranking function of the Sindice [23] search engine relies on a fast metadata-using algorithm. However, Sindice does not make use of any RDF-based metadata for ranking and only covers information such as host ranks. Blanco et al. [2] proposes an adaptation of the BM25F ranking function to the RDF data model that incorporates field weights, document priors and a separate field for the subject URIs. The authors also go on to propose index structures in order to achieve efficient retrieval and ranking of results. Mirizzi et al. [18] propose a novel, hybrid ranking function for DBpedia using external sources such as search engines and bookmarking sites. ReConRank [14] is another efficient algorithm based on PageRank. This algorithm is based on semantic subgraphs whose size influences efficiency and precision of results. Other approaches such as Gupta et al. [11] or xhRank [13] also generate result list rankings. Graves et al. [9] present an approach to rank RDF nodes in result sets according to their centrality. The authors evaluate their approach by showing the top-10 concepts of the CIA-Factbook only. Most recently, PageRankRDF [6] was used to calculate PageRank scores for a VLKG (i.e., the *Wikidata* dump from September 2017). However, PageRankRDF calculates scores exclusively for resources. While it can deal with billions of triples, it builds its index on resources and cannot deal with more than 2.15 billion resources. An overview of other approaches can be found in in [20, 25].

Hare [19] is the first holistic ranking approach for knowledge bases. It aims at ranking triples, properties and resources concurrently. To achieve this goal, it relies on a bi-partite representation of RDF graphs [12]. While it is shown to scale well for graphs with millions of triples, it fails to compute ranks for graphs with 2^{31} triples or more.

Overall, none of the approaches above is able to compute ranks for triples on VLKGs even in an incremental fashion. We address this limitation by presenting algorithms able to compute ranks on VLKGs both in a non-incremental and an incremental fashion.

3 PRELIMINARIES

Throughout this paper, we use the symbols presented in Table 2.

3.1 Knowledge Graphs

Following [19], an RDF *knowledge graph*[4] K can be modeled as a set of triples $(s, p, o) \in (R \cup B) \times P \times (R \cup B \cup L)$ where

- R is the set of all RDF resources, which stand for things of relevance in the domain to model. For example, these could be publications, authors and conferences in a knowledge graph on publications.

[4]See https://www.w3.org/RDF/ for a complete specification.

Table 2: List of symbols

Symbol	Description
B	Set of all blank nodes
$\mathbf{D}$	Transition matrix for a two-node graph in I-HARE
$\mathbf{F}$	Probabilistic transition matrix from entities to triples of size $\alpha \times \beta$
L	Set of all literals
P	Set of all properties
$\mathbf{P}$	Probabilistic transition matrix from entities to entities of size $\alpha \times \alpha$
R	Set of all resources
$\mathbf{S}(\mathcal{N})$	Stationary distribution for entities
$\mathbf{S}(\mathcal{T})$	Stationary distribution for triples
$\mathbf{W}$	Probabilistic transition matrix from triples to entities of size $\beta \times \alpha$
α	Number of entities
β	Number of triples
γ	Damping factor
η	Ratio between ranking scores in I-HARE
λ	a scalar giving the chunk size of $\mathbf{F}$ and $\mathbf{W}$
μ	a scalar giving the chunk size of $\mathbf{P}$
ζ	a scalar giving the chunk size of $\mathbf{P}^T$

- B is the set of all RDF blank nodes, i.e., nodes which describe existential quantifications of entities that do not need a unique global identifier.
- $P \subseteq R$ is the set of all RDF predicates and stands for the set of binary relations which can exist between resources or literals. For example, the predicate `authorOf` can be used to link a person with a publication (s)he (co-)wrote.
- L is the set of all literals, i.e., of all data values used in a given knowledge graph. For example, the predicate for the year of publication links a publication to a literal value for the year in which the publication was accepted.

Each RDF triple stands for a fact described in the knowledge graph. For example, the triple (`:LarryPage`, `:authorOf`, `:PageRank`) is to be understood as stating the fact that Larry Page co-authored the paper on the PageRank algorithm. Like [19], we use *entities* to refer to all elements of $R \cup P \cup B \cup L$.

3.2 Holistic Ranking

The main idea behind holistic ranking on RDF is to devise a ranking algorithm that can compute ranks for both triples (e.g., for use in entity summarization and search engines) and entities (e.g., for question answering and search engines). The HARE algorithm [19] achieves this goal by representing RDF knowledge graphs as bi-partite graphs [12] and using this representation to compute ranking efficiently. The authors show that Hayes et al.'s bi-partite representation leads to the stationary distribution over entities (dubbed $\mathbf{S}(\mathcal{N})$) being dependent on the stationary distribution over triples (dubbed $\mathbf{S}(\mathcal{T})$) by virtue of

$$\mathbf{S}(\mathcal{T}) = \mathbf{F}^T \cdot \mathbf{S}(\mathcal{N}). \tag{1}$$

Hence, solving the holistic ranking problem can be implemented in two steps. First, compute $\mathbf{S}(\mathcal{N})$, then compute $\mathbf{F}^\top \cdot \mathbf{S}(\mathcal{N})$ to get $\mathbf{S}(\mathcal{T})$.

HARE hence first computes the stationary distribution $\mathbf{S}(\mathcal{N})$ of the probabilistic transition matrix $\mathbf{P} = \mathbf{F} \cdot \mathbf{W}$, where $\mathbf{F}_{(\alpha,\beta)}$ is the matrix containing the transition probabilities from entities to triples and $\mathbf{W}_{(\beta,\alpha)}$ is the transition matrix from triples to entities. The entries of $\mathbf{F}$ and $\mathbf{W}$ are computed as follows: Let e_i be an entity and $t_j = (s_j, p_j, o_j)$ be a triple in K. We write $e_i \in t_j$ to signify $e_i \in \{s_j, p_j, o_j\}$. The entries f_{ij} of $\mathbf{F}$ are given by

$$f_{ij} = \begin{cases} \frac{1}{|\{t_k | e_i \in t_k, t \in K\}|} & \text{, iff } e_i \in t_j \\ 0 & \text{, else.} \end{cases} \tag{2}$$

$\mathbf{W}_{(\beta,\alpha)}$ is the transition matrix from triples to entities. Given that the probability of transitioning from a triple to one of its elements is exactly $\frac{1}{3}$, $w_{ji} = \frac{1}{3}$ iff $e_i \in t_j$. Else $w_{ji} = 0$. $\mathbf{S}(\mathcal{N})$ can now be computed by applying the power method to

$$\mathbf{S}(\mathcal{N}) = \gamma \mathbf{P}^T \cdot \mathbf{S}(\mathcal{N}) + \frac{(1-\gamma)\mathbf{I}}{|\mathbf{S}(\mathcal{N})|}, \tag{3}$$

where $\gamma \in [0, 1]$ is the damping factor. While HARE is shown to be faster than PageRank, its current implementation cannot scale to VLKGs. In the following, we show how D-HARE achieves the computation of Equations 3 and 1 in a distributed manner.

4 DISTRIBUTING RANK COMPUTATION

Our distributed rank computation calculates the ranks in four steps: (1) Calculate the transition matrix $\mathbf{P}$, (2) calculate the transposed matrix $\mathbf{P}^T$, (3) calculate the stationary distribution over entities $\mathbf{S}(\mathcal{N})$ and (4) the stationary distribution over triples $\mathbf{S}(\mathcal{T})$.

4.1 Matrix Chunks

To distribute the computation, our approach will split the matrices into chunks. We will use the notation $\mathbf{F}[a : b, c : d]$ to denote the part of matrix $\mathbf{F}$ from row a to row b and from column c to column d. If we want to select all columns or all rows we use : without starting nor end. We assume that the first index is 1.[5] λ is the chunk size of $\mathbf{F}$ (vertical chunk) and $\mathbf{W}$ (horizontal chunk) while μ is the chunk size of $\mathbf{P}$ (horizontal chunk). The chunk size of $\mathbf{P}^T$ is denoted ζ. As the limit identifying those sizes is the number of non-zeros in one chunk, they can vary in values. This give more flexibility to avoid redoing a preceding step when a latter one needs a smaller chunk size. We use $\mathbf{F}^{(i)}$ to denote the i-th chunk of $\mathbf{F}$. The following is used to identify chunks of the main matrices:

$$\mathbf{F}^{(i)} = \mathbf{F}[((i-1)\lambda + 1) \ : \ (i\lambda), :] \tag{4}$$

$$\mathbf{W}^{(i)} = \mathbf{W}[:, ((i-1)\lambda + 1) \ : \ (i\lambda)] \tag{5}$$

$$\mathbf{P}^{(j)} = \mathbf{P}[((j-1)\mu + 1) \ : \ (j\mu), :] \tag{6}$$

$$\mathbf{P}^{T(l)} = \mathbf{P}^T[((l-1)\zeta + 1) \ : \ (l\zeta), :] \tag{7}$$

4.2 Calculating the transition matrix

Computing the matrices $\mathbf{F}$ and $\mathbf{W}$ in memory is not feasible on currently common computer architectures because the number of rows of $\mathbf{F}$ (which is equal to the number of columns of $\mathbf{W}$) can easily be larger than the space which can be addressed with an integer datatype in many programming languages. For example, the 3.03 billion triples found in the *Wikidata* dump of June 13th 2018 exceed this limit. D-HARE circumvents the problem of addressing both $\mathbf{F}$ and $\mathbf{W}$ (which are needed to calculate the transition matrix $\mathbf{P} = \mathbf{F}\cdot\mathbf{W}$) by computing $\mathbf{P}$ (with dimensions $\alpha \times \alpha$) directly based on the assumption that $\alpha < \beta$ (e.g., $\alpha \approx 594 \times 10^6$ and $\beta \approx 3 \times 10^9$ for *Wikidata*). For the sake of simplicity, we explain our approach based on the assumption that α is addressable with 32 bits. Note that our approach can be easily extended to larger values of α.

$$n = \frac{\beta}{\lambda} \tag{8}$$

$$\mathbf{Q}_i = \mathbf{F}^{(i)} \cdot \mathbf{W}^{(i)} \tag{9}$$

$$m = \frac{\alpha}{\mu} \tag{10}$$

$$\mathbf{P}^{(j)} = \sum_{i=1}^{n} \mathbf{Q}_i[((j-1)\mu + 1) \ : \ (j\mu), :] \tag{11}$$

let λ be the chunk size, n be the number of chunks of $\mathbf{F}$ and $\mathbf{W}$. $\mathbf{F}$ and $\mathbf{W}$ are computed in the fly and $\mathbf{Q}_i$ is stored to disk. Dimension of $\mathbf{Q}_i$ is α by α. To avoid hitting the limit of maximum number of non-zeros in sparse matrix (2^{31}) $\mathbf{P}$ is stored as a set of horizontal chunks with μ chunk size. To compute one horizontal chunk of $\mathbf{P}$ we sum the corresponding indexes of all $\mathbf{Q}_i$, see Figure 1. Algorithm 1 implements this approach. The algorithm begins by calculating the number of chunks (`n`) in $\mathbf{F}$. In each iteration, we create two sparse matrices $\mathbf{F}^{(i)}$ and $\mathbf{W}^{(i)}$ to hold one horizontal chunk from $\mathbf{F}$ and $\mathbf{W}$ respectively. The chunks are then filled by calculating the corresponding subset of λ columns of $\mathbf{F}$. The columns are calculated on the fly from four files. The first file contains the number of occurrences of each entity (`E_cnt`), the others contain the index of the entities forming a triple (i.e subject, predicate and object of K). Then, $\mathbf{W}^{(i)}$ is initialized with zeros. For each row, the subject, predicate and object of the row's triple are set to $\frac{1}{3}$. For each pair of corresponding chunks $\mathbf{F}^{(i)}$ and $\mathbf{W}^{(i)}$, $\mathbf{Q}_i = \mathbf{F}^{(i)} \cdot \mathbf{W}^{(i)}$ is calculated and saved to disk. To calculate $\mathbf{P}$, all $\mathbf{Q}_i$ are indexed and added. Chunks of $\mathbf{P}$ will be saved to disk one by one as in Algorithm 1.

[5]We assume that β is divisible by λ however in our implementation we take care of that by using ceiling function and make the last chunk smaller if necessary.

Algorithm 1: Calculating $\mathbf{P} = \mathbf{F} \cdot \mathbf{W}$

```
Data: E_cnt, s, p and o files, chunk sizes $\lambda$ and $\mu$
Result: set of files containing $\mathbf{P} = \mathbf{F} \cdot \mathbf{W}$
n = $\beta/\lambda$;
for ($i = 1; i \leq n; i = i + 1$) do
    $\mathbf{F}^{(i)} = zeros(\alpha \times \lambda)$;
    $\mathbf{W}^{(i)} = zeros(\lambda \times \alpha)$;
    $S_i = readchunkfromsubjectfile$;
    $\mathbf{W}^{(i)}[1 : \lambda, S_i indexes]$=1/3;
    $\mathbf{F}^{(i)}[S_i, 1 : \lambda] = 1/E\_cnt[S_i]$;
    do the same for predicate and object files;
    $\mathbf{Q}_i = \mathbf{F}^{(i)} \cdot \mathbf{W}^{(i)}$;
    save($\mathbf{Q}_i$ to file);
end
//process all $\mathbf{Q}_i$ to get P as horizontal chunks
m = $\alpha/\mu$;
for ($j \leq m$) do
    $\mathbf{P}^{(j)} = \mathbf{Q}_1[((j-1) * \mu + 1) : (j * \mu), :]$;
    for ($i = 2; i \leq n; i = i + 1$) do
        $\mathbf{P}^{(j)} = \mathbf{P}^{(j)} + \mathbf{Q}_i[((j-1) * \mu + 1) : (j * \mu), :]$
    end
    save($\mathbf{P}^{(j)}$ to file);
end
```

4.3 Calculating the transpose of the transition matrix

To calculate the transpose of $\mathbf{P}$, we use the horizontal chunks from the previous step. All the chunks are processed to calculate one horizontal chunk of $\mathbf{P}^T$ as shown in Algorithm 2 and Figure 2. To calculate a horizontal chunk of $\mathbf{P}^T$ (l), the horizontal chunks of $\mathbf{P}$ are read one by one, the rectangle that will be located in the transposed chunk is copied and transposed. This transposed rectangle is stored into l by adding the shift `col_st` to the column indexes. Note that this step is implemented in parallel by assigning each $\mathbf{P}^T$ chunk to one processor.

4.4 Calculating the stationary distribution

HARE's calculation of the the stationary distribution $\mathbf{S}(\mathcal{N})$ is slightly modified to allow using $\mathbf{P}^T$ as a set of chunks. As shown in Algorithm 3, in every iteration of the outer loop, a chunk of $\mathbf{S}(\mathcal{N})$ is updated in the inner loop using the corresponding chunk of $\mathbf{P}^T$ and the current state of $\mathbf{S}(\mathcal{N})$. This step can be carried out in parallel simply by assigning single chunks of $\mathbf{S}(\mathcal{N})$ to workers.

The computation of $\mathbf{S}(\mathcal{T})$, which is the second step of the HARE computation, uses chunks to calculate $\mathbf{F}^T$. The necessary $\mathbf{F}^{(i)}$ chunks are calculated on the fly as described in Section 4.2 and multiplied by $\mathbf{S}(\mathcal{N})$ to get corresponding chunks of $\mathbf{S}(\mathcal{T})$.

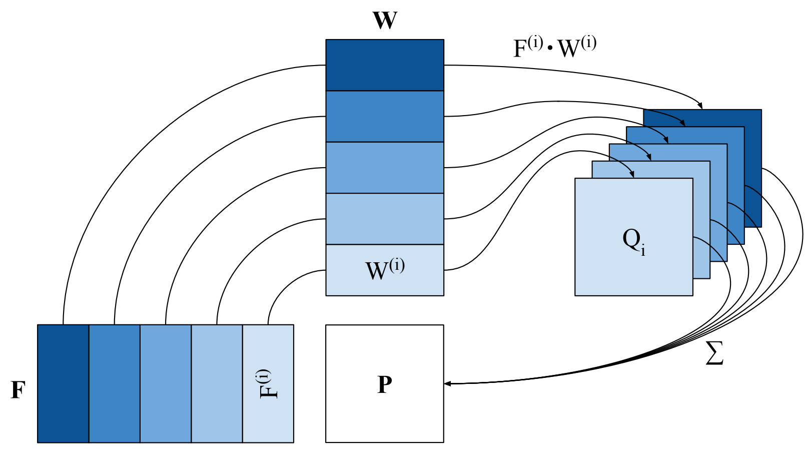

Figure 1: Calculating $\mathbf{P} = \mathbf{F} \cdot \mathbf{W}$ using chunks. Multiply the corresponding chunks $\mathbf{F}^{(i)}$ from F and $\mathbf{W}^{(i)}$ from W to calculate $\mathbf{Q}_i$. Sum all of them up to get P.

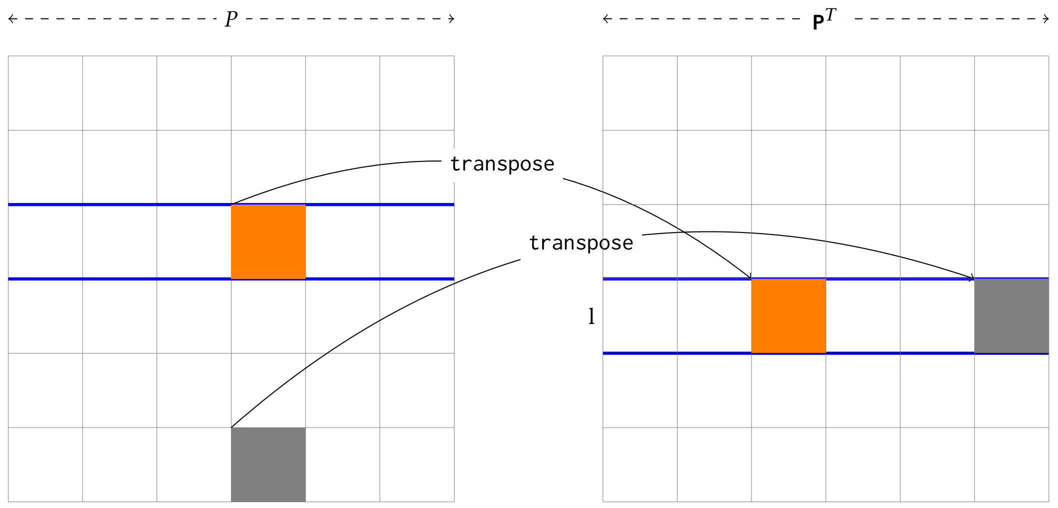

Figure 2: Iterate over chunks of P, transpose a rectangle then put it to corresponding l chunk. The blue borders are of the current chunks loaded in memory.

5 INCREMENTAL RANKING I-HARE

5.1 Approach

With D-HARE, we can scale up to transition matrices of arbitrary size. However, our results show that the computation of rankings is still costly (see Section 6). Interestingly, a number of VLKBs (e.g., *Wikidata* and LinkedGeoData) achieve their size incrementally, i.e., by being updated periodically. We hence build upon D-HARE with I-HARE, which can compute an incremental approximation of the values returned by D-HARE.

The algorithm is based on the monotonicity of the random surfer model. We assume that we are given D-HARE scores for the initial version of a given knowledge graph, which we dub G_{i-1}. The i-th version G_i of the dataset is regarded as being equal to $G_{i-1} + \Delta_{i-1}$, where Δ_{i-1} stands for the triples which were added to G_{i-1} to generate G_i.[6] Our approach aims to approximate the stationary distribution of G_i by computing a weighted average over the stationary distribution of G_{i-1} and Δ_{i-1}. The intuition behind our approach is that the graph G_{i-1} and Δ_{i-1} can be regarded as a two-node graph with nodes G_{i-1} and Δ_{i-1} and a transition matrix which reflects the probability of "jumping" from one entity of G_{i-1} to an entity of Δ_{i-1} and vice-versa.

I-HARE hence works as follows: First, it computes ranking scores of the subgraph Δ_{i-1}. Note that Δ_{i-1} is commonly significantly smaller than G_{i-1}. We now need to build the transition matrix $\mathbf{D}_{i-1}$ between G_{i-1} and Δ_{i-1}. To this end, we find the set of entities common to the two graphs and set

[6] In its current version, I-HARE only takes additions.

```
Algorithm 2: Calculating transpose of P
Data: P as set of files (one chunk per file),
  m: number of chunks in P,
  $\zeta$ :chunk size of $\mathbf{P}^T$
Result: set of files containing horizontal chunks of $\mathbf{P}^T$
k = $\alpha/\zeta$;
for (l = 1; l ≤ k; l = l + 1) do
    col_st = 1;
    //Horizontal chunk of $\mathbf{P}^T$
    $\mathbf{P}^{T(l)}$ = zeros ($\zeta \times \alpha$);
    for ( i = 1;i ≤ m;i = i + 1) do
        $\mathbf{P}^{(i)}$ = read chunk from file;
        index_st= 1 + $(l-1) * \zeta$;
        index_end = $l * \zeta$;
        tmp = transpose($\mathbf{P}^{(i)}$);
        tmp = tmp[index_st:index_end,];
        col_end = col_st + $\mu$-1;
        put sparse matrix in form of 3 fields: i (row index),j
          (column index), and x (value)
        newj=column_index(tmp)+col_st;
        tmp = make sparseMatrix(i=row_index(tmp), j=newj,
          x=values(tmp),
        no_of_rows= $\zeta$,no_of_columns= $\alpha$);
        $\mathbf{P}^{T(l)} = \mathbf{P}^{T(l)}$ + tmp;
        col_st = col_end + 1;
    end
    Save $\mathbf{P}^{T(l)}$ to file;
end
```

```
Algorithm 3: Calculating $\mathbf{S}(\mathcal{N})$ with $\mathbf{P}^T$ as set of chunks
Data: $\epsilon$, $\gamma$:damping factor, maxIterations, k: number of
      chunks, $\mathbf{P}^T$ as set of chunks
Result: $\mathbf{S}(\mathcal{N})$
error = 1;
ni = 0;
I = ones(n);
$\mathbf{S}(\mathcal{N})$ = I/n;
while ($error > \epsilon$ and $ni < maxIterations$) do
    ni = ni + 1;
    $\mathbf{S}(\mathcal{N})_{previous} = \mathbf{S}(\mathcal{N})$;
    for ($t\_ch = 1$ ; $t\_ch < k$ ; $t\_ch = t\_ch + 1$) do
        chunk_range=start index : end index of chunk t_ch;
        $\mathbf{S}(\mathcal{N})$[chunk_range] =
          $\gamma * (\mathbf{P}^{T^{(i)}} * \mathbf{S}(\mathcal{N})_{previous}) + (1-\gamma)$*I[chunk_range]/n
    end
    error = norm($\mathbf{S}(\mathcal{N})_{previous} - \mathbf{S}(\mathcal{N})$);
end
$\mathbf{S}(\mathcal{N}) = \mathbf{S}(\mathcal{N})_{previous}$;
```

$$\mathbf{D}_{i-1} = \begin{pmatrix} \frac{3||\Delta_{i-1}||}{3||\Delta_{i-1}||+G\_ext} & \frac{G\_ext}{3||\Delta_{i-1}||+G\_ext} \\ \frac{Delta\_ext}{3||G_{i-1}||+Delta\_ext} & \frac{3||G_{i-1}||}{3||G_{i-1}||+Delta\_ext} \end{pmatrix} \qquad (12)$$

and the number of external links as shown in Figure 3 and Algorithm 4.

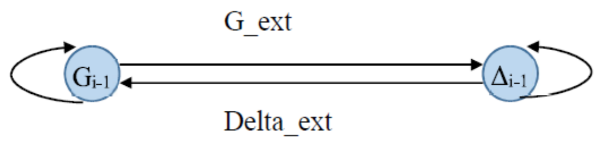

Figure 3: A two-node graph representing the transition between main graph and a revision

`G_ext` and `Delta_ext` are the numbers of external links from the main graph to Δ_{i-1} and vice versa, respectively.

Then apply power method to get the stationary probability for the two nodes using transition matrix **D**—we call them η and $1-\eta$. Then calculate approximate $\mathbf{S}(\mathcal{N})$ ranks for resources, by using the weighted sum for common resources as in Algorithm 4.

```
Algorithm 4: Calculating I-HARE
Data: Graph: The graph $G_{i-1}$
  Delta: Subgraph which was added to $G_{i-1}$ to generate $G_i$
  GScores: pre-calculated scores of $G_{i-1}$ as table with entities
  and scores.
Result: approximate ranking scores IScores of $G_i$
DeltaScores = calculate ranking of Delta subgraph as table
  with entities and scores.;
ce = intersect(Delta.Entities, Graph.Entities);
Delta_ext = count(Delta.triples.subject in ce) +
  count(Delta.triples.predicate in ce) +
  count(Delta.triples.object in ce);
G_ext = count(Graph.triples.subject in ce) +
  count(Graph.triples.predicate in ce) +
  count(Graph.triples.object in ce);
// transition matrix between Graph and Delta
D= matrix(2,2);
D[1,1] =
  $3 * Delta.countTriples/(3 * Graph.countTriples + G\_ext)$;
D[1,2] = $G\_ext/(3 * Delta.countTriples + G\_ext)$;
D[2,1] = $Delta\_ext/(3 * Graph.countTriples + Delta\_ext)$;
D[2,2] = $3 * Graph.countTriples/(3 * Graph.countTriples +$
  $Delta\_ext)$;
$\eta = PowerMethodLoop(\mathbf{D})[1]$;
scoresTable=full outer join of GScores and DeltaScores on
  the entity column;
IScores = $\eta * scoresTable["DeltaScores"] + (1-\eta) *$
  $scoresTable["GScores"]$
```

5.2 Optimal value of η

The value of η that gives the minimum rMSE between approximate scores returned using I-HARE and the scores calculated from HARE is given by:

$$\eta = \frac{1}{\alpha} \sum \frac{M - M_2}{M_1 - M_2} \qquad (13)$$

where α is the total number of entities and M is the ranking scores for the graph that combines the two subgraphs while M_1 and M_2 are the scores for the first and the second subgraph respectively. The I-HARE scores will be given as $\eta M_1 + (1 - \eta)M_2$. The entries where $M_1 - M_2$ are zeros can be excluded.

6 EVALUATING D-HARE

6.1 Experimental Setup

The goal of our first series of experiments was twofold: First, we aimed to gather insights into the overhead introduced by partitioning. To this end, we applied D-HARE to a set of synthetic graphs and compare its runtime with the original implementation of Hare [19]. We used the Lehigh University Benchmark (LUBM) generator [10] for the generation of the data. The number behind the dataset in Table 3 indicates the number of universities created. The chunk size used for $\mathbf{F}$ and $\mathbf{W}$ is 2 million and for $\mathbf{P}$ and $\mathbf{P}^T$ is 500000. Note that all graphs generated in the first part of our experiments were not VLKGs.

To measure the performance of D-HARE on VLKGs, we applied D-HARE to two very large datasets. Firstly, we applied it to the *Wikidata* dump from February 2nd 2018. The dataset has a size of 322GB and contains about 2.6 billion triples. The chunk size used for calculating $\mathbf{F} \cdot \mathbf{W}$ was 25M triples resulting in 105 chunks. We used $\epsilon = 10^{-3}$ and a damping factor $\gamma = 0.85$ as chosen in [1] and [19]. Secondly, we applied D-HARE to the LinkedGeoData [21] datasets of four countries: Austria, Switzerland, Sweden and Greece.[7] We got about 2.44 Billion triples. the average outgoing degree of the LinkedGeoData graph is 2.69 in contrast to 5.02 of *Wikidata*. The chunk size used for calculating $\mathbf{F} \cdot \mathbf{W}$ was 25M triples and 20M rows for $\mathbf{P}$ and $\mathbf{P}^T$ respectively.

All parallel experiments were executed on 16 CPUs of a server cluster with 9.920 processor cores 2.6 GHz Intel Xeon "Sandy Bridge".

6.2 Runtimes

6.2.1 Overhead. The results of this experiment are shown in Table 3. As expected, D-HARE requires more time than Hare. On roughly 138 million triples (LUBM1000), D-HARE requires roughly 3 times the runtime of Hare. The runtime overhead is mostly due to number of chunks in $\mathbf{P}^T$ growing fast and leading a time overhead due to disk accesses. In particular, the longer write access time means that the overhead of writing to disk influence the performance of D-HARE significantly. The overheads are mainly reflected in the loop calculating $\mathbf{S}(\mathcal{N})$ as given in Algorithm 3 at line 10.

6.2.2 Results on VLKGs. In this portion of our experiments, we were interested in the results of the two VLKGs LinkedGeoData and *Wikidata*. Note that these graphs simply could not be processed by Hare by virtue of the large number of triples they contain. Hence, while Hare was not able to run on these datasets, we achieved a runtime of 32.3 hours on LinkedGeoData and 27.9 hours on *Wikidata*. This result confirms that our approach addresses the problem of running of VLKGs. Remember that D-HARE returns exactly the same results as Hare. An overview of the top-10 resources from LinkedGeoData is given in Table 4. These results are similar with those presented in [19] for smaller datasets.

[7]We downloaded the open street maps files on September 10th 2018 from http://download.geofabrik.de

7 EVALUATING I-HARE

7.1 Experimental Setup

To evaluate the performance of I-HARE we used two *Wikidata* dumps taken on 04-Jan-2018 and 13-Jun-2018 respectively. We compared the two dumps and extracted two files that contain the added triples (680M triples, named Delta) and the deleted (60M triples) to the first dump. We appended the deleted triples to the June dump (called newJun). We calculated ranks scores for three graphs: Delta, Jan and newJun using D-HARE. Table 5 shows the runtimes of the different steps for calculating $\mathbf{S}(\mathcal{N})$ and $\mathbf{S}(\mathcal{T})$ for the three graphs. Then, we calculated the I-HARE ranks using the ranks from Delta and Jan graphs. Finally, we compared these scores with Hare scores calculated on newJun graph directly.

7.2 Runtimes

The runtime was about 11.6 hour which means that the total time needed for calculating the exact ranks using D-HARE is more than four times that of calculating the approximate ones using I-HARE.

Table 7 shows the ranking scores and counts of top 10 entities calculated using Hare. Column I-HARE shows the values calculated using I-HARE. The ranks of Hare scores and the I-HARE scores are the same for the top 10 entities.

7.3 The effect of damping factor

We evaluated different values for the damping factor γ which affected the convergence speed (i.e., the number of iterations required) of our implementation (see Table 6). We calculated the ranks with R function `rank` which gives same value to ties. We used minimal values for ties. We also calculated the median (med) and average (avg) of the obtained scores. Our results show clearly that an increase of the damping factor leads to a smaller median of the rank difference. While the number might appear high, the large differences are simply due to ties in the data. As shown by the rMSE evaluation, the score assigned to triples and resources by I-HARE by roughly 10^{-7} from that computed by D-HARE. This suggests that I-HARE computes reliable approximation of the numbers computed by D-HARE.

8 CONCLUSION AND FUTURE WORK

We presented two methods for ranking very large knowledge graphs (with mora than 2^{31} triples). D-HARE uses partitioning to be able to distribute the computation for large matrices. I-HARE provides an incremental extension of D-HARE to rank updates of knowledge graphs. We tested our approaches using *Wikidata* and LinkedGeoData. Our results suggest that while there is an overhead to not keeping all data in memory like Hare, we scale up to datasets which simply do not fit in the main memory of single machines. Moreover, using I-HARE does not distort the ranking computed significantly. Hence, our results show that D-HARE and I-HARE can be used in practical large-scale applications where the knowledge graphs to process are very large. Interestingly, the value of η influenced our results drastically. We will hence revisit the I-HARE approximation

Table 3: Runtimes in seconds of D-HARE on LUBM graphs

Graph	#Triples	#Entities	Hare		D-HARE	
			Preprocessing	$\mathbf{S}(\mathcal{N})$	Preprocessing	$\mathbf{S}(\mathcal{N})$
LUBM20	2 781 322	663 661	42	1	44	4
LUBM50	6 888 642	1 639 709	92	2	102	6
LUBM100	13 875 956	3 301 732	194	4	234	12
LUBM200	27 635 634	6 574 874	403	8	565	32
LUBM500	69 079 764	16 429 334	1060	20	1955	140
LUBM1000	138 278 374	32 885 165	2281	43	5679	545

Table 4: Top-10 D-HARE scores on LinkedGeoData

Rank	Entity	Count ($\times 10^8$)	Score
1	`rdf:type`	6.11	0.055
2	`lgd:ontology/gadmSameAs`	1.64	0.031
3	`wgs84:long`	1.64	0.023
4	`geovocab:geometry`	1.80	0.022
5	`wgs84:lat`	1.64	0.021
6	`geovocab:Geometry`	1.80	0.020
7	`dcterms:modified`	1.80	0.016
8	`geovocab:Feature`	1.80	0.015
9	`"18"^^xsd:int`	1.80	0.014
10	`lgd:triplify/user0`	1.80	0.014

Table 5: Runtimes of the different I-HARE steps in hours.

Graph	$\beta(\times 10^9)$	Calc. $\mathbf{P}$	$\mathbf{P}^T$	$\mathbf{S}(\mathcal{N})$	$\mathbf{S}(\mathcal{T})$	Total
Jan	2.42	11.6	7.8	2.2	6.4	27.9
Delta	0.68	2.9	1.8	0.7	1.6	7.0
newJun	3.10	18.8	14.0	5.3	8.6	46.6

Table 6: The effect of damping factor on I-HARE

Damping factor	Median Rank Diff.($\times 10^6$)	Average Rank Diff.($\times 10^6$)	rMSE ($\times 10^{-8}$)	Number of iterations
0.85	36	50	12.2	20
0.90	33	40	12.7	26
0.95	12	23	13.8	38
0.99	9	18	8.8	56

and ensure that it produces results that are more robust will also dealing with the removal of triples. In addition, we will consider even larger graphs and the parallelization of all our steps in future evaluations of our approaches.

ACKNOWLEDGMENTS

This work has been supported by the BMVI (Bundesministerium für Verkehr und digitale Infrastruktur) projects LIMBO (GA no. 19F2029C) and OPAL (GA no. 19F2028A), the Eurostars Project SAGE (GA no. E!10882) as well as the H2020 project SLIPO (GA no.731581).

REFERENCES

[1] Abdelghani Bellaachia and Mohammed Al-Dhelaan. 2013. Random walks in hypergraph. In *Proceedings of the 2013 international conference on applied mathematics and computational method*. 187–194.

[2] Roi Blanco, Peter Mika, and Sebastiano Vigna. 2011. Effective and efficient entity search in RDF data. In *ISWC*. Springer, 83–97.

[3] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Computer Networks and ISDN Systems*. Elsevier Science Publishers B. V., 107–117.

[4] Gong Cheng, Thanh Tran, and Yuzhong Qu. 2011. RELIN: relatedness and informativeness-based centrality for entity summarization. In *International Semantic Web Conference*. Springer, 114–129.

[5] Andrea Dessi and Maurizio Atzori. 2016. A machine-learning approach to ranking RDF properties. *Future Generation Computer Systems* 54 (2016), 366–377.

[6] Dennis Diefenbach and Andreas Thalhammer. 2018. Pagerank and generic entity summarization for rdf knowledge bases. In *European Semantic Web Conference*. Springer, 145–160.

[7] Li Ding, Rong Pan, Timothy W. Finin, Anupam Joshi, Yun Peng, and Pranam Kolari. 2005. Finding and Ranking Knowledge on the Semantic Web. In *ISWC*, Vol. 3729. Springer, 156–170.

[8] Thomas Franz, Antje Schultz, Sergej Sizov, and Steffen Staab. 2009. Triplerank: Ranking semantic web data by tensor decomposition. In *International semantic web conference*. Springer, 213–228.

[9] Alvaro Graves, Sibel Adali, and Jim Hendler. 2008. A Method to Rank Nodes in an RDF Graph. In *International Semantic Web Conference (Posters & Demos) (CEUR Workshop Proceedings)*, Vol. 401. CEUR-WS.org.

[10] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.* 3, 2-3 (2005), 158–182.

[11] Parul Gupta and AK Sharma. 2012. Ontology driven Pre and Post Ranking based Information Retrieval in Web Search Engines. *International Journal on Computer Science and Engineering* 4, 6 (2012), 1241.

[12] Jonathan Hayes and Claudio Gutierrez. 2004. Bipartite Graphs as Intermediate Model for RDF. In *In Proc. of the 3th Int. Semantic Web Conference (ISWC), number 3298 in LNCS*. Springer-Verlag, 47–61.

[13] Xin He and Mark Baker. 2010. xhRank: Ranking Entities on the Semantic Web. In *ISWC Posters&Demos (CEUR Workshop Proceedings)*. CEUR-WS.org.

[14] Aidan Hogan, Andreas Harth, and Stefan Decker. 2006. ReConRank: A Scalable Ranking Method for Semantic Web Data with Context. In *In 2nd Workshop on Scalable Semantic Web Knowledge Base Systems*.

[15] Jon M. Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *J. ACM* 46, 5 (Sept. 1999), 604–632. https://doi.org/10.1145/324133.324140

[16] Vanessa Lopez, Christina Unger, Philipp Cimiano, and Enrico Motta. 2013. Evaluating question answering over linked data. *Web Semantics: Science, Services and Agents on the World Wide Web* 21 (2013), 3–13.

[17] Edgard Marx, Amrapali Zaveri, Mofeed Mohammed, Sandro Rautenberg, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, and Gong Cheng. 2016. DBtrends: Publishing and Benchmarking RDF Ranking Functions.. In *SumPre@ ESWC*.

[18] Roberto Mirizzi, Azzurra Ragone, Tommaso Di Noia, and Eugenio Di Sciascio. 2010. Ranking the Linked Data: The Case of DBpedia. In *ICWE (Lecture Notes in Computer Science)*, Vol. 6189. Springer, 337–354.

[19] Axel-Cyrille Ngonga Ngomo, Michael Hoffmann, Ricardo Usbeck, and Kunal Jha. 2017. Holistic and Scalable ranking of RDF data. In *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, 746–755.

[20] Antonio J. Roa-Valverde and Miguel-Angel Sicilia. 2014. A Survey of Approaches for Ranking on the Web of Data. *Inf. Retr.* 17, 4 (Aug. 2014), 295–325. https://doi.org/10.1007/s10791-014-9240-0

Table 7: **Ranks of top 10 entities match perfectly with I-HARE ranks. C stands for Count of triples ($\times 10^6$) and P stands for Probability (rank score).**

Entity	Delta			Jan Dump			New Jun	I-HARE	Error (%)
	C	P	Rank	C	P	Rank	P	P	
`schema:description`	294	0.082	1	1,210	0.106	1	0.103	0.100	2.63
`schema:name`	59	0.030	3	192	0.031	2	0.031	0.031	0.13
`rdfs:label`	59	0.030	4	192	0.031	3	0.031	0.031	0.13
`skos:prefLabel`	59	0.030	5	192	0.031	4	0.031	0.031	0.13
`schema:version`	24	0.031	2	44	0.016	6	0.019	0.020	6.58
`schema:dateModified`	24	0.029	6	44	0.014	7	0.016	0.018	8.88
`rdf:type`	14	0.008	8	86	0.017	5	0.015	0.015	0.73
`skos:altLabel`	42	0.029	7	18	0.004	13	0.011	0.011	6.73
`schema:Dataset`	7	0.005	10	44	0.010	8	0.009	0.009	0.88
`schema:about`	7	0.004	12	44	0.009	9	0.008	0.008	0.67

[21] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. 2012. Linkedgeodata: A core for a web of spatial open data. *Semantic Web* 3, 4 (2012), 333–354.

[22] Julia Stoyanovich. 2007. EntityAuthority: Semantically Enriched Graph-Based Authority Propagation.. In *WebDB*.

[23] Giovanni Tummarello, Renaud Delbru, and Eyal Oren. 2007. Sindice.com: Weaving the Open Linked Data. In *ISWC/ASWC (Lecture Notes in Computer Science)*, Vol. 4825. Springer, 552–565.

[24] Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. 2014. Question answering over linked data (QALD-4). In *Working Notes for CLEF 2014 Conference*.

[25] Ricardo Usbeck. 2014. Combining Linked Data and Statistical Information Retrieval - Next Generation Information Systems. In *ESWC (Lecture Notes in Computer Science)*, Vol. 8465. Springer, 845–854.