Pay-as-you-go Approximate Join Top-k Processing for the Web of Data – Technical Report

Andreas Wagner
AIFB, Karlsruhe Institue of Technology
Karlsruhe, Germany
a.wagner@kit.edu

Veli Bicer
IBM Research, Smarter Cities Technology Centre
Dublin, Ireland
velibice@ie.ibm.com

Thanh Duc Tran
AIFB, Karlsruhe Institue of Technology
Karlsruhe, Germany
ducthanh.tran@kit.edu

ABSTRACT

In recent years, the amount of RDF data on the Web has drastically increased. For an effective search over such a large Web of data, ranking of results is crucial. To allow efficient query processing of ranked queries, *top-k join processing* has been proposed, which aims at computing k top-ranked results, without complete result materialization. However, Web search poses novel challenges. Most notably, users are frequently willing to sacrifice result accuracy in favor of result computation time. Thus, there is a strong need to *approximate the top-k results*. Previous work on approximate top-k processing, however, is not applicable for the join top-k setting – it solely targets the *selection top-k problem*. Further, existing approaches require *offline computed score statistics* to be available. Unfortunately, many important kinds of Web search queries, e.g., keyword or spatial queries, commonly *query-/user-dependent ranking* is employed. Thus, one only has score information at runtime. In this paper, we address these shortcomings and propose a novel approximate top-k join framework, well-suited for Web search queries and ranking. For this, all necessary score statistics are learned via a *pay-as-you-go training at runtime*. We study our approach in a theoretical manner, and *formally show its efficiency as well as effectiveness.* Further, we conducted experiments on benchmarks comprising synthetic as well as real-world Web data/-queries. Our results are very promising, as we could achieve up to 65% time savings, while maintaining a high precision/recall.

1. INTRODUCTION

With the proliferation of the Web of data, RDF (Resource Description Format) has gained a significant amount of attention. In fact, RDF has become an accepted standard for publishing data on the Web.[1] Adhering to a set of *triples* $\{\langle s, p, o\rangle\}$, RDF data forms a data graph, cf. Fig. 1-a. Every triple describes a particular entity (the *subject*) s through a *predicate*/*object* pair: p, o.

Web Search Ranking. For web-scale data, queries frequently produce a large number of results (*bindings*), thereby making *ranking* a key factor for an effective search. Each result has a ranking score associated, measuring its relevance for the user/information need. In a Web search setting, however, ranking functions often need to *reflect on query constraints or user characteristics*, for judging a binding's relevance.

Example 1. *Find movies with highest ratings, featuring an actress "Audrey Hepburn", and playing close to Rome, cf. Fig. 1.*

Exp. 1 would require a ranking function to incorporate the movie rating, quality of keyword matches for "Audrey Hepburn", and distance of the movie's location to Rome. While one may assume that a higher `rating` value is preferred by any user and query, *scores for keyword and location constraint dependent on query and/or user characteristics*. For instance, in order to rank a triple t for "Audrey Hepburn", a function may measure the edit distance between that keyword and triple t's attribute value. Notice, given another keyword (e.g., only "Audrey"), t's attribute value would yield a different score. Further, depending on the user's geographic knowledge of Italy, she may have different notions of "closeness" to Rome, e.g., distance $\leq$ 100 km. These three ranking criteria could be aggregated via a summation, Fig. 1-c. Ranking functions of that kind are commonly referred to as *query-/user-dependent ranking* [1, 7, 37], and are employed for many important Web search queries, e.g., keyword, spatial or temporal queries, cf. [4, 8, 9, 22, 23].

Join Top-k Processing. Most RDF query processing approaches focus on efficiency gains via query optimization, indexing or join techniques – *without taking ranking scores into account* [16].

On the other hand, work on relational databases has shown that result computation efficiency may be drastically improved by rank-aware query processing [18]. Rank-aware approaches (also known as *top-k query processing*) aim at computing k top-ranked bindings without full materialization of the result set. That is, after computing a certain number of bindings, the algorithm allows to terminate early, as it can guarantee that no binding with higher score, than those already found, exists [18].

Web search queries commonly feature one or more joins to combine multiple triples into a result [16]. Focusing on such queries, we target a particular flavor of top-k processing: the *join top-k* problem. Here, a complete binding is obtained by joining single triples, and the score of a complete binding is an aggregation of its comprised triples' scores [18].

Two recent papers proposed join top-k for the Web of data [25, 39]. However, these works miss to target common Web search behavior/information needs. That is, search on Web data is mostly performed by end-users having information needs that do not require a high result accuracy or completeness. In fact, while systems may compute a large number of top-ranked results, users oftentimes only investigate a small fraction of those results. For instance, in Exp. 1 a system could return the top-50 movie results. However, a user may just visit one or two results, until she discovers a movie of interest. Thus, users likely "miss" relevant results, because of their search behavior and information need. This drives the need for ap-

[1] `http://webdatacommons.org`

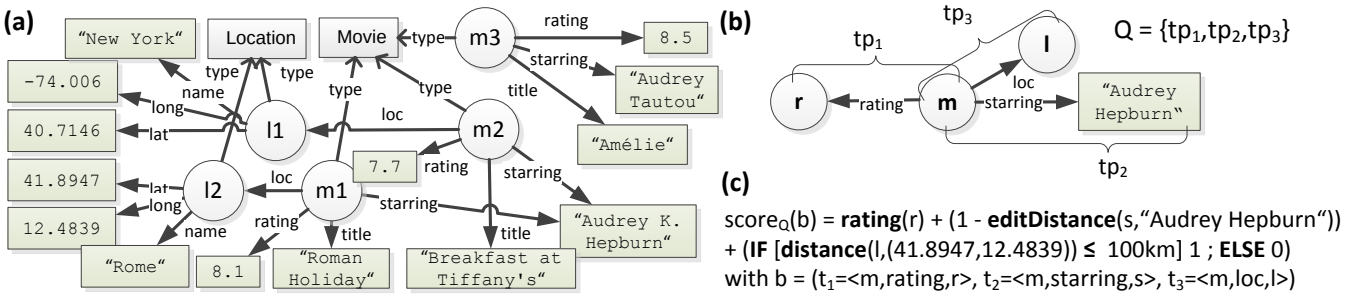

Figure 1: (a) Data about the movies "Roman Holiday", "Breakfast at Tiffany's", and "Amélie". (b) Query graph asking for a movie `starring` "Audrey Hepburn". (c) Scoring function aggregating (via a summation) scores for triple pattern bindings (bold): movie ratings, edit distance w.r.t. "Audrey Hepburn", and distance of the movie's location to Rome (lat: 41.8947, long: 12.4839) ≤ 100 km. While the first constraint is query- and user-independent, the keyword constraint is query-dependent, and the location ranking is user-dependent, due to user-defined notion of "closeness" (here: ≤ 100 km).

proximated Web search query processing: *a system should be able to trade off result accuracy/completeness for computation time.*

Approximate Join Top-k Processing. Approximate top-k strategies, allowing false positive/negative top-k bindings, have been proposed before [2, 3, 28, 35, 38]. Unfortunately, these approaches are not well-suited Web search:

(P1) *Binding Probability.* RDF data adheres to a fine-grained triple model, and frequently many joins are required to form a query. However, existing approaches are not targeted at join top-k, but instead aim at the *selection top-k problem* [2, 3, 28, 35, 38]. Selection top-k is a very different task, as it computes top-ranked results, where each result is a "single" entity. In particular, no joins of multiple triples/entities are considered [18]. Applying such works for approximate join top-k is not straightforward, because those approaches do not capture the "binding probability" of a partial result:

Example 2. *Say a partial binding $b = (*, \langle m_2,$ `starring`$,$ *"Audrey K. Hepburn"* $\rangle, *)$ in Fig. 1-a matches the* `starring` *constraint, tp_2, in Fig. 1-b, but still has to be joined with bindings for its unevaluated patterns (tp_1 and tp_3). The binding probability estimates how likely b will join with bindings for tp_1 and tp_3, thereby contributing to one or more complete bindings.*

(P2) *Score Probability.* To determine the probability for a partial binding to have a "sufficiently" large score to lead to a top-k result, score probability distributions are employed:

Example 3. *Partial binding b in Exp. 2 has a known/fixed score for its triple $\langle m_2,$* `starring`*,* *"Audrey K. Hepburn"* $\rangle$*. However, bindings for pattern tp_1 and tp_3 are unknown, and their scores are modeled by means of a probability distribution.*

Web search often uses an user-/query-dependent ranking, e.g., for keyword or spatial queries. For these ranking functions, *scores are only known at runtime*, see Exp. 1. Existing approaches, however, require ranking scores to be available *at offline time*, in order to estimate and store the score probability distributions, e.g., via histograms [2, 3, 28, 35, 38]. Thus, for user-/query-dependent ranking, previous works would need *to load all matching triples for each pattern into memory, compute their scores, and construct the necessary score statistics/distributions at runtime.* Notice, this must be done every time a query is issued. Such a procedure, however, would nullify the advantages of top-k processing, as materialization of bindings from disk is commonly the most expensive task in query processing.

This problem is exacerbated by the fact that score probability distributions must also capture aggregated scores from multiple patterns. For instance, in Exp. 3 we would need to estimate a distribution over scores for bindings of $tp_1 \bowtie tp_3$. As tp_3 is ranked via an user-dependent functions (Exp. 1), we again have no offline score information about the scores for $tp_1 \bowtie tp_3$. So, one would either need to materialize/sample the join $tp_1 \bowtie tp_3$ at runtime, or (given a summation as score aggregation) compute the desired score distribution as a convolution of the two probability distributions for tp_1 and tp_3. However, the former is associated with prohibitive costs, and the latter restricts the score aggregation to be a summation as well as imposes a harsh independence assumption.

In order to address problem P1 and P2, we propose an approximate join top-k strategy well-suited for the Web of data.

With regard to binding probabilities, *we reuse existing works on selectivity estimation* [24, 31, 32, 36]. Traditionally, selectivity estimation has been exploited to estimate the result set size for a given query, and to optimize join processing accordingly. In the following, we show how to make use of such solutions in order to approximate the binding probability.

For score probabilities, we propose a flexible approach based on Bayesian inferencing. In particular, we support any kind of ranking function, including query-/user-dependent functions. For this, *we use scores seen during join processing as "training examples" to iteratively learn score probability distribution at runtime.*

Contributions. Our contributions are as follows:

- While previous works solely aimed at top-k approximation for a relational setting, this is the first work towards an approximate join top-k processing for Web search.
- Our approach is lightweight, i.e., we do not require offline knowledge about ranking functions or score distributions. Every score statistic needed can be learned at runtime following the pay-as-you-go paradigm.
- We provide a theoretical analyses of our approach, and formally show its efficiency as well as effectiveness. In particular, we proof our distribution learning to have a constant space complexity, and a runtime complexity bounded by the result size. Furthermore, we give bounds for the approximation error of our overall approach.
- We implemented our approach and conducted experiments on two RDF benchmarks. Evaluation results are promising, as we could achieve time savings of up to 65%, while still having a high precision/recall.

Outline. We outline preliminaries in Sect. 2, and present the approximate top-k join in Sect. 3. In Sect. 4, we discuss evaluation results. Last, we give an overview over related works in Sect. 5, and conclude with Sect. 6.

2. PRELIMINARIES

Data Model. Given a set of edge labels ℓ, a RDF data is a labeled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell)$, where $\mathcal{V} = \mathcal{V}_E \uplus \mathcal{V}_A$ with entity nodes as $\mathcal{V}_E$ and attribute nodes as $\mathcal{V}_A$. *Triples* are given by edges $\mathcal{E} = \{\langle s, p, o\rangle\}$, with $s \in \mathcal{V}_E$ as subject, $p \in \ell$ as predicate, and $o \in \mathcal{V}_E \uplus \mathcal{V}_A$ as object. An example is depicted in Fig. 1-a.

Query and Result Model. We abstract from a particular type of queries, e.g., keyword or spatial queries, and model queries as basic graph patterns (BGPs), thereby capturing a key fragment of SPARQL.[2] A BGP query Q constitutes a directed labeled graph $Q = (\mathcal{V}^Q, \mathcal{E}^Q)$, with $\mathcal{V}^Q = \mathcal{V}^Q_V \uplus \mathcal{V}^Q_C$ as disjoint union of variable, $\mathcal{V}^Q_V$, and constant nodes, $\mathcal{V}^Q_C$. Edges $\mathcal{E}^Q$ are called *triple patterns*, with each pattern adhering to $tp = \langle s, p, o\rangle$, where $s \in \mathcal{V}^Q_V$, $p \in \ell$, and $o \in \mathcal{V}^Q_V \uplus \mathcal{V}^Q_C$. For example, pattern $\langle m$, `starring`, "Audrey Hepburn"$\rangle$ has one variable, m, one constant, "Audrey Hepburn", as object, and `starring` as predicate, Fig. 1-b. As shorthand we write Q as a *set of its triple patterns*: $Q = \{tp_i\}_i$.

A *binding* b for a query Q is a vector $(t_1, \ldots, t_n)$ of triples, such that each triple t_i *matches* (defined below) exactly one pattern tp_i in Q, and triples in b form a connected subgraph of $\mathcal{G}$. Via the matching of patterns in Q to triples, b *binds* variables to nodes in the data. Formally, for binding b there is a function $\mu_b : \mathcal{V}^Q_V \mapsto \mathcal{V}$, mapping every variable in Q to an entity/attribute value node in $\mathcal{V}$.

During query processing *partial* bindings, which feature some patterns with no matching triples, may occur. We refer to such a pattern, say tp_i, as *unevaluated*, and write $*$ in b's i-th position: $(t_1, \ldots, t_{i-1}, *, t_{i+1}, \ldots, t_n)$. For a partial binding b we denote its evaluated pattern as $Q(b) \subseteq Q$, and its unevaluated pattern as $Q^u(b) = Q \setminus Q(b)$. b is *complete*, if all pattern have been evaluated.

A binding b *comprises* a binding b', if any matched triple t_i in b' is also contained in b at position i. We say b' *contributes* to b.

Example 4. *In Fig. 1-b,* $Q = \{tp_1, tp_2, tp_3\}$. *Partial binding* $b_{31} = (*, *, t_{31} = \langle m_1,$ `loc`, $l_1\rangle)$ *in Fig. 2-a matches pattern* tp_3, *while* $Q^u(b_{31}) = \{tp_1, tp_2\}$ *are unevaluated.* b_{31} *binds variable* m *and* l *to entity* m_1 *and* l_1, *respectively. Complete binding* $b = (t_{12}, t_{21}, t_{31})$ *comprises partial binding* b_{31}, *and* b_{31} *contributes to* b.

Ranking Function. For quantifying the relevance of a binding b w.r.t. query Q, we employ a *ranking function*: $score_Q : \mathcal{B}^Q \mapsto \mathbb{R}$, with $\mathcal{B}^Q$ as set of all partial/complete bindings for Q. More precisely, $score_Q(b)$ is given by an aggregation over b's triples: $score_Q(b) = \bigoplus_{t \in b} score_Q(t)$, with $\oplus$ as monotonic aggregation function. Function $score_Q$ *is specific for* Q, *and the user who issued* Q. Thus, we allow for user-/query-dependent ranking of triple patterns bindings. A ranking function for our example is in Fig. 1-c. Note, $score_Q$ could be defined *as part of the query*, e.g., by means of an "extended" `ORDER BY` clause in SPARQL.

Sorted Access. For every pattern tp_i in query Q, a *sorted access* sa_i retrieves *matching triples in descending score order*. Previous works on join top-k over Web data discussed efficient sorted access implementations for RDF stores [25, 39]. Let us present simple approaches for our running example, Fig. 2-a:

Example 5. *Given the keyword pattern* $tp_2 = \langle m,$ `starring`, *"Audrey Hepburn"* $\rangle$, *a sorted access must materialize all triples, which have a value that contains "Audrey" or "Hepburn". After materialization, these triples are sorted with descending similarity w.r.t. that keyword – measured via, e.g., edit distance. Thus, sorted access* sa_2 *loads three triples comprising "Audrey" or "Hepburn", and sorts them according to their edit distance to "Audrey Hepburn". On the other hand, for pattern* $\langle m,$ `loc`, $l\rangle$, *an R-tree on the attribute pair (*`lat`, `long`*) may be used [14]. This offline computed index yields two hits:* l_1 *and* l_2. *While* l_1 *is an exact match, thus, ranking its triple* t_{31} *with max. score* 1, l_2 *is more distant from Rome. Note,* l_2 *and triple* t_{32}, *respectively, is only loaded if needed, i.e., if join-*2 *pulls on* sa_3 *a second time. An index for attribute* `rating` *can be constructed offline. Here, triples are sorted with descending attribute value. Sorted access* sa_1 *can simply iterate over this index, and materialize a triple each time join-*1 *pulls.*

[2] `http://www.w3.org/TR/rdf-sparql-query/`

Partial bindings retrieved from sorted accesses are combined via joins. That is, an equi-join combines two (or more) inputs, each of them either being another join or a sorted access. This way, multiple joins form a tree. For instance, three sorted accesses are combined via two joins in Fig. 2-a.

It is important to notice: if a join input i produces a partial binding b, its set of unevaluated triple patterns, $Q^u(b)$, is the same for any binding from that input. Thus, we say input i has a set of unevaluated patterns $Q^u(i)$ associated. For instance, input i_1 has $Q^u(i_1) = \{tp_2, tp_3\}$ as unevaluated patterns, Fig. 2-a.

Problem. Our goal is to produce k high-ranked, complete query bindings that may differ from the true top-k results in terms of false positives/negatives. These approximations aim at saving computation time and input depth (#bindings pulled from sorted accesses). For this, we use a top-k test: for a given partial query binding, we estimate its probability for contributing to the final top-k results, and discard partial bindings having only a minor probability.

In our work, we exploit *conjugate priors* from the field of Bayesian statistics for learning necessary probability distributions.

Bayesian Inference. Let Θ be a set of parameters. One can model *prior beliefs* about these parameters in the form of probabilities: $\Theta \sim P(\Theta \mid \alpha)$, with $\Theta \in \Theta$ [15]. α is a vector of *hyperparameters* allowing to parametrize the prior distribution. Suppose we observe relevant data $\mathbf{x} = \{x_1, \ldots, x_n\}$ w.r.t. Θ, where each $x_i \sim P(x_i \mid \Theta)$. Then, the dependency between observations $\mathbf{x}$ and prior parameters Θ can be written as $P(\mathbf{x} \mid \Theta)$. Using the Bayes theorem we can estimate a *posterior* probability, capturing parameters Θ conditioned on observed events $\mathbf{x}$. In simple terms, a posterior distribution models how likely parameters Θ are, in light of the seen data $\mathbf{x}$, and the prior beliefs [15]:

$$P(\Theta \mid \mathbf{x}, \alpha) \propto P(\mathbf{x} \mid \Theta) \cdot P(\Theta \mid \alpha) = \frac{P(\mathbf{x} \mid \Theta) \cdot P(\Theta \mid \alpha)}{\sum_\Theta P(\mathbf{x} \mid \Theta) P(\Theta)} \quad (1)$$

Example 6. *For the* `rating` *pattern* tp_1 *in Fig-2-a, scores are based on rating values. So, we can compute sufficient statistics, i.e, mean* $\bar{x}_1 = 8.1$ *and variance* $s_1^2 = 0.16$, *for these scores at offline time, cf.* $stat_1$ *in Fig-2-b. Such statistics represent prior beliefs about the true distribution, which is capturing only those scores observed for bindings of* tp_1 *that are part of a complete binding. Only triple* t_{11} *and* t_{12} *contribute to complete bindings, thus, only their scores should be modeled via a distribution. We update the prior beliefs using scores samples* $\mathbf{x}$ *observed during query processing, thereby learning the true (posterior) distribution as we go.*

As we are interested in estimating an *unobserved* event x^*, we need to calculate the *posterior predictive distribution*, i.e., the distribution of new events given observed data $\mathbf{x}$:

$$P(x^* \mid \mathbf{x}, \alpha) = \sum_\Theta P(x^* \mid \Theta) P(\Theta \mid \mathbf{x}, \alpha) \quad (2)$$

An important kind of Bayesian priors are the *conjugate priors* [15]:

Definition 1 (Conjugate Prior)**.** *A prior distribution family* $\mathcal{D}$ *for a parameter set* Θ *is called conjugate iff* $P(\Theta) \in \mathcal{D} \Rightarrow P(\Theta \mid \mathbf{x}) \in \mathcal{D}$.

Intuitively, conjugate priors require the posterior and prior distribution to belong to the same distribution family. In other words, these priors provide an "computational convenience" as they give a closed-form of the posterior distribution [15]. Thus, posterior computation is easy and efficient.

3. APPROXIMATE TOP-K JOIN

We now present an *approximated* top-k join processing algorithm, allowing a trade-off between result computation efficiency and accuracy. For this, we extend the well-known Pull/Bound Rank Join [34] with a novel *probabilistic component* $\mathcal{PC}$.

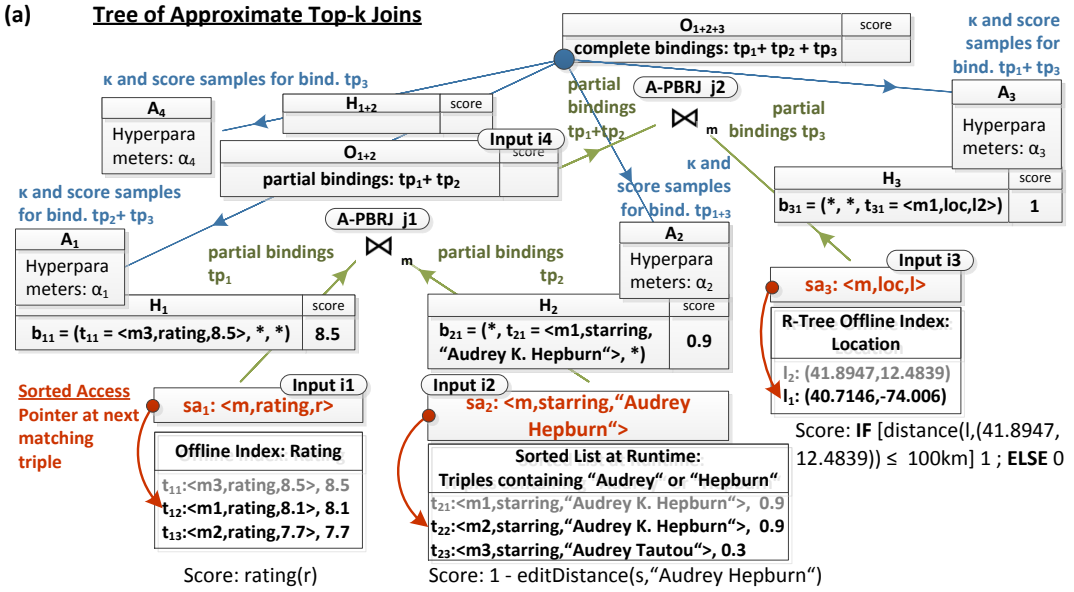

Figure 2: (a) Tree of approximate rank joins, combining three sorted accesses (one for each triple pattern in Fig. 1-b). Two information flows occur between operators in the tree: partial bindings (green), and score samples (blue). (b) Sufficient statistics calculated based on scores observed at indexing time ($stat_1$) and runtime ($stat_2$ and $stat_3$), respectively.

In particular, we will show that our instantiation of $\mathcal{PC}$ is *efficient and effective*. For the former, we show our score distribution training to have a constant space complexity, and a runtime complexity bounded by the result size (Lemma 1 and 2). For the latter, we discuss the quality of learned score distributions in Thm. 1 and 2, and provide an upper bound for approximation errors in Thm. 3.

3.1 Approximate Rank Join Framework

We define an approximate Pull/Bound Rank Join (A-PBRJ) comprising three parts: a pulling strategy $\mathcal{PS}$, a bounding strategy $\mathcal{BS}$, and a probabilistic component $\mathcal{PC}$. $\mathcal{PS}$ determines the next join input to pull from [34]. The bounding strategy $\mathcal{BS}$ gives an upper bound, β, for the maximal possible score of unseen results for that join [34]. Last, we use $\mathcal{PC}$ to estimate a probability for a partial binding to contribute to the final top-k result.

Approximate Pull/Bound Rank Join. The A-PBRJ is depicted in Algo. 1. Following [34], on line 5 we check whether output buffer $\mathbf{O}$ comprises k complete bindings, and if there are unseen bindings with higher scores – measured via bound β. If both conditions hold, the A-PBRJ terminates and reports $\mathbf{O}$. Otherwise, $\mathcal{PS}$ selects an input i to pull from (line 6), and produces a new partial binding b from the sorted access on input i, line 7. After materialization, we update β using bounding strategy $\mathcal{BS}$.

Example 7. By means of strategy $\mathcal{PS}$, join j_2 decides to first pull on sorted access sa_3, and materialize partial binding t_{31}, see Fig. 2-a. Then, join j_2 pulled on input i_4 (join j_1), which in turn pulled on its input i_1 (sa_1) loading binding t_{11}, and afterwards on input i_2 (sa_2) marginalizing t_{21}. $t_{11} \bowtie t_{21}$ on variable m fails as $m_3 \neq m_1$.

In line 9, $\mathcal{PC}$ estimates the probability for partial binding b leading to a complete top-k binding: *the top-k test*. If b fails this top-k test *b is pruned*. That is, we do not attempt to join it, and do not insert it in $\mathbf{H}_i$, which holds "seen" bindings from input i. Otherwise, if this test holds, b is further processed (lines 10 - 15). That is, we join b with seen bindings from the other input j, and add results to $\mathbf{O}$. Further, b is inserted into buffer $\mathbf{H}_i$, line 11. For learning the necessary probability distributions, $\mathcal{PC}$ trains on seen bindings/-scores in $\mathbf{O}$, line 13. Notice, we continuously train $\mathcal{PC}$ throughout the query processing – every time "enough" new bindings are in $\mathbf{O}$, line 12. $\mathcal{PC}$ requires parameter κ for its pruning decision. κ holds

Algorithm 1: Approx. Pull/Bound Rank Join (A-PBRJ).

Params: Pulling strategy $\mathcal{PS}$, bounding strategy $\mathcal{BS}$, probabilistic component $\mathcal{PC}$.
Index : Sorted access sa_i and sa_j for input i and j, resp..
Buffer : Output buffer $\mathbf{O}$. $\mathbf{H}_i$ and $\mathbf{H}_j$ for "seen" bindings pulled from sa_i and sa_j. Hyperparameter buffer $\mathbf{A}$.
Input : Query Q, result size k, and top-k test threshold τ.
Output : Approximated top-k result.

1 **begin**
2 $\mathcal{PC}$.initialize()
3 $\beta \leftarrow \infty$
4 $\kappa \leftarrow -\infty$
5 **while** $|\mathbf{O}| < k$ ***or*** $\min_{b' \in \mathbf{O}} score_Q(b') < \beta$ **do**
6 $i \leftarrow \mathcal{PS}$.input()
7 $b \leftarrow$ next triple pattern binding from sa_i
8 $\beta \leftarrow \mathcal{BS}$.update($b$)
  // top-k test
9 **if** $\mathcal{PC}$.*probabilityTopK*(b,κ) $> \tau$ **then**
10 $\mathbf{H}_j \bowtie \{b\}$ and add each binding to $\mathbf{O}$
11 Add b to $\mathbf{H}_i$
12 **if** *#new bindings* $\mathbf{b}$ *in* $\mathbf{O} \geq$ *training threshold* **then**
13 $\mathcal{PC}$.train($\mathbf{b}$)
14 Retain only k top-ranked bindings in $\mathbf{O}$
15 **if** $|\mathbf{O}| \geq k$ **then** $\kappa \leftarrow \min_{b' \in \mathbf{O}} score_Q(b')$
16
  // return approximated top-k results
17 **return** $\mathbf{O}$

the the smallest currently known top-k score (line 15). Note, κ is initialized as $-\infty$, line 4. See Fig. 2-a for a tree of A-PBRJs.

Choices for $\mathcal{PS}$ and $\mathcal{BS}$. Multiple works proposed bounding, e.g., [12, 17, 26, 34], and pulling strategies, e.g., [17, 27]. Commonly, the *corner bound* [17] is employed as bounding strategy:

Definition 2 (Corner Bound)**.** *For a join operator, we maintain u_i and l_i for each input i. u_i is the highest score observed from i, while l_i is the lowest observed score on i. If input i is exhausted, l_i is set to $-\infty$. The bound for scores of unseen join results is β =*

$\max\{u_1 \oplus l_2, u_2 \oplus l_1\}$.

In example Fig. 2-a, join j_1 currently has $\beta = \max\{8.5 + 0.9, 0.9 + 8.5\}$, with $u_1 = l_1 = 8.5$, and $u_2 = l_2 = 0.9$.

On the other hand, the *corner-bound-adaptive strategy* [17] is frequently used as pulling strategy $\mathcal{PS}$:

Definition 3 (Corner-Bound-Adaptive Pulling)**.** *The corner-bound-adaptive pulling strategy chooses the input i such that: $i = 1$ iff $u_1 \oplus l_2 > u_2 \oplus l_1$, and $i = 2$ otherwise. If $u_1 \oplus l_2 = u_2 \oplus l_1$, the input with the smaller # unseen partial bindings is chosen.*

For instance, as $8.5 + 0.9 = 0.9 + 8.5$ in join j_1, with both inputs having 2 unseen partial bindings, either input may be selected.

3.2 Probabilistic Component

The probabilistic component implements the top-k test, which follows a simple intuition: For a partial binding b, a subset of query patterns are evaluated, $Q(b)$, while some others are not, $Q^u(b)$. Thus, scores (bindings) of unevaluated patterns are uncertain. *However, partial bindings for already evaluated patterns restrict the space of likely/possible scores (bindings) for unevaluated patterns.*

We model these binding and score uncertainties via two probabilities: (1) Probability for partial binding b contributing to one or more complete bindings (*binding probability*). (2) Probability distribution over scores of complete bindings (*score probability*).

Binding Probability. To address the former probability, we use a selectivity estimation function *sel*. Simply put, given a query Q, $sel(Q)$ estimates the probability that there is at least one binding for Q, see [24, 31, 32, 36]. For example, selectivity of pattern $tp_3 = \langle m, \texttt{loc}, l\rangle$ is $sel(tp_3) = \frac{2}{3}$, because out of the three movie entities only two have a `loc` predicate, cf. Fig. 1-a.

By means of function *sel*, we define a *complete binding indicator* for partial binding b:

$$\mathbb{1}(Q^u(b) \mid b) \coloneqq \begin{cases} 1 & \text{if } sel(Q^u(b) \mid b) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Intuitively, $\mathbb{1}(Q^u(b) \mid b)$ models *whether matching triples for b's remaining unevaluated patterns can exist, given variable assignments dictated by b*.

That is, $Q^u(b) \mid b$ is a set of patterns $\{\overline{tp_i}\}$, such that pattern $tp_i \in Q^u(b)$, and each variable v in tp_i that is bound by b is replaced with its assignment in b, $\mu_b(v)$, resulting in a new pattern $\overline{tp_i}$.

Example 8. *Consider Fig. 2-a and partial binding $b_{11} = (t_{11} = \langle m_3,$ `rating`, $8.5\rangle, *, *)$. Here, $Q^u(b_{11}) \mid b_{11}$ is $\{\langle m_3,$ `starring`, "Audrey Hepburn"$\rangle, \langle m_3,$ `loc`, $l\rangle\}$, which is obtained by replacing variable m in tp_2 and tp_3 with its assignment in b_{11} ($\mu_{b11}(m) = m_3$).*

It is important to notice that the complete binding indicator (Eq. 3) is independent of a particular *sel* implementation – *any selectivity estimation for BGP queries may be used.*

For our experiments we reused work from [31, 32]. The authors employed indexes for triple patterns with two constants: `SP`, `PS`, `SO`, `OS`, `PO`, and `OP`. Each index maps a $\langle val_1, val_2\rangle$ pair to its cardinality, i.e., the number of its matching triples in the data. For instance, $\langle m_1,$ `starring`$\rangle$ would map to 1 in Fig. 1-a. However, the binding indicator only requires a selectivity estimation function to make a boolean decision: either $sel(Q^u(b) \mid b) > 0$ or not. Thus, not all six indexes are necessary: `SP`, `PO`, and `SO` are sufficient. A rudimentary implementation of $sel(Q^u(b) \mid b)$ returns 1 iff

$$\forall_{\langle s,p,o\rangle \in Q^u(b) \mid b, \text{ where } s \in \mathcal{V}_E} \texttt{SP}.card(\langle s, p\rangle) > 0 \quad \wedge \tag{4a}$$

$$\forall_{\langle s,p,o\rangle \in Q^u(b) \mid b, \text{ where } o \in \mathcal{V}_E \uplus \mathcal{V}_A} \texttt{PO}.card(\langle p, o\rangle) > 0 \quad \wedge \tag{4b}$$

$$\forall_{\langle s,p,o\rangle \in Q^u(b) \mid b, \text{ where } s \in \mathcal{V}_E \wedge o \in \mathcal{V}_E \uplus \mathcal{V}_A} \texttt{SO}.card(\langle s, o\rangle) > 0 \tag{4c}$$

and 0 otherwise, with *card* as cardinality function.

Example 9. *For $Q^u(b_{11}) \mid b_{11}$ in Exp. 8, $\mathbb{1}(Q^u(b_{11}) \mid b_{11}) = 0$, because selectivity for both patterns is* 0*. In our implementation probe in* `SP` *for $\langle m_3,$* `loc`$\rangle$ *returns "pair does not exist". Thus, our $sel(Q^u(b_{11}) \mid b_{11})$ correctly returns* 0*.*

Score Probability. Given a partial binding b, let X^s_b be a random variable for modeling scores of complete bindings comprising b. Further, *scores for bindings of b's unevaluated patterns, $Q^u(b)$*, are captured via $X^s_{Q^u(b)}$. Then, it holds:

$$P(X^s_b \geq x) = P\left(X^s_{Q^u(b)} \geq \delta(x, b)\right) \tag{5}$$

where $\delta(x, b) \coloneqq x - score_Q(b)$. LHS of Eq. 5 may be simplified to the RHS, because partial binding b already has a "certain" score, $score_Q(b)$, and only the scores for its unevaluated patterns are unknown. So, $\delta(x, b)$ may be seen as "delta" between b's score and a desired score x. For instance, b_{31} has a score of 1, however, scores for bindings to tp_1 and tp_2 are uncertain, and modeled via $X^s_{Q^u(b_{31})}$.

Top-k Test. Having introduced above probabilities, we can define the top-k test. For a partial binding b, we are interested in b's probability of contributing to a complete binding in the top-k results. Thus, we use (1) the binding probability, i.e., the complete binding indicator, to determine whether b might contribute to any complete binding, and (2) the score probability to estimate scores for complete bindings comprising b:

$$P(X^s_b \geq \kappa) \cdot \mathbb{1}(Q^u(b) \mid b) > \tau \Leftrightarrow \tag{6a}$$

$$P\left(X^s_{Q^u(b)} \geq \delta(\kappa, b)\right) \cdot \mathbb{1}(Q^u(b) \mid b) > \tau \tag{6b}$$

with κ as the smallest currently known top-k score (see Algo. 1 line 15), and $\tau \in [0, 1]$ as test threshold.

Discussion. Threshold τ provides a key instrument for Web search systems, as it always to adjust result accuracy, depending on user and information need. For instance, a system may decide to compute top-50 results in total, and increase τ every time a new top-ranked binding could be reported. Generally speaking, τ should not be conceived as a constant, but rather as a function in reported top-ranked results. This way, systems may reflect on the typical user behavior, who frequently only visit a small subset of the computed results.

Second, note that κ in Eq. 6 refers to the smallest currently known top-k binding score. Thus, as long as no k complete bindings have been found, κ is set to $-\infty$ (Algo. 1, line 4), and the score probability is always 1. So, the only reason for a partial binding b to be pruned is, if it fails the complete binding indicator. That is, if b is not expected to contribute to any complete binding.

Last, each top-k test computation causes costs in the form of probability estimations. We will provide empirical results regarding those costs in our evaluation. On the other hand, recent work introduced a "cost-aware" rank join, which schedules sorted and random accesses based on their associated costs [27] – this line of work can be directly applied here. In fact, the top-k test may be treated as "one more" access in their optimization problem [27].

3.3 Score Distribution Learning

With regard to the binding probability, we only require a selectivity estimation function. For this, *all necessary indexes can be computed offline*. We leave the selectivity estimation problem to orthogonal work, and focus on score probabilities in this paper.

Estimation of score probabilities is rather difficult: (1) In a join top-k setting, we are only interested in scores for complete bindings. Consider join j_1 in Fig. 2-a: we are not interested in "any" aggregation of two scores for bindings of tp_1 and tp_2, e.g., $score_Q(t_{11}) + score_Q(t_{21}) = 8.5 + 0.9$. Instead, we wish to capture only scores of valid join results, e.g., $score_Q(t_{11}) + score_Q(t_{23})$. (2) We allow

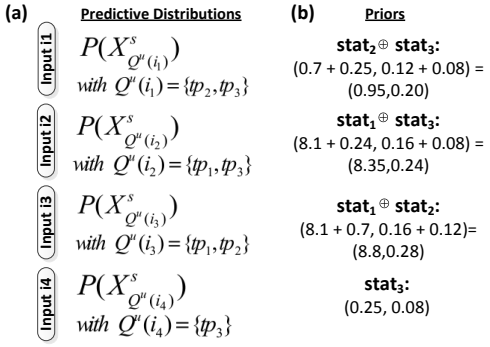

Figure 3: (a) Four predictive score distributions, one for each join input, together with their priors. For instance, $X^s_{Q^u(i_1)}$ models scores for bindings to $tp_2 \bowtie tp_3$, which are comprised in complete bindings. (b) Priors are based on sufficient statistics. Note, aggregation function $\oplus$ in Fig. 1-c is a summation. For instance, $\text{stat}_1 \oplus \text{stat}_3$ becomes $(8.1 + 0.24, 0.16 + 0.08)$.

for user-/query-dependent ranking. For those functions, no offline score statistics can be constructed. At the same time, for other ranking functions, e.g., based on the `rating` predicate, we do have offline score statistics. Thus, we aim for a flexible framework allowing to incorporate any offline information.

3.3.1 Overview

We estimate probabilities for $X^s_{Q^u(b)}$ by *approximating its true distribution* with $X^s_{Q^u(i)}$ with i as the input from which b was pulled, and $Q^u(i)$ as set of i's unevaluated patterns:

$$P\left(X^s_{Q^u(b)}\right) \approx P\left(X^s_{Q^u(i)}\right)$$

Example 10. *For instance, in Fig. 2-a we approximate $P(X^s_{Q^u(b_{11})})$ with $P(X^s_{Q^u(i_1)})$, as binding b_{11} is produced by input i_1. $X^s_{Q^u(i_1)}$ captures scores for bindings to patterns $\{tp_2, tp_3\}$.*

So, instead of learning a distribution for every partial binding b, we train a score distribution $X^s_{Q^u(i)}$ for each input i. For our example we learn four distributions, cf. Fig. 3.

Relying on $X^s_{Q^u(i)}$, we provide an implementation of $\mathcal{PC}$.initialize(), $\mathcal{PC}$.train() in Algo. 2, and $\mathcal{PC}$.probabilityTopK() in Algo. 3 – as required by our A-PBRJ framework in Algo. 1.

3.3.2 Pay-as-you-go Distribution Learning

We use conjugate priors, which allow a pay-as-you-go distribution learning for $X^s_{Q^u(i)}$. With runtime ranking scores, we have no information about the true distribution of $X^s_{Q^u(i)}$. In such a scenario, a common assumption is to use a Gaussian score distribution. Thus, we employ a Gaussian conjugate prior (Eq. 7a). However, we also outline how to extend our approach to other distributions, if offline knowledge about $X^s_{Q^u(i)}$ is available (Sect. 3.3.3).

Using a Gaussian conjugate prior with unknown mean and unknown variance, prior and posterior distribution can be decomposed as: $P(\mu, \sigma^2 \mid \alpha) = P(\mu \mid \sigma^2, \alpha) \cdot P(\sigma^2 \mid \alpha)$ [15]. The mean μ follows a Gaussian, Eq. 7b, and the variance σ^2 a inverse-Gamma distribution, Eq. 7c. Hyperparameters $\alpha_0 = (\mu_0, \eta_0, \sigma_0^2, \nu_0)$ parameterize both distributions, where μ_0 is prior mean with quality η_0, and σ_0^2 is prior variance with quality ν_0 [15]:

$$X^s_{Q^u(i)} \sim \text{normal}\left(\mu, \sigma^2\right) \tag{7a}$$

$$\mu \mid \sigma^2 \sim \text{normal}\left(\mu_0, \frac{\sigma^2}{\eta_0}\right) \tag{7b}$$

$$\sigma^2 \sim \text{inverse-gamma}\left(0.5 \cdot \nu_0, 0.5 \cdot \nu_0 \sigma_0^2\right) \tag{7c}$$

Prior Distribution. Prior initialization is implemented by means of the $\mathcal{PC}$.initialize() method in Algo. 1 (line 2). For each input i we specify a prior distribution for $X^s_{Q^u(i)}$ via prior hyperparameters α_0. For α_0 we require sufficient score statistics in the form of a sample mean, $\bar{x} = \frac{1}{n}\sum_{x_i \in \mathbf{x}} x_i$, and a sample variance $s^2 = \frac{1}{(n-1)}\sum_{x_i \in \mathbf{x}}(x_i - \bar{x})^2$, with $\mathbf{x}$ as sample. There are multiple ways to obtain the necessary score samples, depending on what kinds of ranking functions are used:

Example 11. *Fig. 2-b depicts three sufficient statistics based on information from the sorted accesses: (1) Offline information in the case of sa_1. That is, scores are known before runtime, thus, $\bar{x}_1 = 8.1$ and $s_1^2 = 0.16$ can be computed offline. (2) Online information for access sa_2. Recall, the list of matching triples for keywords "Audrey" and "Hepburn" must be fully materialized. So, $\bar{x}_2 = 0.7$ and $s_2^2 = 0.12$ may be computed from runtime score samples. (3) Last, given access sa_3, we have neither offline scores, nor a fully materialized list of triples (sa_3 loads a triple solely upon a pull request). In lack of more information, we assume each score to be equal likely, i.e., a uniform distribution. With min. score as 0 and max. score as 1: $\bar{x}_3 = 0.25$ and $s_3^2 = 0.08$.*

Algorithm 2: $\mathcal{PC}$.train()

Params: Weight $w \geq 1$ for score sample $\mathbf{x}$.
Buffer : Buffer $\mathbf{A}$ storing hyperparameters.
Input : Complete bindings $\mathbf{B} \subseteq \mathbf{O}$, and join j.

1 **begin**
  // train hyperparameters for each input
2 **foreach** *input i in join j* **do**
    // load prior hyperparameters for input i
3    $\alpha_n = (\mu_n, \eta_n, \sigma_n^2, \nu_n) \leftarrow \mathbf{A}_i$
    // get scores of bindings of unevaluated patterns
4   **foreach** *complete binding $b \in \mathbf{B}$* **do**
5   get b' *comprised in* b for patterns $Q^u(i)$
6   add $score_Q(b')$ to score sample $\mathbf{x}$
    // sample mean and variance
7    $\bar{x} \leftarrow \text{mean}(\mathbf{x}) = \frac{1}{n}\sum x_i$
8    $s^2 \leftarrow \text{var}(\mathbf{x}) = \frac{1}{(n-1)}\sum(x_i - \bar{x})^2$
    // posterior hyperparameters
9    $\nu_{n+1} \leftarrow \nu_n + w$
10   $\eta_{n+1} \leftarrow \eta_n + w$
11   $\mu_{n+1} \leftarrow \frac{\eta_n \mu_n + w\bar{x}}{\eta_{n+1}}$
12   $\sigma^2_{n+1} \leftarrow \frac{1}{\nu_{n+1}} \cdot \left(\nu_n \sigma_n^2 + (w-1)s^2 + \frac{\eta_n w}{\eta_{n+1}} \cdot (\bar{x} - \mu_n)^2\right)$
    // store hyperparameters for input i in join j
13   $\mathbf{A}_i \leftarrow \alpha_{n+1} = (\mu_{n+1}, \eta_{n+1}, \sigma^2_{n+1}, \nu_{n+1})$

Similar to [15], we initialize hyperparameters α_0 with: μ_0 as sample mean, σ_0^2 as sample variance, and $\eta_0 = \nu_0$ as sample size/-quality. For every input, we aggregate necessary sample means and variances for μ_0 and σ_0^2, respectively. The aggregation function for these statistics must be the same as for the ranking function, $\oplus$. For example, given input i_1 where $Q^u(i_1) = \{tp_2, tp_3\}$, we sum up (aggregate) statistics $stat_2$ and $stat_3$: $\bar{x}_2 + \bar{x}_3$ for μ_0, and $s_2^2 + s_3^2$ for σ_0^2. Similarly, priors for the remaining three distributions can be calculated, see Fig. 3. Further, η_0 and ν_0 are used to quantify the prior quality. For instance, $stat_1$ and $stat_2$ are exact statistics, while $stat_3$ relies on a uniform distribution. So, a prior may be weighted depending on its employed statistics. In other words, weighting reflects the "trustworthiness" of the prior.

Posterior Distribution. Having estimated a prior distribution, *we continuously update the distribution with seen scores during query processing.* Based on [15], the training procedure $\mathcal{PC}$.train()

is given in Algo. 2.

Intuitively, each time new complete bindings (Algo. 1, line 12) are produced, all distributions $X^s_{Q^u(i)}$ could be trained. That is, complete binding scores are used to update hyperparameters from the previous n-th training iteration, α_n, resulting in new posterior hyperparameters, α_{n+1}. For this, we use standard training (Algo. 2, lines 11-12) as discussed in [15]:

$$\mu_{n+1} = \frac{\eta_n \mu_n + w\bar{x}}{\eta_{n+1}} \tag{8a}$$

$$\sigma^2_{n+1} = \frac{1}{\nu_{n+1}} \cdot \left(\nu_n \sigma^2_n + (w-1)s^2 + \frac{\eta_n w}{\eta_{n+1}} \cdot (\bar{x} - \mu_n)^2 \right) \tag{8b}$$

In simple terms, the prior mean μ_n is updated with the new sample mean $\bar{x}$, Eq. 8a, and the prior variance σ^2_n is updated with the sample variance s^2, Eq. 8b. Each input "extracts" its own score sample $\mathbf{x}$ (Algo. 2, lines 5-6), from which the sample mean and variance is computed. This is because every $X^s_{Q^u(i)}$ models scores for different "unevaluated" patterns.

Prior hyperparameters are weighted via η_n and ν_n. Further, for each hyperparameter update, a parameter w is used as weight, which indicates the quality of the score samples $\mathbf{x}$. In our experiments, we set w as sample size. Finally, new hyperparameters α_{n+1} are stored (Algo. 2, line 13), and used as prior for the next training.

Example 12. *Consider input i_1, and say $\eta_0 = \nu_0 = 1$, its prior is $\alpha_0 = (0.95, 1, 0.20, 1)$, cf. Fig. 3. We observe scores $\mathbf{x} = \{x_1, x_2\}$ from $\mathbf{B} = \{(t_{12}, t_{21}, t_{31}), (t_{13}, t_{22}, t_{32})\}$, with $w = |\mathbf{x}| = 2$, $x_1 = 1.9 = score_Q(t_{21}) + score_Q(t_{31})$, and $x_2 = 0.9 = score_Q(t_{22}) + score_Q(t_{32})$. Then, $s^2 = 0.26$, $\bar{x} = 1.4$, and posterior hyperparameters are:*

$$\eta_1 = \nu_1 = 1 + 2 = 3$$

$$\sigma^2_1 = \frac{1}{3} \cdot \left(0.2 + (2-1) \cdot 0.26 + \frac{(1.4 - 0.95)^2}{3} \right) = 0.26$$

$$\mu_1 = \frac{(0.95 + 2 \cdot 1.4)}{3} = 1.25$$

After each such update only posterior hyperparameters are stored, thereby making the learning highly space efficient:

Lemma 1 (Distribution Learning Space Complexity)**.** *Given an A-PRBJ operator j, at any time during query processing, we require $O(1)$ of space for score distribution learning.*

Proof Sketch. Given an A-PRBJ operator, for every of its inputs a single parameter vector (hyperparameters α) is stored after a training iteration (Algo. 2, line 13). As each vector, α, features a fixed number of parameters, the space consumption remains constant, i.e., independent of #training iterations ∎

Further, for the learning time complexity we can show:

Lemma 2 (Distribution Learning Time Complexity)**.** *Given an A-PRBJ operator j, and $\mathbf{B}$ complete bindings, score learning time complexity is bounded by $O(|\mathbf{B}|)$.*

Proof Sketch. Given an A-PRBJ operator, and a set of new results $\mathbf{B}$: A score sample, $\mathbf{x}$, is constructed (Algo. 2, lines 5-6) with $O(|\mathbf{B}|)$ complexity. Mean and variance is computed from $\mathbf{x}$ in $O(|\mathbf{x}|)$ time. However, as $|\mathbf{x}| \leq |\mathbf{B}|$, it holds that $O(|\mathbf{x}|) \in O(|\mathbf{B}|)$. Notice, computation of mean and variance could also be done while collecting the sample in lines 5-6. Further, hyperparameters are updated via $\mathbf{x}$ in constant time (Algo. 2, lines 9-12). Thus, every training iteration costs overall $O(|\mathbf{B}|)$ ∎

Predictive Distribution. In Algo. 3, we provide an implementation of the $\mathcal{PC}$.probabilityTopK() method, see Algo. 1 line 9. At any point during query processing, one may need to perform a top-k test. For this, our approach allows to always give a distribution for $X^s_{Q^u(i)}$ based on the currently known hyperparameters α_n (Algo. 3, line 2). As hyperparameters are continuously trained, the distributions improve over time.

More specifically, we use the posterior predictive distribution. This distribution estimates probabilities for *new* scores, based on observed scores, and prior distribution. As shown in [15], this distribution can be easily obtained in a closed form as non-standardized Student's t-distribution with ν_n degrees of freedom, see Algo. 3, line 3. Being able to compute the posterior predictive in such an easy manner is an advantage of a Gaussian conjugate prior – for other priors this estimation may be more complex [15].

Using the posterior predictive distribution, we may give an approximation of $P(X^s_{Q^u(b)})$ via $P(X^s_{Q^u(i)})$, Algo. 3, line 4. Last, in line 5 we compute the binding probability (Eq. 3) by means of the selectivity estimation function. Now we have obtained binding as well as score probability, and can return a probability for b contributing to the top-k results, line 6 (Eq. 6).

Algorithm 3: $\mathcal{PC}$.probabilityTopK()

Buffer : Buffer $\mathbf{A}$ storing hyperparameters.
Input : Partial bindings b, input i, and join j.
Output : Probability that b will result in one (or more) final top-k bindings.

1 **begin**
  `// load hyperparameters for input i at join j`
2 $\alpha_n = (\mu_n, \eta_n, \sigma^2_n, \nu_n) \leftarrow \mathbf{A}_i$
  `// posterior predictive distribution based on current` α_n
3 $X^s_{Q^u(i)} \sim t_{(\nu_n)}\left(x \mid \mu_n, \frac{\sigma^2_n(\eta_n+1)}{\eta_n}\right)$
  `// approximate score probability`
4 $p_S \leftarrow P\left(X^s_{Q^u(b)} \geq \delta(\kappa, b)\right) \approx P\left(X^s_{Q^u(i)} \geq \delta(\kappa, b)\right)$
  `// compute binding probability`
5 $p_{\mathcal{B}} \leftarrow \mathbb{1}(Q^u(b) \mid b)$
  `// probability that` b `contributes to top-`k
6 **return** $p_S \cdot p_{\mathcal{B}}$

3.3.3 Discussion

Refined Conjugate Priors. Whenever one has offline information about the true distribution of $X^s_{Q^u(i)}$ (or good approximation for it), we may replace the Gaussian conjugate prior in Eq. 7 with a more suitable one. For this, only minor changes in the training (Algo. 2), and predictive distribution estimation (Algo. 3, line 3) are required. Both these tasks are well-known problems in the literature [15]. No further modifications are needed – the top-k test works with any valid score distribution for $X^s_{Q^u(i)}$.

A wide variety of discrete/continuous conjugate priors are known, thus, in the best case, there is a conjugate prior for the true distribution of $X^s_{Q^u(i)}$. Otherwise, if no matching conjugate prior exists, we can exploit a mixture of multiple conjugate priors: $\sum_i w_i P_i(\Theta)$ with each $P_i(\Theta)$ being a conjugate prior, and w_i as weights such that $\sum_i w_i = 1$ and $0 < w_i < 1$. Notice, [5] showed that any distribution from the exponential family could be approximated (arbitrarily close) by means of a mixed conjugate prior.

Maintenance. We require maintenance of binding and score probabilities. Binding probabilities are estimated via a selectivity estimation function, Eq. 3. Maintenance of these statistics varies with the specific selectivity estimation implementation, and is out of scope for this work.

With regard to score probabilities, we train posterior distributions during query processing. Thus, only for prior distributions sufficient statistics are needed. These statistics may differ depending on the ranking functions employed. For example, given the

`rating` ranking in Fig. 1-c, sufficient statistics can be computed before runtime in the form of a sample mean and variance: $stat_1 = (\bar{x}_1 = 8.1, s_1^2 = 0.16)$, Fig. 2-b. In fact, one may even store further distribution characteristics, e.g., distribution skewness or symmetry. This way, more refined conjugate priors could be estimated (see paragraph above). In contrast, for user-/query-dependent ranking, e.g., the keyword constraint "Audrey Hepburn" in Fig. 1-c, scores are unknown before runtime. Thus, no sufficient statistic can be stored and/or maintained. However, in such a case, a minimal and maximal score may be kept. For instance, for the `distance` ranking constraint, we would store a minimal and maximal score as 0 and 1, respectively. This way, we may assume an uniform score distribution as naive prior, and compute mean and variance as: $stat_3 = (\bar{x}_3 = 0.25, s_3^2 = 0.08)$, cf. Fig. 2-b and Exp. 11.

3.4 Theoretical Analysis

In this section, we discuss theoretical aspects concerning the *effectiveness of the A-PBRJ operator*. That is, we discuss the quality of the learned score distributions (Thm. 1 and 2), and provide bounds for the approximation error, cf. Thm. 3.

Distribution Quality. The aggregation function $\oplus$ for our ranking function $score_Q$ could be any monotonic function. However, when we restrict the aggregation to a summation (see Fig. 1-c), we can formally show that a Gaussian distribution/conjugate prior in Eq. 7 is a good approximation for the true distribution of $X^s_{Q^u(i)}$. Notice, many common aggregations employ summations, e.g., TF-IDF inspired functions may be represented by summations [38]. For such a summation-based aggregation function it holds:

Theorem 1. *Given a query $Q = \{tp_k\}$ and $X^s_{Q^u(i)} = \sum_{tp_k \in Q^u(i)} X^s_{tp_k}$, the Central Limit Theorem (CLT) holds:*

$$\frac{\sum_k \left(X^s_{tp_k} - \mu_k\right)}{\sqrt{\sum_k \sigma_k^2}} \underset{n\to\infty}{\sim} normal(0,1)$$

with n as the number of patterns in $Q^u(i)$, and $X^s_{tp_k}$ as random variable modeling scores of bindings for pattern tp_k. Further, μ_k and σ_k^2 stand for the finite mean and variance of $X^s_{tp_k}$.

Informal Proof Sketch. As we do not have knowledge about the ranking functions, $score_Q$, or the distributions for $X^s_{tp_k}$, we can only outline a very informal sketch of proof in the following.

Our argumentation is based on two assumptions: (A1) Recall, we define a *separate* ranking function $score_Q$ for every triple pattern tp_k in query Q, cf. Sect. 2. In particular, each function computes its score solely by considering the triple binding of "its own" pattern, tp_k. Thus, assuming scores from different ranking functions to be independently distributed is a reasonable simplification. Formally, for every two pattern tp_1 and tp_2 in Q, we assume: $X^s_{tp_1} \perp X^s_{tp_2}$. (A2) We assume each random variable $X^s_{tp_k}$ to have a finite mean μ_k and variance σ_k^2. Notice, most common distributions feature a finite mean and variance. So, this assumption does not restrict $score_Q$ and $X^s_{tp_k}$, respectively.

In its simplest form, the Central Limit Theorem is only applicable to i.i.d. random variables [13, p. 329]. However, in the Lindeberg Theorem this restriction is lifted, i.e., every variable $X^s_{tp_k}$ may adhere to a different distribution [13, p. 330]:

Lemma 3 (Lindeberg Condition)**.** *If*

$$\lim_{n\to\infty} \frac{1}{s_n^2} \sum_k E\left((X^s_{tp_k} - \mu_k)^2\right) \cdot \mathbb{1}\left(|X^s_{tp_k} - \mu_k| \geq \varepsilon s_n\right) = 0$$

holds, where $s_n^2 = \sum_k \sigma_k^2$, then the Central Limit Theorem in Thm. 1 holds.

Further, it is known that [6, p. 368]:

Lemma 4. *If each random variable $X^s_{tp_k}$ is uniformly bounded, and $\lim_{n\to\infty} s_n = \infty$, then the Lindeberg Condition in Lemma 3 holds.*

As our $score_Q$ function is bounded in $[0, 1]$ (defined in Sect. 2), every variable $X^s_{tp_k}$ is also bounded: $P(0 \leq X^s_{tp_k} \leq 1) = 1$. Further, the variance σ_k^2 can be expected to be > 0 for each $X^s_{tp_k}$, because $X^s_{tp_k}$ models ranking scores. That is, ranking scores are supposed to vary between different results, in order to assist an users in discovering results of interest. Therefore, it holds that $s_n = \sum_k \sigma_k^2 \to \infty$ with $n \to \infty$ ∎

In simple terms, Thm. 1 states that the true distribution of $X^s_{Q^u(i)}$ converges (in the number of patterns in $Q^u(i)$) to a Gaussian distribution. Now, the question is: "how fast" does $X^s_{Q^u(i)}$ converge to a Gaussian distribution? For this convergence it holds:

Theorem 2 (Berry-Esseen Theorem)**.** *Let $\rho_k = E\left(|X^s_{tp_k}|^3\right) < \infty$ be the third absolute normalized moment of $X^s_{tp_k}$. Then, it holds [13, p. 355]:*

$$\sup_x |F(x) - \phi(x)| \leq C \cdot \frac{\sum_k \rho_k}{\left(\sum_k \sigma_k^2\right)^{\frac{3}{2}}}$$

with $\phi(x)$ as standard Gaussian CDF, and $F(x)$ as exact CDF of $X^s_{Q^u(i)}$. Further, tp_k, μ_k, and σ_k^2 are defined as in Thm. 1.

C is a constant in Thm. 2, and is currently estimated as $0.4097 \leq C \leq 0.5600$ [13, p. 355]. Thus, intuitively, Thm. 2 gives an absolute bound on how close the true distribution of $X^s_{Q^u(i)}$ is to a Gaussian.

Approximation Error. Let X_i^e denote a random variable for the error introduced by pruning from input i. This error may be measured as the number of pruned partial bindings from i, which would have contributed to the final top-k result. Then, it holds that:

Lemma 5. *Random variable X_i^e follows a binomial distribution such that: $X_i^e \sim bin(c_i, \varepsilon + \tau)$. c_i stands for the # bindings pulled from input i, in order to produce the top-k results. Further, τ is the error threshold from Eq. 6, and ε is a small constant ≥ 0.*

Proof Sketch. From a given input i we pull c_i partial bindings. Every of these bindings could be pruned "wrongfully" by the top-k test in Eq. 6, either because the binding probability was falsely estimated as 0, or because the score probability was smaller than the threshold τ. Let the probability for the former be bounded by a constant $\varepsilon \geq 0$, while the probability for the latter is known to be $\leq \tau$. Thus, the overall probability for a partial bindings to be wrongfully pruned in Eq. 6 is $\leq \varepsilon + \tau$. Further, pulling c_i partial bindings from input i may be conceived as c_i trails, where a wrongly pruned binding is a "hit". Therefore, we can model X_i^e by means of binomial distribution, with c_i as number of trails, and $\varepsilon + \tau$ as "success" probability. Formally, $X_i^e \sim \text{bin}(c_i, \varepsilon + \tau)$, cf. Lemma 5 ∎

Note, ε is a small error introduced by the binding indicator function $\mathbb{1}(Q^u(b) \mid b)$, see also Eq. 3. This error depends on the accuracy of the selectivity estimation, however, as the binding indicator only requires a binary decision, its induced error is frequently very small. In fact, our simplistic implementation in Eq. 4 is exact, i.e., leads to $\varepsilon = 0$.

Given a tree of one or more A-PBRJ operators having a total of n inputs, let X^e capture the overall error, i.e., number of false positives/negatives in the top-k results. We can show that X^e also adheres to a binomial distribution:

Theorem 3. *$X^e \sim bin\left(\sum_{i=1}^n c_i, \varepsilon + \tau\right)$, with input depth c_i, threshold τ, and ε, as defined in Lemma 5.*

Proof Sketch. Given a tree of joins having n inputs: $\{i_1, \ldots, i_n\}$. Let every input i_j pull c_j partial bindings, in order for the join tree to produce the desired k top-ranked results. Further, the error (# wrongly pruned partial bindings) for each input i_j is modeled via variable $X_i^e \sim \text{bin}(c_i, \varepsilon + \tau)$, cf. Lemma 5. False positives/negatives results comprised in the final top-k bindings are caused by wrongly pruned partial bindings. More precisely, for a given input i_j, every wrongfully pruned partial binding could lead to a false positive/negative top-k result. Thus, errors from the individual inputs "sum up" to a total error – captured by X^e. In other words, random variable X^e is a summation over the random variables X_i^e. Further, errors made in the inputs are independent of each other, i.e., every pair of variables, X_i^e and X_j^e, is independent: $X_i^e \perp X_j^e$. Thus, X^e is again a binomial distribution with $\sum_{i=1}^n c_i$ trails and "success" probability $\varepsilon + \tau$: $X^e \sim \text{bin}\left(\sum_{i=1}^n c_i, \varepsilon + \tau\right)$ ∎

Exploiting Thm. 3, we can give an expected error as a function of threshold τ: $E\left(X^e\right) = \sum_{i=1}^n c_i \cdot (\varepsilon + \tau)$.

4. EVALUATION

We conducted experiments for (1) *analyzing the efficiency and effectiveness of the A-PBRJ operator* in Sect. 4.2, and (2) *inspecting the behavior of our probabilistic component* in Sect. 4.3. By means of the former, we illustrate the overall performance of our approach, when compared with the exact PBRJ. The latter provides insights into overhead and accuracy of the probabilistic component.

4.1 Evaluation Setting

Data. We used two RDF benchmarks based on synthetic/real-world data: (1) The SP² benchmark features synthetic DBLP data comprising information about computer science bibliographies [33]. (2) The DBpedia SPARQL benchmark (DBPSB), which holds real-world data extracted from the DBpedia knowledge base [29]. For both benchmarks we used datasets of 10M triples each.

Queries. We employed queries from the SP² benchmark [33] and the DBPSB benchmark [29]. As both benchmarks comprise SPARQL queries, we translated the queries to our query model (basic graph patterns). Queries featuring no basic graph patterns were discarded, i.e., we omitted 12 and 4 queries in DBPSB and SP², respectively. This lead to 13 SP² queries and 120 queries for DBPSB – a comprehensive load of 133 queries in total.

The DBPSB queries were generated from 8 "seed queries" as proposed in [29]: each seed query comprises a special variable, which is randomly assigned a constant. Note, each constant is chosen such that the resulting query has a non-empty result. With every instantiation of that variable, a new benchmark query is given. Overall, we instantiated each of the 8 seed queries with 15 random bindings, resulting in a total of 120 queries. Queries varied in # pattern $\in [2, 9]$ and result sizes $\in [1, 5390436]$. Additional query statistics are in Table 1, and a complete query listing is given in the appendix, Sect. 8.

	SP² Queries	*DBPSB Queries*
# Queries	13	120
# Triple pattern	[2, 9]	[2, 4]
Mean(#Triple pattern)	5	2.8
Var(#Triple pattern)	6.4	0.6
# Results	[1, 5.4M]	[1, 50]
Mean(#Results)	590K	3.9
Var(#Results)	2.1B	51.6

Table 1: Query statistics for the SP² and DBPSB benchmark.

Systems. We randomly generated bushy query plans. For a given query, all systems rely on the same plan. We implemented three systems, solely differing in their join operator: (1) A system with *join-sort* operator, `JS`, which does not employ top-k processing, but instead produces all results, and then sorts them. (2) An *exact* rank join operator, `PBRJ`, featuring the corner-bound (Def. 2), and the corner-bound-adaptive pulling strategy in Def. 3. (3) Last, we implemented our *approximate* operator, `A-PBRJ`, see Sect. 3. All systems have indexes for random and sorted access.

Aiming at Web search ranking, we assume *no offline score information* for the `A-PBRJ` operator. Thus, we employ a Gaussian conjugate prior (with unknown mean and variance) for score probability learning. Training and top-k test implementation follows Algo. 2 and 3, see Sect. 3.3.2. Further, we used sufficient statistics based on a uniform distribution over [0, 1], as discussed in Exp. 11 for sorted access sa_3. Prior weights ν_0 and η_0 are both 1. Weight w in Algo. 2 is the sample size, $|\mathbf{x}|$. Scores for single triple pattern bindings are random (see below), and complete binding scores are computed as summation. For binding probabilities, we aimed at a simple implementation as presented in Eq. 4, Sect. 3.2. The necessary `SP`, `PO`, and `SO` indexes were loaded into memory.

We implemented all systems in Java 6. Experiments were run on a Linux server with two Intel Xeon 5140 CPUs at 2.33GHz, 48GB memory (16GB assigned to the JVM), and a RAID10 with IBM SAS 148GB 10K rpm disks. Before each query execution, all operating system caches were cleared. The presented values are averages collected over five runs.

Hypothesis (H1): We expect that `JS` is outperformed by `PBRJ`, as it computes all results for a query. Further, we expect `A-PBRJ` to outperform `JS` and `PBRJ`, by trading effectiveness for efficiency.

Parameters. We used parameters as follows: We vary the number of results to be computed: $k \in \{1, 5, 10, 20\}$.

Hypothesis (H2): We predict efficiency to decrease in parameter k for `A-PBRJ` and `PBRJ`. Effectiveness should not be affected.

We chose triple pattern binding scores, $score_Q(t)$, at random with distribution $d \in \{u, n, e\}$ (uniform, normal, and exponential distribution). By means of varying distributions, we aim at an abstraction from a particular ranking function, and examine performance for different "classes" of functions. We employed standard parameters for all distributions, and normalized scores to be in [0, 1].

Hypothesis (H3): `A-PBRJ`'s efficiency and effectiveness is not influenced by the choice of the ranking function and its score distribution, respectively.

We used top-k test thresholds $\tau \in [0, 0.8]$ for inspecting the trade-off between computation efficiency and effectiveness.

Hypothesis (H4): We expect efficiency of `A-PBRJ` to be increasing in τ, while its effectiveness will be decreasing in τ.

Metrics. We measure efficiency via: (1) Number of sorted and random accesses. (2) Time spend for top-k result computation. (3) Max. memory needed for buffering intermediate results.

As effectiveness metrics we use: (1) Precision: fraction of approximated top-k results being exact top-k results. (2) Recall: fraction of exact top-k results being reported as approximate results. Notice, precision and recall have identical values, as both share the same denominator k. We therefore discuss only precision results in the following. Further, precision is given as average over our query load, the so-called macro-precision. (3) Rank distance: approximate vs. exact top-k rank: $\frac{1}{k} \sum_{i=1,\ldots,k} |rank^*(b) - rank(b)|$, with $rank^*(b)$ and $rank(b)$ as approximated and exact rank for binding b [19]. (4) Score error: approximate vs. exact top-k score: $\frac{1}{k} \sum_{b=1,\ldots,k} |score_Q^*(b) - score_Q(b)|$, with $score_Q^*(b)$ and $score_Q(b)$ as approximated and exact score for binding b [38].

4.2 Evaluation: A-PBRJ

First, we the performance of the `A-PBRJ` system in terms of its

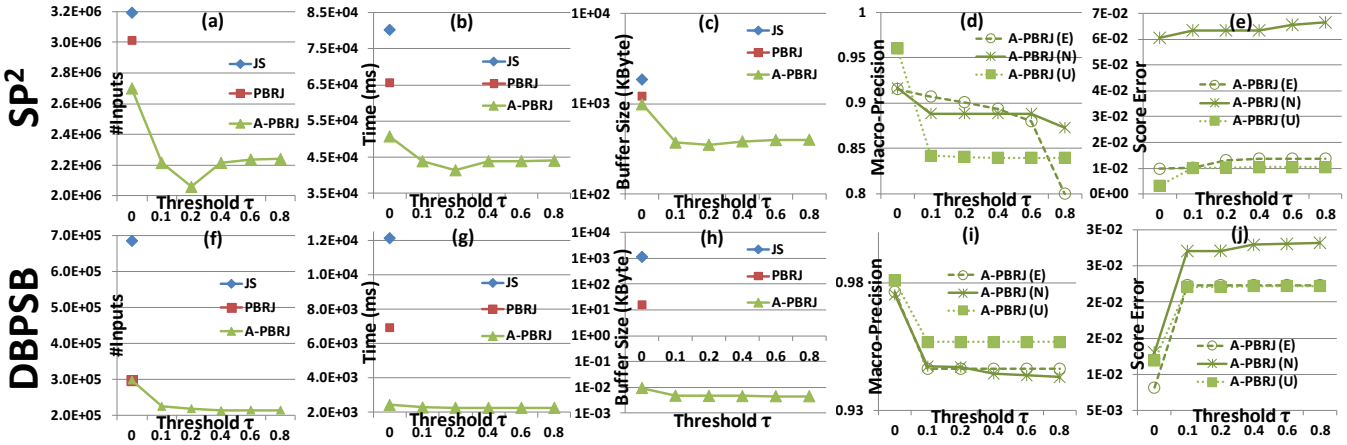

Figure 4: Average results over all queries from SP² and DBPSB: (a) + (f) Efficiency: #inputs vs. threshold τ. (b) + (g) Efficiency: time vs. threshold τ. (c) + (h) Efficiency: buffer sizes vs. threshold τ. (d) + (i) Effectiveness: macro-precision vs. threshold τ, given varying score distributions. (e) + (j) Effectiveness: score error vs. threshold τ, given varying score distributions.

effectiveness and efficiency.

Efficiency: Overall Results. Efficiency results are depicted in Fig. 4 (a)+(f), (b)+(g), and (c)+(h), as functions in threshold τ for SP² and DBPSB. As expected in hypothesis H1, we observed #inputs and computation time to be decreasing in τ, cf. Fig. 4 (a)+(f) and (b)+(g). For SP² (DBPSB), A-PBRJ needed up to 25% (23%) less inputs w.r.t. baseline PBRJ, and 30% (67%) regarding JS. We explain these gains with pruning of partial bindings, thereby omitting "unnecessary" joins and join attempts. Thus, A-PBRJ produces overall fewer partial and complete bindings, as determined by pruning of the top-k test. Fewer #inputs translated into average time savings of 35% (65%) as compared to PBRJ, and 47% (80%) w.r.t. JS, given SP² (DBPSB). Further, we noticed smaller buffers, holding less intermediate results, to contribute to these time savings, due to more efficient probing and maintenance of their hash tables.

Interestingly, we saw an increase in #inputs for $\tau \in [0.2, 0.4]$ in SP², and $\tau \in [0.4, 0.8]$ in DBPSB, Fig. 4 (a)+(f). For instance, comparing $\tau = 0.2$ and $\tau = 0.4$ in SP², A-PBRJ read 8% more inputs. DBPSB was less effected – we noticed a marginal increase of 2% for $\tau = 0.4$ vs. $\tau = 0.6$. We explain the increase in both benchmarks with a too "aggressive" pruning, i.e., too many partial bindings have been wrongfully pruned, leading to more inputs being read in order to produce the desired k results. More precisely, we inspected SP² benchmark runs for $\tau = 0.2$ vs. $\tau = 0.4$, and noticed a large number of partial bindings throughout the tree were pruned "too early", i.e., they would have led to a larger or even a complete binding. In both cases, β thresholds (Algo. 1, line 8), capturing the upper-bound for unseen join results, were not accurate any more, as they depended on seeing these pruned bindings. In fact, $\tau \in [0.6, 0.8]$ was even more aggressive, however, the ratio between pruned bindings and read inputs was high enough to compensate for the extra inputs, Fig. 4 (a). Note, such wrongfully pruned bindings not necessarily have an effect on result quality. That is, we frequently observed these bindings to not contribute to the exact top-k result. Overall, we saw a "sweet spot" at $\tau \approx 0.2$ for SP² and DBPSB – here, we noted pruning to be fairly accurate, i.e., only few partial bindings were wrongfully pruned. This is also reflected in high precision (recall) values for both benchmarks given $\tau \approx 0.2$: 88% (95%) in SP² (DBPSB) – as discussed in the following.

With regard to computation time for SP² and DBPSB queries, we noticed similar effects as for the #inputs, cf. Fig. 4 (b)+(g). In particular, the "sweet spot" at $\tau \approx 0.2$ is also reflected here. That is, for $\tau = 0.2$ A-PBRJ needs 40% (68%) less time than PBRJ. In comparison to JS, A-PBRJ is 51% (82%) faster in SP² (DBPSB). We explain such drastic time savings with the less #inputs being read, due partial bindings pruned by our top-k test. For instance, we were able to prune up to 40% (90%) of the inputs, given SP² (DBPSB). This lead to significantly less join attempts, and also smaller buffers (explained below). In fact, we noted that binding probabilities played a crucial role, as they oftentimes allowed reliable discovery (pruning) of partial bindings with little/no probability of leading to complete bindings. Thus, similar to a random access, binding probabilities allowed to "probe" unevaluated patterns.

Last, considering buffer sizes in SP² (DBPSB), we saw 60% (95%) less space consumption w.r.t. PBRJ, and 75% (99%) in contrast to JS, cf. Fig. 4 (c)+(h). Note, Fig. 4 (c)+(h) shows average buffers size per join in KByte, while employing a dictionary encoding for nodes in the data graph. Smaller buffers in A-PBRJ are possible due to fewer intermediate results and less inputs being loaded. Interestingly, as these buffers are based on hash tables, we observed smaller buffers to also have a positive effect on runtime behavior. That is, less maintenance was necessary, and they allowed for more efficient probing, due to less entries. So, buffer sizes also contributed to the above discussed time savings.

Efficiency: Sensitivity Analysis. Let us discuss efficiency sensitivity for different parameters – as expressed in hypotheses H2-H4.

As expressed by hypothesis H2, we observed #inputs and time to increase in k for A-PBRJ and PBRJ. For instance, comparing $k = 1$ and $k = 20$, A-PBRJ needed a factor of 1.2 (5.7) more time, given SP² (DBPSB). Similarly, 1.2 (6.8) times more inputs were consumed by A-PBRJ for SP² (DBPSB). Such a behavior is expected (H2), as more inputs/join attempts are required to produce a larger result. PBRJ leads to a similar performance decrease in SP². For instance, for $k = 1$ vs. $k = 20$ a factor of 1.3 (1.2) more inputs (time) are needed. Given DBPSB queries, an interesting observation is that PBRJ achieves a $50 - 60\%$ smaller performance decrease than A-PBRJ for $k = 1$ vs. $k = 20$. We explain this with many DBPSB queries having a cardinality ≤ 10, e.g., Q_1 where cardinality is 1. For such queries, A-PBRJ can not prune "more" bindings for a larger k, as the algorithm first needs to compute k complete results, which fails due to their small cardinality. Thus, A-PBRJ is more sensitive to k w.r.t. queries with cardinality $\leq k$.

Furthermore, we can confirm our hypothesis H3 with regard to system efficiency: *we did not find correlations between system performance and score distributions.* In other words, score distributions (ranking functions) had no impact on performance of the

A-PBRJ, given SP² or DBPSB. For SP², we noticed A-PBRJ to outperform PBRJ by 22% (42%) for *e* distribution, 21% (35%) given *u* distribution, and 8% (14%) for *n* distribution w.r.t. #inputs (time). For DBPSB queries, A-PBRJ resulted in different gains over PBRJ: 27% (65%) for *e* distribution, 23% (64%) given *u* distribution, and 21% (64%) for *n* distribution w.r.t. #inputs (time).

Last, with regard to parameter τ, we noted A-PBRJ efficiency to increase with $\tau \in [0, 2]$ for SP² (DBPSB) – partially confirming hypothesis H4. However, as outlined above, too aggressive pruning let to "inverse" effects. An important observation is, however, that our approach was already able to achieve performance gains with a very small $\tau < 0.1$. Here, partial bindings were pruned primarily due to their low binding probability. In fact, A-PBRJ could even save time for $\tau = 0$: 26% (60%) with SP² (DBPSB). We inspected queries leading to such saving, and saw that many of their partial bindings had a binding probability of 0. Thus, they were pruned even with $\tau = 0$. We argue that this a strong advantage of A-PBRJ: *even for low error thresholds (leading to minor effectiveness decrease), we could achieve significant efficiency gains.*

Effectiveness: Overall Results. Fig. 4 (d)+(i), and (e)+(j) depict effectiveness evaluation results for varying score distributions as average over all queries. More precisely, (d)+(i) give macro-precision, and (e)+(j) show score error – both as functions in τ. For space reasons, figures depicting the rank distance were omitted. Instead, we describe key observations in the text.

We observed high precision values $\in [0.8, 0.95]$ for both benchmarks and over all queries, cf. Fig. 4 (d)+(i). More precisely, we saw best results for a small $\tau < 0.1$, and the uniform distribution. However, given $\tau < 0.1$, all distributions led to very similar precision results $\in [0.9, 0.95]$ and $[0.95, 0.98]$ for SP² and DBPSB, respectively. This confirms our hypothesis H3: *A-PBRJ's effectiveness is not affected by a particular score distribution.* We explain these good approximations with accurate score distributions (discussed in Sect. 4.3), and reliable binding probabilities. In fact, our simplistic implementation in Eq. 4 led to *exact* binding probabilities, thereby pruning partial bindings *with certainty*. This is because Eq. 4 checks necessary (but not sufficient) conditions, which a partial bindings has to fulfill, in oder to contribute to a complete binding. This resulted in accurate pruning of many partial bindings – for some queries in DBPSB up to 97% of their total inputs. Such DBPSB queries featured highly selective patterns, and had only a small result cardinality ≤ 10, thereby allowing for an highly effective pruning via binding probabilities.

In order to quantify "how bad" false positive/negative results are, we employed the score error and rank distance metric. Score error results depicted in Fig. (e)+(j). We observed that rank distance (score error) was $\in [0.01, 0.02]$ ([0.07,0.11]) for a small $\tau < 0.1$, over all distributions and both benchmarks. We explain this we our high precision (recall), i.e., A-PBRJ led to only few false positive/negative top-*k* results given $\tau = 0$. *Further, as expected in H4, both metrics increased in τ, due to more pruning and false positives/negatives.* For instance, for $\tau = 0$ vs. $\tau = 0.8$ we noted an increase by a factor of 3.3 (2.6) for rank distance in SP² (DBPSB). Overall, rank distance and score error results were very promising: we saw an average score error of 0.03 (0.02) over all τ and queries, given SP² (DBPSB).

Effectiveness: Sensitivity Analysis. Concerning the sensitivity effectiveness w.r.t. parameter *k*, we can confirm the initial hypothesis H2: *k does not impact the A-PBRJ's effectiveness*. Given SP², we saw A-PBRJ to be fairly stable in different values for parameter *k*. For instance, macro-precision was in $[0.87, 0.89]$. Other metrics showed similar, minor fluctuations. Also for DBPSB, we noted only minor effectiveness fluctuations, e.g., macro-precision varied around 7% w.r.t. different *k*. We explain this good behavior with (1) quality of learned score distributions for SP² (discussed later), and (2) query characteristics of DBPSB. As for the latter, we noted that DBPSB mostly contained very selective queries having a cardinality ≤ 10. Thus, for such queries the risk of pruning "the wrong" partial bindings did not increase in *k*. Note, SP² featured very different queries – many SP² queries had a large cardinality $\gg 100$. Thus, we argue the stable performance of A-PBRJ for SP² to be an indicator for a reliable top-*k* test and score/binding probabilities.

As predicted in hypothesis H3, *we observed A-PBRJ to not be influenced by varying score distributions*, Fig. 4 (d)+(i), and (e)+(j). Given SP², we saw a macro-precision (over all queries and values for τ) of 0.86 for *u* distribution, 0.88 for *e* distribution, and 0.89 for *n* distribution. Also for the DBPSB benchmark, we observed only minor changes in macro-precision: 0.96 for *u* distribution, 0.95 for *e* as well as *n* distribution. We explain this with a good score distribution quality (see Sect. 4.3), leading to reliable top-*k* tests – independent of the actual triple scores. Note, we made similar observations for score error and rank distance metric. One exception was, however, the score error for distribution *n*, Fig. 4 (e)+(j). This distribution led to an higher score error than, e.g., the *u* distribution by a factor of 6.9 (1.13) for SP² (DBPSB). We analyzed these outlier queries in SP² and DBPSB: scores from wrongly pruned top-*k* results were higher for *n* than for other distributions. This caused an increase in the score error.

With regard to the effectiveness of A-PBRJ vs. parameter τ, we noticed that metrics over both benchmarks decreased with increasing τ. For instance, macro-precision decreased for $\tau = 0$ vs. $\tau = 0.8$ with 11% (5%), given SP² (DBPSB). Such a behavior is expected (H4), as chances of pruning "the wrong" bindings increase with higher τ values. Thus, while leading to efficiency gains (discussed above), a higher value for τ causes effectiveness loses. *Overall, this confirms hypotheses H4 and H1 that A-PBRJ trades off effectiveness for efficiency, as dictated by threshold τ.*

4.3 Evaluation: Probabilistic Component

In this section, we analyze the performance of the probabilistic component in terms of its efficiency and effectiveness. More specifically, as binding probabilities are estimated via a given selectivity estimation framework based on previous work [31, 32], we focus on learning and computation of score probabilities.

	SP²	*DBPSB*
Time (ms)	[1, 300]	[1, 24]
# Samples	[1, 4M]	[1, 50]
Avg. #Sample	400K	4

(a) Efficiency: average learning time and # learning samples.

Dist.	*SP²*	*DBPSB*
e	0.34	0.04
n	0.34	0.01
u	0.31	0.02

(b) Effectiveness: average *p*-value from the goodness-of-fit test.

Table 2: Efficiency and effectiveness of score distribution learning.

Efficiency. First, we analyze the overhead introduced by score distribution learning. For this, we measured the time needed for hyperparameter training, cf. Algo. 2 in Sect. 3.2. We set the training threshold, i.e., the # new bindings after which a new training procedure is triggered, to 1 (Algo. 1, line 12). Table 2-a gives average training times and # samples for distribution training.

We observed average learning times $\in [1, 300]$ ms ($[1, 24]$ ms) over all score distributions, queries and thresholds τ, given the SP² (DBPSB) benchmark. We noted the driving factor to be the overall query selectivity. That is, SP² queries often had a large cardinality $\gg 100$, which led to a high number (up to $4,227,732$) of training samples. In contrast, DBPSB featured highly selective queries, resulting in few training iterations – only up to 50 score

samples were available on average. Overall, this explains the additional training time (factor 12.5) needed for SP² queries, when compared to DBPSB. Note, one may easily cope with high cardinality queries by: (1) setting a larger training threshold, or (2) stop distribution learning, if distribution "quality" does not improve any more. However, such optimizations are left to future work.

Second, we measured the extra time required for performing a top-k test, see Algo. 3 in Sect. 3.2. On average over both benchmarks and all parameters, a top-k test needed 4.3K ns. This time comprises a selectivity estimation lookup for the binding probability, and the score probability computation. In contrast, a sorted (random) access took 26.8K ns (1.7M ns) on average. Thus, a top-k test is fairly cheap in comparison to a sorted/random access.

Effectiveness. For judging learning effectiveness, we captured how well the trained distributions "fits" the observed complete binding scores. More precisely, we applied the well-known Kolmogorov-Smirnov test [13], which measures, via a p-value in $[0, 1]$, whether a sample comes from the population of a specific distribution. Table 2-b shows p-values as averages over both benchmarks.

We observed drastic differences between p-values for SP² and DBPSB. SP² results were very promising, and reflect that learned distributions accurately capture the true scores of complete bindings. *This confirms hypothesis H3, as our distribution learning could achieve high-quality approximations: a p-value of* 0.34 *for distribution e as well as n, and* 0.31 *for u distribution*. With regard to DBPSB, we could not train good distributions, i.e., we measured poor p-values: 0.04 for distribution e, 0.01 given n, and 0.02 for u distribution. We explain this with the few training samples available in DBPSB queries, due to their high selectivity. As discussed above, SP² queries featured much more score samples, cf. Table 2-a. However, the interesting observation is that the overall approach, `A-PBRJ`, was hardly affected. In fact, `A-PBRJ` achieved a very high precision $\in [0.95, 0.98]$ for DBPSB. This is because the score probabilities are only relevant for the top-k test, if k complete bindings have been computed (see discussion in Sect. 3.2). However, for many DBPSB queries, their cardinality was $\leq k$. Thus, the low quality distributions had little to no effect.

Overall, we can conclude that the probabilistic component trains score distributions in an efficient and effective manner, if sufficient score samples are available.

5. RELATED WORK

Early work on rank-aware query processing aimed at the selection top-k problem [10, 11]. Here, the goal is to find k top-ranked results, where each result is an entity with n attributes, that is ranked according to m criteria, defined on those attributes [18].

To foster efficient result computation, *approximate selection top-k* techniques have been proposed [2, 3, 28, 35, 38]. [38] used score statistics to predict the highest possible complete score of a partial binding. Partial results are discarded, if they are not likely to contribute to a top-k result. Focusing on distributed top-k queries, [28] employed histograms to predict aggregated score values over a space of data sources. Anytime measures for selection top-k have been introduced by [2, 3]. For this, the authors used offline score information, e.g., histograms, to predict complete binding scores at runtime. In [35], a framework for approximate top-k processing under budgetary constraints, as well as algorithms for scheduling sorted and random access have been presented.

In contrast, we target the join top-k problem. In this setting, scores are assigned to single triples, and a complete result is obtained by joining these triples. The score of a complete binding is an aggregation of the scores for its comprised triples [18]. A large body of work aimed at an *exact join top-k* processing, e.g., [12, 17, 20, 21, 27, 30, 34]. In particular, a rank-join based on the A* algorithm, J*, was proposed in [30]. [17] introduced the hash rank-join algorithm (HRJN), which was further addressed in [21]. [20] investigated the join top-k problem given a non-monotonic aggregation function. In [34], the authors developed an algorithm template, the Pull/Bound Rank Join (PBRJ), which covers previous work on rank join approaches. Using the PBRJ, the authors introduced a novel bound for the case that a join features more than two inputs, and a triple has multiple scores. Extending [34], efficiency and costs aspects were further discussed in [12]. A cost-aware scheduling strategy for random and sorted accesses in rank-joins was presented in [27]. Recently, two works adapted top-k join processing to RDF data and SPARQL queries [25, 39].

As result accuracy is not of high importance for Web search, we aim at an *approximate join top-k* processing. However, to the best of our knowledge, there is no work addressing this problem. Further, approximate selection top-k techniques are not directly applicable. This is because the selection top-k problem solely ranks "single" entities, and does not consider joins. In a join top-k setting, multiple triples are combined via joins. This leads to many partial bindings that never contribute to a top-k binding, because they do not satisfy the join conditions. Thus, an approximate join top-k should not only capture a score probability, but also a probability for satisfying the query constraints. However, approximate selection top-k strategies solely judge the likelihood of a partial binding leading to a top-k result by means of score probabilities [2, 3, 28, 35, 38].

Moreover, existing approximate top-k approaches heavily rely on "offline ranking". That is, scores must be known before runtime for computing statistics, e.g., histograms [2, 3, 28, 35, 38], or suitable/approximated score distributions [38]. However, we target a Web search context, where systems oftentimes exploit user-/query-dependent ranking functions. Given such a ranking, scores are only known at runtime. In fact, our approach not only supports such an "online ranking", but allows a flexible integration of any kind of ranking function and offline available score information.

6. CONCLUSION

In this paper, we introduced an approximate join top-k algorithm, A-PBRJ, well-suited for the Web of data. For this, we extended the well-known PBRJ framework with a novel probabilistic component. This component allows us to estimate the probability of a partial query binding (1) to lead to a complete binding, and (2) to have a score higher than the smallest currently currently known top-k score. We conducted a theoretical analyses showing that our approach features an efficient and effective instantiation of the probabilistic component. Furthermore, we implemented and evaluated the A-PBRJ system by means of two state-of-the-art Web data benchmarks. Our results are promising, as we could achieve times savings of up to 65% over the baselines, while maintaining a high precision/recall.

7. REFERENCES

[1] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *SIGMOD*, 2006.

[2] B. Arai, G. Das, D. Gunopulos, and N. Koudas. Anytime measures for top-k algorithms. In *VLDB*, 2007.

[3] B. Arai, G. Das, D. Gunopulos, and N. Koudas. Anytime measures for top-k algorithms on exact and fuzzy data sets. *VLDB Journal*, 18:407–427, 2009.

[4] A. Baid, I. Rae, J. Li, A. Doan, and J. Naughton. Toward scalable keyword search over relational data. *VLDB*, 2010.

[5] J. M. Bernardo and A. F. M. Smith. *Bayesian Theory*. John Wiley & Sons, 1994.
[6] P. Billingsley. *Probability and Measure*. Wiley-Interscience, 3 edition, 1995.
[7] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *TODS*, 31(3):1134–1168, 2006.
[8] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD*, 2006.
[9] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *VLDB*, 2009.
[10] R. Fagin. Combining fuzzy information from multiple systems. *JCSS*, 58:83–99, 1999.
[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *SIGMOD*, 2001.
[12] J. Finger and N. Polyzotis. Robust and efficient algorithms for rank join evaluation. In *SIGMOD*, 2009.
[13] A. Gut. *Probability: A Graduate Course*. Springer, 2012.
[14] A. Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, 1984.
[15] P. D. Hoff. *A First Course in Bayesian Statistical Methods*. Springer, 2009.
[16] K. Hose, R. Schenkel, M. Theobald, and G. Weikum. Database foundations for scalable RDF processing. In *RR*, 2011.
[17] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB Journal*, 13:207–221, 2004.
[18] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.
[19] M. Kendall and J. D. Gibbons. *Rank Correlation Methods*. Oxford University Press, 1990.
[20] B. Kimelfeld and Y. Sagiv. Incrementally Computing Ordered Answers of Acyclic Conjunctive Queries. In *NGITS*, volume 4032 of *Lecture Notes in Computer Science*, pages 141–152. 2006.
[21] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. RankSQL: query algebra and optimization for relational top-k queries. In *SIGMOD*, 2005.
[22] F. Li, K. Yi, and W. Le. Top-k queries on temporal data. *The VLDB Journal*, 19(5):715–733, 2010.
[23] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD*, 2007.
[24] A. Maduko, K. Anyanwu, A. Sheth, and P. Schliekelman. Graph summaries for subgraph frequency estimation. In *ESWC*, 2008.
[25] S. Magliacane, A. Bozzon, and E. Della Valle. Efficient execution of top-k SPARQL queries. In *ISWC*, 2012.
[26] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung. Efficient top-k aggregation of ranked inputs. *TODS*, 32, 2007.
[27] D. Martinenghi and M. Tagliasacchi. Cost-Aware Rank Join with Random and Sorted Access. *TKDE*, 24(12):2143–2155, 2012.
[28] S. Michel, P. Triantafillou, and G. Weikum. KLEE: a framework for distributed top-k query algorithms. In *VLDB*, 2005.
[29] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data. In *ISWC*, 2011.
[30] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting Incremental Join Queries on Ranked Inputs. In *VLDB*, 2001.
[31] T. Neumann and G. Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *ICDE*, 2011.
[32] T. Neumann and G. Weikum. Scalable join processing on very large RDF graphs. In *SIGMOD*, 2009.
[33] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP²Bench: A SPARQL Performance Benchmark. In *ICDE*, 2009.
[34] K. Schnaitter and N. Polyzotis. Evaluating rank joins with optimal cost. In *PODS*, 2008.
[35] M. Shmueli-Scheuer, C. Li, Y. Mass, H. Roitman, R. Schenkel, and G. Weikum. Best-Effort Top-k Query Processing Under Budgetary Constraints. In *ICDE*, 2009.
[36] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *WWW*, 2008.
[37] A. Telang, C. Li, and S. Chakravarthy. One Size Does Not Fit All: Toward User- and Query-Dependent Ranking for Web Databases. *TKDE*, 24(9):1671–1685, 2012.
[38] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB*, 2004.
[39] A. Wagner, T. T. Duc, G. Ladwig, A. Harth, and R. Studer. Top-k linked data query processing. In *ESWC*, 2012.

8. APPENDIX

In this section, we present the query load that was used during our experiments. Queries for the SP² benchmark are based on [33], while the DBPSB benchmark queries are generated from seed queries in [29]. All queries are given in RDF N3[3] notation.

Listing 1: Prefixes used for SP² and DBPSB queries.

```
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:
  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dc:
  <http://purl.org/dc/elements/1.1/> .
@prefix dcterms:
  <http://purl.org/dc/terms/> .
@prefix xs:
  <http://www.w3.org/2001/XMLSchema#> .
@prefix bench:
  <http://localhost/vocabulary/bench/> .
@prefix foaf:
  <http://xmlns.com/foaf/0.1/> .
@prefix swrc:
  <http://swrc.ontoware.org/ontology#> .
@prefix dbpedia:
  <http://dbpedia.org/ontology/> .
@prefix dbpediaprop:
  <http://dbpedia.org/property/> .
@prefix dbpediares:
  <http://dbpedia.org/resource/> .
@prefix skos:
  <http://www.w3.org/2004/02/skos/core> .
@prefix yago:
  <http://dbpedia.org/class/yago/> .
```

Listing 2: Queries for SP² benchmark [33].

[3] http://www.w3.org/TeamSubmission/n3/

```
### 1
?journal dc:title "Journal 1 (1940)"^^xs:string .
?journal dcterms:issued ?yr .
?journal rdf:type bench:Journal .

### 2
?inproc dcterms:partOf ?proc .
?inproc bench:booktitle ?booktitle .
?inproc swrc:pages ?page .
?inproc dc:title ?title .
?inproc rdfs:seeAlso ?ee .
?inproc foaf:homepage ?url .
?inproc dcterms:issued ?yr .
?inproc dc:creator ?author .
?inproc rdf:type bench:Inproceedings .

### 3
?article rdf:type bench:Article .
?article swrc:pages ?value .

### 4
?article rdf:type bench:Article .
?article swrc:month ?value .

### 5
?article rdf:type bench:Article .
?article swrc:isbn ?value  .

### 6
?article1 dc:creator ?author1 .
?author1 foaf:name ?name1 .
?article1 swrc:journal ?journal .
?article2 swrc:journal ?journal .
?article2 dc:creator ?author2 .
?author2 foaf:name ?name2 .
?article1 rdf:type bench:Article .
?article2 rdf:type bench:Article .

### 7
?article dc:creator ?person .
?person foaf:name ?name .
?person2 foaf:name ?name .
?inproc dc:creator ?person2 .
?article rdf:type bench:Article .
?inproc rdf:type bench:Inproceedings .

### 8
?document dcterms:issued ?yr .
?document dc:creator ?author .
?author foaf:name ?name .
?document rdf:type ?class .
?class rdfs:subClassOf foaf:Document .

### 9
?doc dc:title ?title .
?bag2 ?member2 ?doc .
?doc2 dcterms:references ?bag2 .
?doc rdf:type ?class .
?class rdfs:subClassOf foaf:Document .

### 10
?erdoes foaf:name "Paul Erdoes"^^xs:string .
?document dc:creator ?erdoes .
?document dc:creator ?author .
?document2 dc:creator ?author .
?document2 dc:creator ?author2 .
?author2 foaf:name ?name .
?erdoes rdf:type foaf:Person .

### 11
?person rdf:type foaf:Person .
?subject ?predicate ?person .

### 12
?article dc:creator ?person1 .
?person1 foaf:name ?name .
?person2 foaf:name ?name .
?inproc dc:creator ?person2 .
?inproc rdf:type bench:Inproceedings .
?article rdf:type bench:Article .

### 13
?erdoes foaf:name "Paul Erdoes"^^xs:string .
?document dc:creator ?erdoes .
?document dc:creator ?author .
?document2 dc:creator ?author .
?document2 dc:creator ?author2 .
?author2 foaf:name ?name .
?erdoes rdf:type foaf:Person .
```

Listing 3: Queries for DBPSB benchmark [29].

```
### 1
?var5 rdf:type dbpedia:Person .
?var5 foaf:page ?var8 .
?var5 dbpedia:thumbnail ?var4 .
?var5 rdfs:label "Thaksin Shinawatra"@nn .

### 2
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label
        "\u0420\u0438\u0448\u0435,
         \u0428\u0430\u0440\u043B\u044C"@ru .
?var5 foaf:page ?var8 .

### 3
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label
                "Amadeo, quinto
                 Duque de Aosta"@es .
?var5 foaf:page ?var8 .

### 4
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label "Godeberta"@en .
?var5 foaf:page ?var8 .

### 5
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label "Thaksin Shinawatra"@nl .
?var5 foaf:page ?var8 .

### 6
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label
                "\u827E\u9A30\u00B7
                 \u4F0A\u683C\u8A00"@zh .
?var5 foaf:page ?var8 .

### 7
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label "Vlad\u00EDmir Karpets"@es .
?var5 foaf:page ?var8 .

### 8
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label "Daniel Pearl"@en .
?var5 foaf:page ?var8 .
```

```
### 9
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label
       "\u30DE\u30EA\u30FC\u30FB\u30EB
        \u30A4\u30FC\u30BA\u30FB\u30C9\u30EB
        \u30EC\u30A2\u30F3"@ja .
?var5 foaf:page ?var8 .

### 10
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label "Walter Hodge"@en .
?var5 foaf:page ?var8 .

### 11
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label "Damian Wayne"@en .
?var5 foaf:page ?var8 .

### 12
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label
        "\u0417\u0430\u043B
         \u0435\u0432\u0441\u043A
         \u0438\u0439, \u041A\u0430\u0437
         \u0438\u043C\u0435\u0436"@ru .
?var5 foaf:page ?var8 .

### 13
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label
        "\u0413\u0438
         \u043B\u0430\u0443\u0440\u0438,
         \u041D\u0438\u043A\u043E\u043B
         \u043E\u0437 \u0417\u0443\u0440
         \u0430\u0431\u043E\u0432\u0438
         \u0447"@ru .
?var5 foaf:page ?var8 .

### 14
?var5dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label "Francis Atterbury"@en .
?var5 foaf:page ?var8 .

### 15
?var5 dbpedia:thumbnail ?var4 .
?var5 rdf:type dbpedia:Person .
?var5 rdfs:label "Damian Wayne"@es .
?var5 foaf:page ?var8 .

### 16
?var4 dbpediaprop:birthPlace
                  "Vigny, Val d'Oise"@en .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 17
?var4 dbpediaprop:birthPlace
                  "Salisbury, England"@en .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 18
?var4 dbpediaprop:birthPlace
         "Bailey in the city of Durham"@en .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 19
?var4 dbpediaprop:birthPlace
        "Vasilievskaya,
         Tambov Governorate,"@en .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 20
?var4 dbpediaprop:birthPlace
        dbpediares:Waltham%2C_Massachusetts .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 21
?var4 dbpediaprop:birthPlace
                  dbpediares:Valencia%2C_Spain .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 22
?var4 dbpediaprop:birthPlace
        dbpediares:Halifax%2C_West_Yorkshire .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 23
?var4 dbpediaprop:birthPlace dbpediares:Sucre .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 24
?var4 dbpediaprop:birthPlace
                  dbpediares:L%C3%BAcar .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 25
?var4 dbpediaprop:birthPlace
                  dbpediares:%C3%89tampes .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 26
?var4 dbpediaprop:birthPlace
        dbpediares:Montgomery_County%2C_Maryland
            .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 27
?var4 dbpediaprop:birthPlace
                  "Berkeley, Gloucestershire"@en .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 28
?var4 dbpediaprop:birthPlace
                  dbpediares:Papal_States .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .
```

```
### 29
?var4 dbpediaprop:birthPlace
                dbpediares:City_of_London .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 30
?var4 dbpediaprop:birthPlace
                "Houghton, Norfolk, England"@en .
?var4 dbpedia:birthDate ?var6 .
?var4 foaf:name ?var8 .
?var4 dbpedia:deathDate ?var10 .

### 31
?var4 rdfs:label "(372) Palma"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 32
?var4 rdfs:label "(11554) Asios"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 33
?var4 rdfs:label "(3080) Moisseiev"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 34
?var4 rdfs:label "(1273) Helma"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 35
?var4 rdfs:label
        "(119878) 2002 CY224"@en .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 36
?var4 rdfs:label
        "039A\u578B\u6F5C
         \u6C34\u8266"@ja .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 37
?var4 rdfs:label
        "(4444) \u042D
         \u0448\u0435\u0440"@ru .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 38
?var4 rdfs:label
        "(3834) Zappafrank"@es .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 39
?var4 rdfs:label "(2612) Kathryn"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 40
?var4 rdfs:label "(290) Bruna"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 41
?var4 rdfs:label "(438) Zeuxo"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 42
?var4 rdfs:label "!X\u00F3\u00F5"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 43
?var4 rdfs:label "(1083) Salvia"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 44
?var4 rdfs:label
        "(1296) Andr\u00E9e"@de .
?var3 skos:broader ?var4 .
?var3 rdfs:label ?var6 .

### 45
?var1 rdf:type yago:ChristianLGBTPeople .
?var1 foaf:givenName ?var2 .

### 46
?var1 rdf:type yago:DefJamRecordingsArtists .
?var1 foaf:givenName ?var2 .

### 47
?var1 rdf:type yago:IndianFilmActors .
?var1 foaf:givenName ?var2 .

### 48
?var1 rdf:type yago:EnglishKeyboardists .
?var1 foaf:givenName ?var2 .

### 49
?var1 rdf:type yago:GuitarPlayers .
?var1 foaf:givenName ?var2 .

### 50
?var1 rdf:type yago:FilipinoFemaleModels .
?var1 foaf:givenName ?var2 .

### 51
?var1 rdf:type yago:BluesBrothers .
?var1 foaf:givenName ?var2 .

### 52
?var1 rdf:type yago:AmericanSongwriters .
?var1 foaf:givenName ?var2 .

### 53
?var1 rdf:type yago:FrenchJazzViolinists .
?var1 foaf:givenName ?var2 .

### 54
?var1 rdf:type yago:EnglishJazzComposers .
?var1 foaf:givenName ?var2 .

### 55
?var1 rdf:type yago:HarveyMuddCollegeAlumni .
?var1 foaf:givenName ?var2 .

### 56
?var1 rdf:type yago:Bassist109842629 .
?var1 foaf:givenName ?var2 .

### 57
?var1 rdf:type yago:Curate109983572 .
?var1 foaf:givenName ?var2 .

### 58
?var1 rdf:type yago:GreekFemaleModels .
```

```
?var1 foaf:givenName ?var2 .

### 59
?var1 rdf:type yago:FilipinoReligiousLeaders .
?var1 foaf:givenName ?var2 .

### 60
?var4 skos:subject
        dbpediares:Category:1004_deaths .
?var4 foaf:name ?var6 .

### 61
?var4 skos:subject
        dbpediares:Category:
        11th_century_in_England .
?var4 foaf:name ?var6 .

### 62
?var4 skos:subject
        dbpediares:Category:1067_deaths .
?var4 foaf:name ?var6 .

### 63
?var4 skos:subject
        dbpediares:Category:1107_births .
?var4 foaf:name ?var6 .

### 64
?var4 skos:subject
        dbpediares:Category:%C5%A0koda_trams .
?var4 foaf:name ?var6 .

### 65
?var4 skos:subject
        dbpediares:Category:
        %C3%81guilas_Cibae%C3%B1as_players .
?var4 foaf:name ?var6 .

### 66
?var4 skos:subject
        dbpediares:Category:1255_births .
?var4 foaf:name ?var6 .

### 67
?var4 skos:subject
        dbpediares:Category:0s_BC_births .
?var4 foaf:name ?var6 .

### 68
?var4 skos:subject
        dbpediares:Category:
        1130_disestablishments .
?var4 foaf:name ?var6 .

### 69
?var4 skos:subject
        dbpediares:Category:
        .32_S%26W_Long_firearms .
?var4 foaf:name ?var6 .

### 70
?var4 skos:subject
        dbpediares:Category:1009_deaths .
?var4 foaf:name ?var6 .

### 71
?var4 skos:subject
        dbpediares:Category:1144_deaths .
?var4 foaf:name ?var6 .

### 72
?var4 skos:subject
        dbpediares:Category:1239_deaths .
?var4 foaf:name ?var6 .

### 73
?var4 skos:subject
        dbpediares:Category:1070s_deaths .
?var4 foaf:name ?var6 .

### 74
?var4 skos:subject
        dbpediares:Category:1105 .
?var4 foaf:name ?var6 .

### 75
?var3 dbpedia:influenced
        dbpediares:Ram%C3%B3n_Emeterio_Betances .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 76
?var3 dbpedia:influenced dbpediares:Rob_Corddry .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 77
?var3 dbpedia:influenced
        dbpediares:Parakrama_Niriella .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 78
?var3 dbpedia:influenced
        dbpediares:Alexander_VI .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 79
?var3 dbpedia:influenced dbpediares:Iqbal .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 80
?var3 dbpedia:influenced
        dbpediares:Al-Maqrizi .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 81
?var3 dbpedia:influenced
        dbpediares:Clarence_Irving_Lewis .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 82
?var3 dbpedia:influenced
        dbpediares:Ibn_Khaleel .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 83
?var3 dbpedia:influenced
        dbpediares:David_Friedl%C3%A4nder .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 84
?var3 dbpedia:influenced
        dbpediares:John_Warnock .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 85
?var3 dbpedia:influenced
        dbpediares:Vladimir_Lenin .
```

```
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 86
?var3 dbpedia:influenced
        dbpediares:Niall_McLaren .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 87
?var3 dbpedia:influenced
        dbpediares:David_J._Farber .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 88
?var3 dbpedia:influenced
        dbpediares:Fran_Lebowitz .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 89
?var3 dbpedia:influenced
        dbpediares:Kathleen_Raine .
?var3 foaf:page ?var4 .
?var3 rdfs:label ?var6 .

### 90
?var0 rdfs:label "The Subtle Knife"@en .
?var0 rdf:type ?var1 .

### 91
?var0 rdfs:label "Patrioter"@sv .
?var0 rdf:type ?var1 .

### 92
?var0 rdfs:label "Scar Tissue (libro)"@es .
?var0 rdf:type ?var1 .

### 93
?var0 rdfs:label
        "Jason Bournes ultimatum"@nn .
?var0 rdf:type ?var1 .

### 94
?var0 rdfs:label "Jane Eyre"@fr .
?var0 rdf:type ?var1 .

### 95
?var0 rdfs:label
        "Gone with the Wind (livro)"@pt .
?var0 rdf:type ?var1 .

### 96
?var0 rdfs:label "Alkumets\u00E4"@fi .
?var0 rdf:type ?var1 .

### 97
?var0 rdfs:label
        "Flight from the Dark"@en .
?var0 rdf:type ?var1 .

### 98
?var0 rdfs:label "Mongol Empire"@en .
?var0 rdf:type ?var1 .

### 99
?var0 rdfs:label "Kultahattu"@fi .
?var0 rdf:type ?var1 .

### 100
?var0 rdfs:label
        "Aseiden k\u00E4ytt\u00F6"@fi .
?var0 rdf:type ?var1 .

### 101
?var0 rdfs:label "Marrow (novel)"@en .
?var0 rdf:type ?var1 .

### 102
?var0 rdfs:label "The Acid House"@en .
?var0 rdf:type ?var1 .

### 103
?var0 rdfs:label "\u6B63\u4E49\u8BBA"@zh .
?var0 rdf:type ?var1 .

### 104
?var0 rdfs:label "Dawn of the Dragons"@en .
?var0 rdf:type ?var1 .

### 105
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label
        "Ab\u016B l-Hasan Ban\u012Bsadr"@de .
?var2 foaf:page ?var4 .

### 106
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label
        "Abdul Rahman of Negeri Sembilan"@en .
?var2 foaf:page ?var4 .

### 107
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "A.W. Farwick"@en .
?var2 foaf:page ?var4 .

### 108
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "Abdullah G\u00FCl"@sv .
?var2 foaf:page ?var4 .

### 109
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "Aaron Pe\u00F1a"@en .
?var2 foaf:page ?var4 .

### 110
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "Abby Lockhart"@fr .
?var2 foaf:page ?var4 .

### 111
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "Abd al-Latif"@en .
?var2 foaf:page ?var4 .

### 112
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "Abdel Halim Khaddam"@fr .
?var2 foaf:page ?var4 .

### 113
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "Abdur Rahman Khan"@fr .
?var2 foaf:page ?var4 .

### 114
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label
        "A\u0142\u0142a Kudriawcewa"@pl .
?var2 foaf:page ?var4 .

### 115
?var2 rdf:type dbpedia:Person .
```

```
?var2 rdfs:label "Aaron Raper"@en .
?var2 foaf:page ?var4 .

### 116
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "A.L. Williams"@en .
?var2 foaf:page ?var4 .

### 117
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "Abdul Kadir Khan"@sv .
?var2 foaf:page ?var4 .

### 118
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "A. J. Pierzynski"@en .
?var2 foaf:page ?var4 .

### 119
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label "Abdullah Ahmad Badawi"@pl .
?var2 foaf:page ?var4 .

### 120
?var2 rdf:type dbpedia:Person .
?var2 rdfs:label
        "Abdelbaset Ali Mohmed Al Megrahi"@en .
?var2 foaf:page ?var4 .
```