Keyword Search Over Probabilistic RDF Graphs

Xiang Lian, Lei Chen, *Member, IEEE*, and Zi Huang

Abstract—In many real applications, RDF (Resource Description Framework) has been widely used as a W3C standard to describe data in the Semantic Web. In practice, RDF data may often suffer from the unreliability of their data sources, and exhibit errors or inconsistencies. In this paper, we model such unreliable RDF data by probabilistic RDF graphs, and study an important problem, *keyword search query over probabilistic RDF graphs* (namely, the pg-KWS query). To retrieve meaningful keyword search answers, we design the score rankings for subgraph answers specific for RDF data. Furthermore, we propose effective pruning methods (via offline pre-computed score bounds and probabilistic threshold) to quickly filter out false alarms. We construct an index over the pre-computed data for RDF, and present an efficient query answering approach through the index. Extensive experiments have been conducted to verify the effectiveness and efficiency of our proposed approaches.

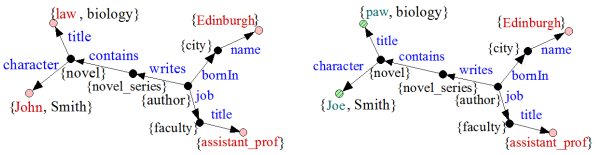
Index Terms—Probabilistic RDF graph, keyword search, pg-KWS.

1 Introduction

Resource Description Framework (RDF) is a W3C standard to represent resource information in the Semantic Web [1]. Nowadays, in many real applications, the RDF format has been widely used to model semantic data, in the form of triples (*subject*, *predicate*, *object*) (or (S, P, O) for short). Equivalently, a collection of RDF triples can be represented by a directed graph [1], [13], in which vertices are labeled by subjects (S) or objects (O), and edges (connecting from S to O) are labeled by predicates (P).

For example, we use an RDF triple ($\langle$author$\rangle$, $\langle$writes$\rangle$,$\langle$novel_series$\rangle$) to describe a case that an author writes a novel series. Equivalently, we can transform this triple to a directed edge (labeled by "writes") that connects from vertex "author" to vertex "novel_series", as shown in RDF Graph g_1 of Fig. 1(a). Similarly, a triple ($\langle$author$\rangle$, $\langle$bornIn$\rangle$,$\langle$city$\rangle$) can be converted into an edge (labeled by "bornIn") pointing from vertex "author" to vertex "city".

In the literature [22], [16], [25], [44], the keyword search over graphs has drawn much attention from the database community due to many applications (e.g., chemical data analysis and social network mining). In these applications, the underlying data are often represented by graph structures, in which vertices/edges are associated with keywords. In the RDF graph of Fig. 1(a), each vertex is often associated with one or multiple keywords (extracted from the label or URI of the vertex), for example, the vertex that represents the character of the novel has 2 keywords {"John", "Smith"}.



(a) Graph g_1 from data source A (b) Graph g_2 from data source B

Fig. 1. RDF graphs from different data sources (keywords: "John", "law", "Edinburgh", and "assistant_prof").

The keyword search over such data graphs is quite useful, due to its ease of use by non-experts. That is, the keyword search techniques do not require users to master the domain knowledge of the data schema (e.g., graph structures and keywords), and meanwhile they can easily return satisfactory query results by only specifying a few enquiry keywords (rather than a detailed query graph).

The keyword search over RDF graphs is also useful in real applications such as searching the Semantic Web with query keywords. Different from general graphs, in practice, RDF graphs contain RDF resources following the RDF schema, and their vertices are often associated with (imprecise) keyword information. Further, queries on general graphs include the subgraph matching and keyword search queries [22], [16], which do not have to consider RDF properties (e.g., RDF resources or schema). In contrast, RDF has its own standard language, SPARQL, which requires the knowledge of RDF graph structure and labels. Moreover, keyword search queries over RDF graphs [25], [44], [14], [12] need to consider properties of RDF graphs (e.g., RDF resources), which is the focus of our problem.

During the data extraction/integration in the Semantic Web, RDF data often contain errors or inconsistencies [15], [19]. For example, in the data extraction application, we apply information extraction (IE) methods, and extract RDF data from unstructured texts on the Web. The resulting RDF data may incur incorrectness, due to the inaccuracy of IE

- *X. Lian is with the Department of Computer Science, University of Texas - Pan American, Edinburg, TX 78539, USA, Email: lianx@utpa.edu.*
- *L. Chen is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong, China, Email: leichen@cse.ust.hk.*
- *Z. Huang is with the School of Information Technology and Electrical Engineering, the University of Queensland, Australia, Email: huang@itee.uq.edu.au.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2365791, IEEE Transactions on Knowledge and Data Engineering

techniques. Moreover, with different IE methods, extraction results may exhibit inconsistencies (e.g., with conflicting keywords for the same RDF resources).

Due to the unreliability of data sources (e.g., data expirations), when we integrate RDF graphs from different sources [10], [11], keyword information of the same RDF entity may conflict with one another. For example, Figs. 1(a) and 1(b) show 2 RDF graphs g_1 and g_2, from data sources A and B, respectively. Although most portions of the two graphs are the same, there exist some different vertex keywords. That is, the "novel" entity has title either "law" or "paw", and character either "John" or "Joe".

To resolve inconsistencies in RDF data, we treat RDF graph as a probabilistic graph [15], [19], [45], [46]. In particular, keywords of vertices in probabilistic RDF graphs are assigned with (conditional) probabilities, indicating the confidences that keywords are (conditionally) true in reality, which can be derived from the accuracy of IE techniques or inferred from the reliability of different data sources. One example of real probabilistic RDF graphs is the YAGO data set [42], which contains probabilistic RDF data, integrated from Wikipedia and WordNet. In this data set, RDF triples are associated with probabilities, indicating the confidences of facts (due to the unreliability of data sources). These RDF triples can be equivalently transformed to the representation of probabilistic RDF graphs.

As in the previous example of Fig. 1, we can integrate Graphs g_1 and g_2 from data sources A and B, respectively, by merging the same RDF entities (e.g., graph structures and keywords) from the two data sources. As a result, the integrated probabilistic RDF graph is illustrated in Fig. 2, where the conflicting keywords of vertices are associated with probabilities (indicating the confidences that vertices take the keywords). That is, the title of the novel has keyword "law" with probability 0.7, and "paw" with probability 0.3. Similarly, the character in the novel is "John" with probability 0.5, "Joe" with probability 0.4, and an unknown (wildcard or NULL) keyword other than "John" and "Joe" with probability 0.1 ($= 1 - 0.5 - 0.4$). Here, the probabilities can be learnt/inferred from statistics of historical data or reliability of data sources [10].

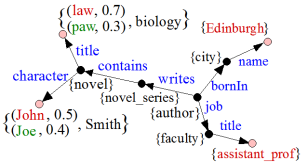
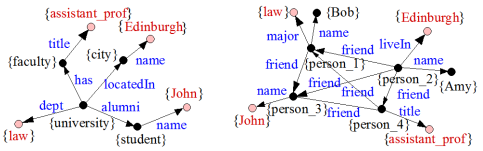


Fig. 2. Probabilistic RDF graph, $g_{1,2}$.

Thus, this paper will tackle the keyword search problem on probabilistic RDF graphs (namely, pg-KWS). That is, we want to retrieve probabilistic RDF subgraphs, which contain query keywords and have both high existence probabilities and ranks. Specifically, there are many real applications of the pg-KWS query. As an example, in the application of the job recruitment, a project manager may want to identify a team of engineers, who have close relationships and qualified skills (such as C++, UML, SQL, etc.) to assign tasks, from social networks (e.g., probabilistic graphs integrated from networks such as Facebook and LinkedIn). In this case, we can treat the integrated networks as probabilistic graphs, and issue a pg-KWS query over probabilistic graphs. That is, the project manager (user) can specify a set of query keywords, including the required skills, and retrieve the pg-KWS query answers from the integrated probabilistic graphs with high confidence. As a result, the returned pg-KWS subgraphs would contain candidates to form the project team, and their close relationships (i.e., edges) can be used as the evidence or basis for project manager to decide and assign tasks (since friends in the team may well collaborate with each other). Moreover, the returned probabilistic subgraphs can be used for further analysis. For example, with knowledge bases about relationships between skills and task capabilities, given the joint probability of the observed skills (keywords such as Java and SQL) inferred from probabilistic subgraphs, the project manager can compute probabilities that team members can accomplish a task (e.g., writing a Java application that connects to the database).

It is important yet challenging to return meaningful answers (with high ranks) from probabilistic RDF graphs, especially in real applications like the Semantic Web. For example, in Figs. 1(a) and 3(a), both RDF graphs, g_1 and g_3 (resp.), contain 4 keywords "John", "law", "Edinburgh", and "assistant_prof", and it is desirable to rank them with reasonable measures (keyword ranking score), which will be proposed in this paper specific for RDF graphs.



(a) Graph g_3 (b) Graph g_4

Fig. 3. RDF data graphs g_3 and g_4 (keywords: "John", "law", "Edinburgh", and "assistant_prof").

Under the *possible worlds* semantics [9], probabilistic RDF graphs can have exponential number of possible instances (i.e., possible worlds) that can appear in the real world. Thus, the straightforward method to consider keyword search for every subgraph and possible world would incur high cost, and it is infeasible in terms of the time complexity. Inspired by this, in this paper, we will investigate effective pruning methods via pre-computations to greatly rule out false alarms of the pg-KWS results. In addition, we will construct the index, and design an efficient approach to process pg-KWS queries.

Note that, different from subgraph matching queries (or other queries as well) in probabilistic RDF graphs [30], keyword search over graphs can retrieve query results without the necessity of specifying a query graph structure. While existing works [16], [20], [22] only studied keyword search on deterministic (general) graphs, to our best knowledge, no prior works studied the pg-KWS problem on probabilistic RDF graphs. Thus, previous techniques cannot be applied to solve our pg-KWS problem directly.

In this paper, we make the following contributions:

1) We propose the pg-KWS problem in probabilistic RDF graphs with appropriate ranks in Section 2.

2) We design effective pruning strategies to reduce the pg-KWS search space in Section 3.
3) We propose an efficient approach to process pg-KWS queries via pre-computed data in Section 4.
4) We conduct extensive experiments to show the pg-KWS query performance in Section 5.

Section 6 reviews related works on certain/probabilistic RDF and keyword search. Section 7 concludes this paper.

2 Problem Definition

Please refer to Table 2 in supplementary materials for symbols and their descriptions.

2.1 Data Model for Probabilistic RDF Graphs

Probabilistic RDF Graphs: A probabilistic RDF graph database $\mathcal{G}$ consists of probabilistic RDF graphs G, which are modeled by Bayesian networks [46].

Definition 2.1: (Probabilistic RDF Graph) A probabilistic RDF graph G is a triple $\langle V(G), E(G), S(G)\rangle$:

- $V(G)$ is a set of vertices v_i, each associated with a set of possible keywords (denoted as $k(v_i)$);
- $E(G)$ is a set of directed edges e_{ij} from v_i to v_j, associated with labels (each denoted as $l(e_{ij})$);
- $S(G)$ is a set of *conditional probability tables* (CPTs) [46] that contain conditional probabilities $T(k(v_i)|k(pa(v_i)))$, associated with vertices $v_i \in V(G)$, where $pa(v_i)$ is the set of v_i's parent vertices. ■

Specifically, in Definition 2.1, $V(G)$ is a set of vertices with keywords extracted from *subjects*/*objects* in RDF triples, whereas $E(G)$ is a set of directed edges with *predicates* as labels. The model of probabilistic RDF graphs follows that of Bayesian networks, which are directed acyclic graphs and are general to model correlations of keywords in probabilistic graphs. Each vertex v_i has a random variable (or a composite variable) for one or multiple keyword variables. That is, if there is only one keyword variable associated with v_i, then this random variable is exactly this keyword variable; otherwise, vertex v_i has a composite variable of multiple keyword variables, which takes different value combinations of keyword variables (i.e., events). Different events of keyword combinations are mutually exclusive. The relationships among vertex keywords can be represented by CPTs, $T(k(v_i)|k(pa(v_i)))$, with conditional probabilities that vertex v_i take keywords $k(v_i)$, given keywords $k(v_j)$ of their parents $v_j \in pa(v_i)$. The classical model of Bayesian networks is widely used and accepted, and it is practical to describe data in real applications such as the data integration from social networks or Semantic Web, where the integrated graph data (e.g., keywords) are uncertain or inconsistent.

Note that, the data model of probabilistic RDF graph in Definition 2.1 is different from that in prior works such as [30] and [19]. Specifically, in our data model, each vertex is associated with one (composite) random variable that may take different value combinations of keyword variable(s), rather than uncertain labels [30] or considering edge existence probabilities in RDF graphs [19].

In this paper, we consider uncertain keywords in vertices only, and certain labels in edges. In the case where edge labels are uncertain (denoted by CPTs), we can easily extend our solution to the one on a transformed probabilistic RDF graph. That is, we can add a new (dummy) vertex to each edge in probabilistic RDF graph, and assign CPTs of uncertain edge keywords to this new vertex. This way, we can transform probabilistic RDF graph with uncertain vertex/edge keywords to the one with uncertain keywords in vertices only, to which we apply our proposed approaches.

We also assume that RDF schema, Σ, is available [35], [42], [1], [49], which holds on many real RDF data [42]. Thus, we can identify RDF entities from probabilistic RDF graph, which are real-world objects (e.g., persons/novels).

Possible Worlds of Probabilistic RDF Graphs: In probabilistic databases [9], we usually consider the *possible worlds* semantics, where each possible world is a materialized instance of the database that can appear in reality. We define possible worlds of probabilistic RDF graphs below.

Definition 2.2: (Possible World of a Probabilistic RDF Graph [30]) A possible world, $pw(G)$, of a probabilistic RDF graph G is a graph instance where each vertex $v_i \in V(G)$ is assigned with a keyword value $l(v_i)$. ■

Intuitively, a possible keyword assignment to each vertex can result in a possible world of probabilistic RDF graph in Definition 2.2, whose existence probability is given by multiplying conditional probabilities of keywords in CPTs.

2.2 Probabilistic RDF Keyword Search Queries

2.2.1 The r-Radius Graph

Recently, EASE [25] has been proposed to answer keyword search queries on certain graphs. Keyword search answers in EASE are related to the notion of r-radius graph below.

Definition 2.3: (r-Radius Graph [25], g_c) Given a certain graph G_c, an r-radius graph, g_c, is a subgraph of G_c, centered at node $v \in V(G_c)$ and with radius r (i.e., any vertex $u \in V(g_c)$ is within r distance from v). ■

Intuitively, the r-radius graph defined in Definition 2.3 is an answer unit (i.e., a subgraph) to the keyword search query. Within r-radius graph g_c, each vertex u has the shortest path distance (which can be computed by Dijkstra algorithm in the graph) to center vertex v smaller than radius r. Here, the radius r is used to control the small size of keyword search answer graphs. This is because large and complex answer subgraphs are not so meaningful and relevant to keyword search queries, and users may be frustrated by the returned subgraphs with large size [25].

In this paper, we use the r-radius graph, g, as the smallest answer unit to be returned. However, we consider its probabilistic version, namely *probabilistic r-radius graph*, which is a probabilistic subgraph of probabilistic RDF graph G, where each vertex is associated with one (composite) random variable that represents one or multiple keyword variables. Note that, due to value combinations of multiple keyword variables, probabilistic r-radius graph g can be also considered as a compact representation of possible worlds. Each possible world $pw(g)$ corresponds to

a possible keyword valuation/assignment of the composite variable to each vertex of subgraph g, and has the appearance probability in the real world (given by multiplying conditional probabilities of CPTs).

To extract probabilistic r-radius graphs g from G, we can start from each vertex $v \in V(G)$, and perform a breadth-first search with radius r. All vertices/edges (including uncertain/certain keywords) that are traversed form a probabilistic r-radius graph g centered at v and with radius r.

In the subsequent discussion, we will use terms, subgraph and probabilistic r-radius graph, interchangeably with the same meaning when the context is clear.

2.2.2 Ranking Scores for RDF Keyword Search

Traditional keyword search problem uses different measures for ranking keyword query answers, for example, IR-based score [33], structural score [22], [16], or a combination of both score types [25]. In this paper, we consider ranking semantics for keyword search over RDF graphs, and rank a subgraph g_c by a 2-dimensional score vector $\langle feature(g_c),\ entropy(g_c)\rangle$, where $feature(g_c)$ is the *feature score* of g_c, and $entropy(g_c)$ is the *entropy score*. While the feature score uses a hybrid of structural and IR scores that are similar to existing ranking scores [25], the entropy score can reflect our preference to keyword distributions over RDF subgraph entities.

We first consider two scores, $feature(g_c)$ and $entropy(g_c)$, in a certain subgraph g_c below. Later, we will generalize it to probabilistic RDF graph.

Feature Score: The first dimension, $feature(g_c)$, in the 2D ranking score vector considers both structural and IR features of subgraph g_c, the same as that in [25].

Definition 2.4: (Feature Score, $feature(g_c)$ [25]) Given q query keywords k_1, k_2, ..., and k_q, and a (certain) subgraph g_c, the feature score, $feature(g_c)$, of g_c is:

$$feature(g_c) = \sum_{1\le i,j\le q} \sum_{k_i\in n_i \wedge k_j \in n_j} (score_{struc}(k_i,k_j,g_c) \cdot (score_{IR}(k_i,g_c) + score_{IR}(k_j,g_c))), \quad (1)$$

where n_i (or n_j) is a node that contains keyword k_i (or k_j), and the structure-based and IR-based scores, $score_{struc}(k_i, k_j, g_c)$ and $score_{IR}(k_i, g_c)$ are defined below.

$$score_{struc}(k_i,k_j,g_c) = \frac{1}{|\{n_i\}\cup\{n_j\}|} \cdot \frac{1}{(dist(n_i,n_j)+1)^2}, \quad (2)$$

$$score_{IR}(k_i,g_c) = \frac{ntf(k_i,g_c)\cdot idf(k_i,g_c)}{ndl_{g_c}} \quad (3)$$

$$= \frac{(1+ln(1+ln(1+tf_{k_i,g_c})))\cdot ln\frac{N+1}{N_{k_i}+1}}{(1-s)+s\cdot\frac{dl_{g_c}}{avg\_dl}},$$

where $dist(n_i, n_j)$ is the shortest path length between nodes n_i and n_j (ignoring edge directions), s is a constant that is typically set to 0.2 similar to [31], [25], tf_{k_i,g_c} is the term frequency of keyword k_i in g_c (i.e., the number of keywords k_i in subgraph g_c), $ntf(k_i, g_c)$ is the normalized term frequency given in Eq. (3) [31], [25], N is the total number of subgraphs, N_{k_i} is the number of subgraphs with keyword k_i, dl_{g_c} is the total number of keywords in g_c, avg_dl is the average number of keywords for all subgraphs, and ndl_{g_c} is the normalized number of keywords in subgraph g_c (i.e., $(1-s)+s\cdot\frac{dl_{g_c}}{avg\_dl}$). ■

In Definition 2.4, the feature score $feature(g_c)$ is a combined IR- and structure-based score. Specifically, in Eq. (1), $feature(g_c)$ is given by summing up the multiplication of structural and IR scores, for all pairwise query keywords k_i and k_j. Here, the structural score w.r.t. a keyword pair (k_i, k_j) is calculated by summing up $score_{struc}(k_i, k_j, g_c)$ in Eq. (2) over all pairs of nodes (n_i, n_j) (containing k_i and k_j). In Eq. (2), $score_{struc}(k_i, k_j, g_c)$ is inversely proportional to the (squared) shortest path lengths between nodes n_i and n_j. The intuition is that, smaller distance between two query keywords shows stronger relationship between nodes, and thus their structural score should be higher. Moreover, the IR-based score is given by Eq. (3) w.r.t. TF and IDF, whose basic idea is to treat each subgraph (i.e., r-radius graph) as a document, and each label (keyword) of vertices as a term. This way, TF, IDF and document length can be computed and used for producing the IR score in Eq. (3), similar to [31], [25].

A subgraph g_c with larger feature score $feature(g_c)$ is expected to have higher rank, in terms of the structural and IR features of query keywords in g_c. Please refer to Appendix E1 in supplementary materials for an example of the feature score. As will be discussed later in Section 2.2.3, we will generalize this feature score to the probabilistic scenario under possible worlds semantics.

Entropy Score: Next, we will define the second dimension of the 2D score vector, entropy score $entropy(g_c)$, which considers the property specific for RDF graphs with entities.

The notion of entropy (or Shannon entropy) was originally used in the *information theory* to measure the uncertainty in a random variable. For example, if a variable may take two possible values (0 or 1) with equal probabilities, then its possible values are the most uncertain (since we cannot tell which value has higher chance to be the actual value), which can be reflected by the entropy measure.

In this paper, we will utilize this entropy concept to propose a metric that can indicate our preference to RDF keyword search results in probabilistic RDF graphs. This metric is related to keyword distributions among RDF entities in (probabilistic) r-radius graphs. In particular, our basic idea is that we want to retrieve those subgraphs, in which (1) query keywords should be distributed in as few RDF entities as possible, and (2) most query keywords should be located in a few RDF entities, rather than evenly scattered in many RDF entities.

Intuitively, similar to the entropy notion in the information theory, we can treat each RDF entity that contains query keywords (in our keyword search result) as a possible value of the variable, and the percentage of query keywords located in this RDF entity as the probability that the variable takes this value. This way, we can define an entropy score, $entropy(g_c)$, for a subgraph g_c, which is defined below.

Definition 2.5: (Entropy Score, $entropy(g_c)$) Assume that query keywords are in m RDF entities in a subgraph g_c. Let f_i be the number of query keywords in the i-th RDF entity. Then, we define the entropy score $entropy(g_c)$ as:

$$entropy(g_c) = -\sum_{i=1}^{m}\left(\frac{f_i}{\sum_{j=1}^{m} f_j}\cdot log\frac{f_i}{\sum_{j=1}^{m} f_j}\right). \quad ■ \quad (4)$$

In Eq. (4), the fraction, $\frac{f_i}{\sum_{j=1}^{m} f_j}$, is the percentage of query keywords that are located in the i-th RDF entity in the subgraph g_c. The formula of entropy score $entropy(g_c)$ is given by the Shannon entropy. Note that, different from the feature score with "the higher, the better" semantics, the entropy score follows "the smaller, the better" semantics. That is, a subgraph g_c with smaller entropy score, $entropy(g_c)$, would be ranked higher.

As an example, given a set of query keywords and two subgraphs g and g', assume that subgraph g contains 2 RDF entities with evenly distributed query keywords, and g' has query keywords equally distributed in 3 RDF entities. In this case, subgraph g has the entropy score: $entropy(g) = log(2)$, which is smaller than that of subgraph g' (i.e., $entropy(g) = log(3)$). This reflects our preference that we want to obtain those subgraphs with query keywords residing in fewer RDF entities.

Note that, different from the feature score that encodes structural distances among keywords in the RDF graph, the entropy score considers the keyword distribution among RDF entities in the RDF graph. While the structural information in the feature score does not take into account RDF entities, the entropy score is a different measure for our preference to the keyword search answer. Later on, we will use both scores as orthogonal dimensions to rank RDF graphs. Please refer to Appendix E2 in supplementary materials for an example of the entropy score.

Dominance of 2D Score Vectors: According to feature and entropy scores, we can rank RDF subgraphs to retrieve keyword search answers. Since feature and entropy scores are two different and orthogonal scores (i.e., one for structural/IR feature, and the other one for summarizing the keyword distribution in RDF entities), it is not trivial how to aggregate (e.g., sum up or multiple) them together. Thus, in this paper, we will consider these two scores as separate dimensions, and define the dominance relationship between two subgraphs on both dimensions.

In particular, each RDF subgraph (keyword-search answer candidates) can be represented by a data point (i.e., 2D score vector) in a feature-and-entropy score space. We can define the dominance relationships between two subgraphs g_c and g'_c in the 2D score space below.

Definition 2.6: (Dominance Relationship, $g'_c \prec g_c$) Given two subgraphs g_c and g'_c, we say g'_c *dominates* g_c (denoted as $g'_c \prec g_c$), if both conditions below hold:
(1) $feature(g'_c) \geq feature(g_c)$ and $entropy(g'_c) \leq entropy(g_c)$;
(2) $feature(g'_c) > feature(g_c)$ or $entropy(g'_c) < entropy(g_c)$. ■

In the 2D score space, attributes of each subgraph are dynamically computed with respect to ad-hoc query keywords. Intuitively, the dominance relationship indicates our preference of r-radius graphs w.r.t. keywords in terms of feature and entropy scores. Here, our dominance definition is a bit different from the one in traditional skyline query [7], which takes "the smaller the better" semantics. That is, in Definition 2.6, the first dimension, $feature(\cdot)$, follows the "the larger the better semantics", whereas the second one, $entropy(\cdot)$, follows the traditional semantics. Please refer to Appendix E3 in supplementary materials for an example of the dominance in the 2D score space.

2.2.3 The pg-KWS Problem

In the sequel, we formally define the problem of the keyword search query in the probabilistic RDF graph. As mentioned in Section 2.2.1, the *r-radius graph* [25] is the basic subgraph unit that we want to obtain for the keyword search in RDF graphs. The r-radius graph centers at some vertex in the graph and has the breadth-first traversal depth (radius) not greater than r. Specific to our problem where the RDF graph is probabilistic, we need to consider the ranking score under the possible worlds semantics [9].

Definition 2.7: (Keyword Search Query over Probabilistic RDF Graph, pg-KWS) Given a probabilistic RDF graph G, a set of q query keywords k_1, k_2, ..., k_q, and a probabilistic threshold $\alpha \in [0, 1)$, a keyword search query over probabilistic RDF graph G (pg-KWS) retrieves r-radius graphs $g \subseteq G$ such that g contains all the query keywords, and $\langle feature(g), entropy(g)\rangle$ is not *dominated* by that of other r-radius subgraphs g' with probability $P(g)$ greater than threshold α. Here, we have:

$$P(g) = Pr\{g \text{ is not dominated by } g' \mid \forall\, g'\} = \sum_{\forall pw(g) \in \mathcal{PW}(g)} Pr\{pw(g)\} \cdot Pr\left\{\bigwedge_{\forall g'} g' \not\prec pw(g)\right\} \quad (5)$$

where $\mathcal{PW}(g)$ is a set containing all possible worlds of g, and $Pr\{g' \not\prec pw(g)\} = \sum_{\forall pw(g'), pw(g') \not\prec pw(g)} Pr\{pw(g')\}$. ■

Definition 2.7 obtains probabilistic r-radius graphs g, not dominated by other graphs g' in the 2D score space, with the dominance probability greater than α (note that, threshold-based probabilistic queries are often used in literature such as [8], [43], [21], [29], [37]). Specifically, as shown in Eq. (5), the confidence $P(g)$ is given by the probability that g is not dominated by other graphs g' in possible worlds (i.e., $pw(g') \not\prec pw(g)$). In other words, $P(g)$ is the probability that g is not worse than other subgraphs g' (in terms of feature/entropy dimensions), by considering different pairs of possible worlds $(pw(g'), pw(g))$, and summing up existence probabilities of possible world pairs where $pw(g')$ does not dominate $pw(g)$. Here, since each possible world, $pw(g')$ or $pw(g)$, of probabilistic r-radius graph can be considered as a certain graph (associated with its appearance probability in the real world), the dominance relationship between $pw(g')$ and $pw(g)$ can be given by Definition 2.6.

Intuitively, pg-KWS query answers are expected to have high feature scores (with high structure/IR scores) and low entropy scores (with good keyword distributions in RDF entities), which are not worse than other non-answers (in terms of feature/entropy scores) and thus preferred by users. Please refer to Appendix E4 in supplementary materials for an example of the pg-KWS problem.

The pg-KWS problem is similar to computing probabilistic skylines [37] over 2D uncertain score vectors. However, previous work [37] considered probabilistic skyline

Procedure pg-KWS_Framework {
Input: *a probabilistic RDF graph database* $\mathcal{G}$
Output: *an index* $\mathcal{I}$ *over* $\mathcal{G}$
(1) build a multidimensional index $\mathcal{I}$ offline on probabilistic RDF graphs // *indexing phase*
Input: *an index* $\mathcal{I}$ *over probabilistic RDF graph database* $\mathcal{G}$, *a set of query keywords* k_1, k_2, ..., k_q, *and a probabilistic threshold* α
Output: *the answer to pg-KWS query*
(2) for each online pg-KWS query // *pruning phase*
(3) traverse index $\mathcal{I}$ by applying the score bound and probabilistic pruning
(4) refine candidates in the set S_{cand} and return pg-KWS answers // *refinement phase*
}

Fig. 4. General framework for pg-KWS processing.

on static uncertain data. In contrast, attributes (i.e., feature/entropy scores) in our problem should be dynamically computed w.r.t. online query keywords. Furthermore, each probabilistic subgraph g has exponential number of possible worlds, and their scores in possible worlds are also exponential, which is more challenging than [37] (with linear sample size). Thus, we cannot efficiently solve our pg-KWS problem by directly using previous techniques [37], which inspires us to design efficient processing approaches.

2.3 General Framework

Fig. 4 illustrates the general framework for our pg-KWS query processing. In particular, the framework consists of three phases, indexing, pruning, and refinement phases. In the indexing phase, we will offline extract probabilistic r-radius graphs from the probabilistic RDF graph, and pre-compute data for each graph. Then, we construct an index over these pre-computed data for probabilistic r-radius graphs, which will be used later for online pruning and pg-KWS query answering (line 1). Given any pg-KWS query, the second pruning phase traverses the index, and meanwhile applies pruning methods to quickly rule out false alarms (i.e., those subgraphs that cannot be pg-KWS query answers; lines 2-3). In particular, we will propose two pruning strategies, *score bound pruning* and *probabilistic pruning*, which utilize score bounds or probabilistic threshold, respectively, to enable the pruning. After the index traversal, we can obtain a candidate set S_{cand}. Finally, in the refinement phase (line 4), we refine candidates in S_{cand} by checking the condition in Eq. (5), and return the actual pg-KWS answers (line 4).

3 Pruning Strategies

3.1 Score Bound Pruning

In this subsection, we present the basic idea of our proposed pruning methods via score bounds. In particular, for a probabilistic r-radius graph g, we will offline pre-compute data for all keywords or keyword pairs in g. Then, during the pg-KWS query processing, we can utilize these pre-computed data to efficiently calculate lower/upper bounds of feature and entropy scores in probabilistic r-radius graph. Intuitively, the score bounds we obtain are minimum /maximum possible scores for all possible worlds of the probabilistic r-radius graph, and we can use them to facilitate effective filtering of false alarms.

Specifically, we denote lower and upper bounds of the feature score, $feature(g)$, in Eq. (1) as $lb\_feature(g)$ and $ub\_feature(g)$, respectively. Moreover, let lower and upper bounds of the entropy score, $entropy(g)$, in Eq. (4) be $lb\_entropy(g)$ and $ub\_entropy(g)$, respectively. From the dominance relationship between subgraphs in 2D score space, we can have the pruning lemma below.

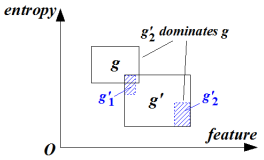


Fig. 5. Probabilistic pruning (g'_2 is dominating g).

Lemma 3.1: (Score Bound Pruning) Given two graphs g and g', if $ub\_feature(g)$<$lb\_feature(g')$ and $lb\_entropy(g)$>$ub\_entropy(g')$ hold, g can be pruned.

Proof. Please refer to Appendix P1 in supplementary materials. □

Next, the remaining issue is how to obtain the bounds for feature and entropy scores.

Computation of Feature Score Bounds: From Eq. (1), to obtain the feature bounds $lb\_feature(g)$ and $ub\_feature(g)$, we can, instead, compute the bounds for IR- and structure-based scores, $score_{struc}(\cdot)$ and $score_{IR}(\cdot)$, in Eqs. (2) and (3), respectively.

Bounds of Structure-Based Score: For any keyword pair $\langle k_i, k_j \rangle$ in an r-radius graph g, denote lower/upper bounds of its structure-based score as $lb\_score_{struc}(k_i, k_j, g)$ and $ub\_score_{struc}\ (k_i, k_j, g)$, respectively. Since g is a probabilistic graph, we offline pre-compute its bounds defined below, by considering all possible worlds $pw(g)$.

$$lb\_score_{struc}(k_i, k_j, g) = \min_{\forall pw(g)} score_{struc}(k_i, k_j, pw(g)), \quad (6)$$

$$ub\_score_{struc}(k_i, k_j, g) = \max_{\forall pw(g)} score_{struc}(k_i, k_j, pw(g)), \quad (7)$$

Note that, in Eqs. (6) and (7), the structural score bounds need to explore possible worlds, which can be achieved by joining CPTs in subgraph g and calculating structural scores for the corresponding label (keyword) assignment.

Bounds of IR-Based Score: Similarly, for the IR-based score, we can also offline pre-compute its bounds:

$$lb\_score_{IR}(k_i, g) = \min_{\forall pw(g)} score_{IR}(k_i, pw(g)), \quad (8)$$

$$ub\_score_{IR}(k_i, g) = \max_{\forall pw(g)} score_{IR}(k_i, pw(g)), \quad (9)$$

where $score_{IR}(k_i, pw(g))$ is given by Eq. (3).

Therefore, for any online query keywords k_1, k_2, ..., and k_q, we can dynamically compute lower/upper bounds of a feature score in subgragh g as follows.

$$lb\_feature(g) = \sum_{1 \le i,j \le q} \sum_{k_i \in n_i \wedge k_j \in n_j} (lb\_score_{struc}(k_i, k_j, g) \cdot (lb\_score_{IR}(k_i, g) + lb\_score_{IR}(k_j, g))) \quad (10)$$

$$ub\_feature(g) = \sum_{1 \le i,j \le q} \sum_{k_i \in n_i \wedge k_j \in n_j} (ub\_score_{struc}(k_i, k_j, g) \cdot (ub\_score_{IR}(k_i, g) + ub\_score_{IR}(k_j, g))) \quad (11)$$

Please refer to Appendix E5 in supplementary materials for an example of feature score bounds.

Computation of Entropy Score Bounds: Next, we aim to compute lower/upper bounds of an entropy score. Specifically, assume that f_i is the number of query keywords that are in (possible worlds) of the i-th RDF entity. Let $[lb\_f_i, ub\_f_i]$ be the bounds of f_i (i.e., $f_i \in [lb\_f_i, ub\_f_i]$).

Then, we can calculate the lower/upper bounds of the entropy score $entropy(g)$ in the following lemma.

Lemma 3.2: (Bounds of Entropy Score) Lower/upper bounds of the entropy score are given by:

$$lb\_entropy(g) = -\max_i \left\{ log \frac{ub\_f_i}{ub\_f_i + \sum_{\forall j \neq i, 1 \leq j \leq m} lb\_f_j} \right\} \quad (12)$$

$$ub\_entropy(g) = -\min_i \left\{ log \frac{lb\_f_i}{lb\_f_i + \sum_{\forall j \neq i, 1 \leq j \leq m} ub\_f_j} \right\} \quad (13)$$

Proof. Please refer to Appendix P2 in supplementary materials. □

Please refer to Appendix E6 in supplementary materials for an example of entropy score bounds.

3.2 Probabilistic Pruning

Up to now, we have discussed how to use score bounds (calculated from pre-computed scores for keywords or keyword pairs over all possible worlds) to prune those false alarms (subgraphs) that are definitely dominated by others. In order to further enhance the pruning power, in this subsection, we will propose a probabilistic pruning method, and filter out those false alarms whose score bounds are partially dominated by considering the probabilistic confidences.

As shown in the example of Fig. 5, uncertain bounds for subgraphs g and g' intersect with each other on both dimensions. In other words, not all possible worlds of subgraph g are dominated by that of subgraph g'. As a result, we cannot simply prune subgraph g, using our proposed score bound pruning method (as given in Lemma 3.1). In contrast, our probabilistic pruning method may prune subgraph g, if g is dominated by g' with probability $P(g)$ not greater than threshold α (Definition 2.7).

Probabilistic Pruning With Probability Upper Bounds: Specifically, our basic idea of the probabilistic pruning is to derive an upper bound of probability $P(g)$ with low cost, and then prune g with small probability upper bound (i.e., $\leq \alpha$). We give the lemma for probabilistic pruning below.

Lemma 3.3: (Probabilistic Pruning) Let $UB\_P(g)$ be an upper bound of probability $P(g)$ in Eq. (5). If it holds that $UB\_P(g) \leq \alpha$, then subgraph g can be safely pruned.

Proof. Please refer to Appendix P3 in supplementary materials. □

From Lemma 3.3, to enable probabilistic pruning, one important issue is to obtain an upper bound of the probability $P(g)$. We have the following lemma about this upper bound derived from Eq. (5).

Lemma 3.4: (Upper Bound of $P(g)$) Let g' be a probabilistic r-radius graph. Then, an upper bound, $UB\_P(g)$, of probability $P(g)$ in Eq. (5) can be given by:

$$UB\_P(g) = \sum_{\forall pw(g)} Pr\{pw(g)\} \cdot Pr\{g' \not\prec pw(g)\} \quad (14)$$

Proof. Please refer to Appendix P4 in supplementary materials. □

From Eq. (14) of Lemma 3.4, in order to obtain the upper bound $UB\_P(g)$ of probability $P(g)$ (as given in Eq. (5)), instead of checking all other probabilistic r-radius graphs, it is sufficient to consider just one single probabilistic r-radius graph g', and compute the probability $Pr\{g' \not\prec pw(g)\}$.

The Computation of Relaxed Probability Upper Bounds: Note, however, that the calculation of Eq. (14) still involves an exponential number of possible worlds for graphs g and g'. Thus, we aim to further reduce the computation cost by computing a relaxed probability upper bound $UB\_P(g)$ with respect to groups of possible worlds (i.e., without enumerating all possible worlds).

We illustrate our heuristics of probabilistic pruning via groups of possible worlds, by using the example of Fig. 5, where $\alpha = 0.5$. In Fig. 5, assume that score vectors $\langle feature(pw(g'), entropy(pw(g'))\rangle$ (for all possible worlds $pw(g')$) in the 2D space are located in two regions g'_1 and g'_2, with probabilities p_1 (= 0.3) and p_2 (= 0.7), respectively. Without loss of generality, we can divide all possible worlds of g' into two disjoint groups g'_1 and g'_2, each with appearance probability 0.3 or 0.7, respectively. As depicted in Fig. 5, it holds that $g'_2 \prec pw(g)$ (equivalently, $Pr\{g'_2 \not\prec pw(g)\} = 0$), and moreover, $Pr\{g'_1 \not\prec pw(g)\} \leq p_1 = 0.3$. Thus, in this case, we can derive a probability upper bound $UB\_P(g)$ from Eq. (14):

$$\begin{aligned} UB\_P(g) &= \sum_{\forall pw(g)} Pr\{pw(g)\}(Pr\{g'_1 \not\prec pw(g)\} + Pr\{g'_2 \not\prec pw(g)\}) \\ &\leq \sum_{\forall pw(g)} Pr\{pw(g)\} \cdot (p_1 + 0) = p_1 = 0.3. \end{aligned}$$

Therefore, p_1 is called a relaxed probability upper bound of $P(g)$. According to the probabilistic pruning in Lemma 3.3, since g has low confidence of not being dominated by other subgraphs such as g' (i.e., $p_1 = 0.3 \leq \alpha = 0.5$ holds), we can safely filter out subgraph g. This way, our probabilistic pruning method partitions possible worlds of graph g' into 2 groups, and derives a probability upper bound to rule out subgraph g.

In a general case, we can divide all possible worlds of each probabilistic r-radius graph g into w (> 1) disjoint groups, namely *possible world groups* (PWGs), denoted as $PWG_1(g)$, $PWG_2(g)$, ..., $PWG_w(g)$. For each PWG $PWG_i(g)$, we calculate lower/upper bounds of feature/entropy scores, and the summed existence probability, $PWG_i(g).p$, of all possible worlds in $PWG_i(g)$. Then, we summarize the general case of deriving the relaxed probability upper bound in the lemma below.

Lemma 3.5: (Relaxed Upper Bound of $P(g)$ in PWGs) A relaxed upper bound $UB\_P(g)$ of $P(g)$ in Eq. (5) is:

$$UB\_P(g) = \sum_i \sum_j PWG_i(g).p \cdot PWG_j(g').p \cdot \chi(PWG_j(g') \not\prec PWG_i(g)), \quad (15)$$

where $\chi(z) = 1$, if $z = true$, and $\chi(z) = 0$, otherwise.

Proof. Please refer to Appendix P5 in supplementary materials. □

Please refer to Appendix E7 in supplementary materials for an example of probability upper bounds.

Note that, the partitioning of possible worlds for probabilistic r-radius graphs, as well as the pre-computations for score bounds of PWGs, can be processed offline. As mentioned in the example of Fig. 5, with respect to PWGs, our relaxed probability upper bound in Eq. (18) can be derived online and used for pruning false alarms (e.g., g) that cannot be pruned by score bound pruning method.

Optimization: We next design a cost model to guide the selection of a good partitioning strategy to divide possible

worlds of probabilistic subgraph g into w PWGs, which is critical for the probabilistic pruning.

Cost Model for Possible World Partitioning: Since different partitioning strategies result in different bounds of feature/entropy scores, it is important to choose a strategy that produces tight bounds and has high pruning power. To evaluate the goodness of a partitioning strategy, we use a measure, which is the (summed) difference between upper and lower score bounds for keyword pairs (or keywords).

Denote lower and upper bounds of the (normalized) feature score in a PWG, $PWG_i(g)$, as $lb\_f(PWG_i(g))$ and $ub\_f(PWG_i(g))$, respectively. Let lower/upper bounds of the (normalized) entropy score be $lb\_e(PWG_i(g))$ and $ub\_e(PWG_i(g))$, respectively. Moreover, let $\beta \in [0,1]$ balance the importance of tightness between feature and entropy scores.

We measure the tightness of bounds w.r.t. PWGs by:

$$Tight(g) = \beta \cdot \sum_{i=1}^{w} PWG_i(g).p \cdot (ub\_f(PWG_i(g)) - lb\_f(PWG_i(g)) \quad (16)$$
$$+(1-\beta) \cdot \sum_{i=1}^{w} PWG_i(g).p \cdot (ub\_e(PWG_i(g)) - lb\_e(PWG_i(g))$$

In Eq. (16), $Tight(g)$ is the summation of bounds for PWGs, weighted by existence probabilities of PWGs (i.e., bounds with higher existence probability tend to have more importance to prune false alarms) and parameter β. Intuitively, smaller $Tight(g)$ indicates tighter score bounds, which have less chance to overlap with others, and its corresponding PWGs are expected to achieve higher pruning power. Thus, our goal is to find a PWG partitioning strategy, which divides all possible worlds of a subgraph g into w PWGs, such that the target function $Tight(g)$ is minimized.

Challenges and Time Complexity: The challenges of our PWG partitioning problem are two-fold. First, the number of possible worlds itself is exponential, which corresponds to the size of join results among all CPTs in the subgraph g. Second, the number of possible partitioning strategies is also exponential (i.e., to the power of the number of possible worlds). Formally, the time complexity of enumerating all partitioning strategies is given by $O(w^{\prod_{\forall i}|CPT_i|})$, where $|CPT_i|$ is the number of tuples in CPT CPT_i.

PWG Partitioning Algorithm: Inspired by the complexity of finding a good PWG partitioning, we propose a heuristic-based algorithm, which obtains w PWGs of a probabilistic subgraph g with low cost.

Specifically, denote $\mathcal{PS}(i,j)$ as a partitioning strategy, $\{PWG_1, ..., PWG_j\}$, which indicates that we have accessed i CPTs so far, and divide all possible worlds into j PWGs, PWG_1, PWG_2, ..., and PWG_j. Let $\mathcal{PS}_{i,j}.cost$ be the total cost of partitioning strategy $\mathcal{PS}(i,j)$. We have:

$$\begin{aligned}\mathcal{PS}(i,j) &= argmin_{\mathcal{PS}(\cdot,\cdot)}(\mathcal{PS}(i,j).cost) \quad (17)\\ &= argmin_{\mathcal{PS}(\cdot,\cdot)} \min\{\mathcal{PS}(i-1,j).cost,\\ &\quad F(\mathcal{PS}(i-1,j-1), PWG_1, CPT_i).cost, ...,\\ &\quad F(\mathcal{PS}(i-1,j-1), PWG_{j-1}, CPT_i).cost\}.\end{aligned}$$

where $\mathcal{PS}(1,1) = \{PWG_1\}$ (all possible worlds in one PWG), and $\mathcal{PS}(1,j) = \{PWG_1, ..., PWG_j\}$ (j PWGs split on attributes of the first CPT we access).

Specifically, in Eq. (17), $\mathcal{PS}(i-1,j).cost$ is the cost of the partitioning strategy (given by Eq. (16)), where we perform the partitioning on (at most) $(i-1)$ CPTs we have accessed and obtain j partitions. Moreover, function $F(\mathcal{PS}(i-1,j-1), PWG_u, CPT_i)$ (for $1 \le u \le j-1$) removes the u-th PWG (i.e., PWG_u) from partitioning strategy $\mathcal{PS}(i-1,j-1)$, and divides PWG_u (with partitioning attributes in CPT_i) into two disjoint groups, PWG_{u1} and PWG_{u2}, such that the resulting $\mathcal{PS}(\cdot)$ cost is minimized. The output of function $F(\cdot)$ is a set of j PWGs, $\{PWG_1, ..., PWG_{u-1}, PWG_{u1}, PWG_{u2}, PWG_{u+1}, ..., PWG_{j-1}\}$, and $F(\cdot).cost$ is the cost of this partitioning strategy using Eq. (16).

Note that, in the recursive function of Eq. (17), we apply the heuristics that the partitioning strategy $\mathcal{PS}(i,j)$ can be obtained from $\mathcal{PS}(i-1,j)$ and the modified strategies of $\mathcal{PS}(i-1,j-1)$ by splitting PWG_u ($1 \le u \le j-1$) into two groups, which can save the computation cost. The split function in $F(\cdot)$ can apply a randomized approach to randomly obtain two partitions of approximate equal size. We execute the randomized algorithm for $iter$ rounds, and select the one with the lowest cost (as given in Eq. (16)) as the output of function $F(\cdot)$.

Given a subgraph g with $|V(g)|$ vertices (i.e., $|V(g)|$ CPTs), our PWG partitioning algorithm will iteratively obtain PWG partitioning strategy $\mathcal{PS}(i,j)$ in Eq. (17), as well as its corresponding cost $\mathcal{PS}(i,j).cost$. Specifically, the algorithm maintains a cost matrix, $cost\_matrix$, in which each element $cost\_matrix[i][j] = \mathcal{PS}(i,j).cost$ for $i \le j$. Initially, we start from the first row $cost\_matrix[1][.]$. Then, we repeatedly compute the i-th row by utilizing the values on the $(i-1)$-th row (via Eq. (17)).

Finally, to obtain the best PWG partitioning of g, we return w PWGs in entry $cost\_matrix[|V(g)|][w]$, which is corresponding to the PWG partitioning strategy, $\mathcal{PS}(|V(g)|, w)$. The returned partitioning strategy is expected to achieve high pruning power. The time complexity of the algorithm is given by $O(|V(g)| \cdot w \cdot C)$, where C is the cost to evaluate the cost model as given in Eq. (16).

4 PG-KWS Query Processing

4.1 Index Construction

We now discuss how to construct a tree index, $\mathcal{I}$, over probabilistic RDF graphs G for our pg-KWS problem. Specifically, as mentioned in Section 3, we will first offline extract all probabilistic r-radius graphs g (centered at each vertex and with radius r) from probabilistic RDF graph G, and then build the index on these graphs g. Since we need to apply pruning methods to reduce the search space during the index traversal, we will store some offline pre-computed data in this index, which can be used for calculating score or probability bounds to enable the pruning.

In particular, we build a tree index $\mathcal{I}$ over probabilistic r-radius graphs in a bottom-up manner. That is, starting from leaf nodes of the tree on the bottom level, we iteratively group nodes on the i-th level ($0 \le i < h$, for tree height h), and construct parent nodes on the $(i+1)$-th level, until finally only one root node, $root(\mathcal{I})$, is obtained. We illustrate the detailed structures in (non-)leaf nodes below.

Leaf Nodes: A leaf node e contains entries that correspond to probabilistic r-radius graphs g, respectively.

To enable the score bound pruning, for each graph (entry) g, we maintain the bounds of its feature/entropy scores (e.g., feature bounds of pairwise keywords, and counter bounds for each keyword). Similarly, for probabilistic pruning, we keep bitmaps or probabilities for w PWGs.

Specifically, for each probabilistic r-radius graph g in a leaf node e, we store w sets of score bounds for w PWGs, respectively, each of which includes:

- structural score bounds $[lb\_score_{struc}(k_i, k_j, g)$, $ub\_score_{struc}\ (k_i, k_j, g)]$, for keyword pairs (k_i, k_j);
- IR-based score bounds $[lb\_score_{IR}(k_i, g)$, $ub\_score_{IR}(k_i, g)]$, for keywords k_i;
- counter bounds $[lb\_f_i(k_i, g), ub\_f_i(k_i, g)]$ in RDF entities, for keywords k_i;
- bitmaps, $BITMAP(PWG)$, which stores the existence information of all possible keywords $l(v_i)$ in vertices $v_i \in V(g)$ in a PWG PWG; and
- the existence probability of the PWG.

The first 3 bounds above (i.e., for structural score, IR-based score, and counter) can be used for computing the feature/entropy score bounds, and apply the score bound pruning (Section 3.1). Further, for a PWG of graph g, we compute bitmaps $BITMAP\ (PWG)$ that record the existence of possible keywords in this PWG for keyword search, and the existence probability of the PWG used for probabilistic pruning. Note that, here the bitmap is a synopsis (bit vector), each of whose positions (bits), $BITMAP(PWG)[i]$, corresponds to a keyword, and takes value "1" if this keyword exists in PWG ("0", otherwise).

Let K_g be the number of distinct keywords in graph g. The total space cost for g (considering w PWGs) can be given by $O(w \cdot (2K_g^2 + 4K_g + K_g/8 + 1))$.

Non-Leaf Nodes: In a non-leaf node e on the $(i+1)$-th level, each entry e_x contains aggregates of probabilistic r-radius graphs under e_x (e.g., structural score bounds, IR-based score bounds, counter bounds, bitmaps, and PWG existence probabilities), and a pointer pointing to a child on the i-th level. Let g be any probabilistic r-radius graph (in leaf nodes) under e_x. The aggregates in entry e_x are:

- structural score bounds $[\min_{\forall g \in e_x}\{lb\_score_{struc}(k_i, k_j, g)\}$, $\max_{\forall g \in e_x}\{ub\_score_{struc}(k_i, k_j, g)\}]$;
- IR-based score bounds $[\min_{\forall g \in e_x}\{lb\_score_{IR}(k_i, g)\}$, $\max_{\forall g \in e_x}\{ub\_score_{IR}(k_i, g)\}]$;
- counter bounds $[\min_{\forall g \in e_x}\{lb\_f_i(k_i, g)\}$, $\max_{\forall g \in e_x}\{ub\_f_i(k_i, g)\}]$ in RDF entities; and
- bitmaps, $BITMAP(e_x) = \bigvee_{\forall g \in e_x} BITMAP(g)$;
- a pointer pointing to its child node.

Note that, the lower (upper) bounds above underestimate (or overestimate) score bounds, by considering bounds of all graphs under e_x. Different from leaf nodes, aggregates in non-leaf nodes summarize bounds or keyword existence information for all possible worlds of graphs (rather than w PWGs). Thus, we do not need to record existence probabilities of PWGs in non-leaf nodes. Moreover, entries in non-leaf nodes have additional pointers pointing to their children, which do not exist in leaf nodes.

Therefore, the total space cost for each entry e_x in a non-leaf node is given by $O(2K_x^2 + 4K_x + K_x/8 + 1)$, where K_x is the number of distinct keywords under node e_x.

The Index Construction: To build a hierarchical tree index, we start from all probabilistic r-radius graphs, and recursively group those graphs/nodes with small (summed) *hamming distance* among their pairwise bitmaps to build parent nodes on a higher level, where the hamming distance is defined as the number of distinct bits in 2 bitmaps of sizes B, that is, $Hamming\_dist(BITMAP(e_1), BITMAP(e_2)) = \sum_{i=1}^{B} |BITMAP(e_1)[i] - BITMAP(e_2)[i]|$. Intuitively, probabilistic r-radius graphs with similar keywords tend to have similar score bounds. This way, tree nodes on different levels can be iteratively built until a final root is obtained.

4.2 Pruning With Index Nodes

After we build a tree index over probabilistic r-radius graphs, we can answer pg-KWS queries by traversing the index from root to leaf nodes in a breadth-first manner. In this subsection, we will provide two pruning methods, score bound pruning and probabilistic pruning, for filtering out index nodes e. Note that, once we prune nodes e during the tree traversal, we can save the computation cost of checking all probabilistic r-radius graphs under e.

Let $ub\_feature(e)$ be the upper bound of feature scores for all graphs under node e, and $lb\_entropy(e)$ be the lower bound of entropy scores for all graphs under node e. We summarize the two pruning methods in lemmas below.

Lemma 4.1: (Score Bound Pruning With Nodes) Given a probabilistic r-radius graph candidate g' and a node e, if it holds that $ub\_feature(e) < lb\_feature(g')$ and $lb\_entropy(e) > ub\_entropy(g')$, node e can be pruned.

We can derive a probability upper bound for a node e:

Lemma 4.2: (Probability Upper Bound, $UB\_P(e)$, w.r.t. Node e) Given a graph candidate g' with w PWGs, probability $P(g)$ in Eq. (5) for $g \in e$ can be upper bounded by:

$$UB\_P(e) = \sum_j PWG_j(g').p \cdot \chi(PWG_j(g') \not\simeq e) \quad (18)$$

where $\chi(z) = 1$, if $z = true$, and $\chi(z) = 0$, otherwise.

Lemma 4.2 derives a probability upper bound, $UB\_P(e)$, of any graph under node e. Thus, similar to Lemma 3.3, we can utilize this upper bound to apply the probabilistic pruning method with respect to node e. That is, if it holds that $UB\_P(e) \leq \alpha$, then node e can be safely pruned.

4.3 Query Procedure

We illustrate the pseudo code of pg-KWS processing, namely procedure pg_KWS_Processing, in Fig. 6. In particular, our query procedure uses a heap to traverse index $\mathcal{I}$ over probabilistic RDF graph $\mathcal{G}$, and obtain probabilistic r-radius graphs g whose 2D scores are not dominated by others with probability greater than α. Please refer to the detailed discussion of pg-KWS query procedure in Q1 of supplementary materials.

Procedure pg_KWS_Processing {
Input: a probabilistic RDF graph database $\mathcal{G}$, an index $\mathcal{I}$ over $\mathcal{G}$, a set of q query keywords k_1, k_2, ..., k_q, and a probabilistic threshold α
Output: subgraphs g from $\mathcal{G}$ that contain keywords and satisfy Inequality (5)
(1) initialize a min-heap $\mathcal{H}$ accepting entries in the form (e, key)
(2) $S_{cand} = \emptyset$;
(3) insert $(root(\mathcal{I}), 0)$ into heap $\mathcal{H}$
(4) while $\mathcal{H}$ is not empty
(5) (e, key) = de-heap $\mathcal{H}$
(6) if e cannot be pruned by score bound/probabilistic pruning
(7) if e is a probabilistic r-radius graph g // Lemmas 4.1 and 3.3
(8) if PWGs of g cannot be pruned by probabilistic pruning
(9) add g to S_{cand} // Lemma 3.3
(10) if e is a leaf node
(11) for each probabilistic r-radius graph $g \in e$
(12) if g cannot be pruned by score bound/probabilistic pruning
(13) insert $(g, key(g))$ into heap $\mathcal{H}$ // Lemmas 3.1 and 3.3
(14) if e is a non-leaf node
(15) for each entry $e_x \in e$
(16) if e_x cannot be pruned by score bound/probabilistic pruning
(17) insert $(e_x, key(e_x))$ into heap $\mathcal{H}$ // Lemmas 4.1 and 3.3
(18) refine candidates in S_{cand} and report final pg-KWS query answers
}

Fig. 6. The pg-KWS query procedure.

5 Experimental Evaluation

In this section, we report the experimental results of our proposed approaches for answering pg-KWS queries on both real and synthetic data. For synthetic data, we first generate a schema graph, Σ, by producing N_e vertices that represent N_e RDF entities, and randomly connecting vertices via directed edges, where the averages of vertex out-degrees and in-degrees are set to 3 by default. Note that, here we consider a vertex (and its associated nodes) having at least one out-going edge as one RDF entity, which will be used for the calculation of the entropy score. After we generate the RDF schema graph Σ, we next create L instance graphs that follow this schema graph, where the default value of L is 100. Then, for each vertex v_i of instance graphs, we randomly generate m possible keywords $l(v_i)$ (each keyword is hashed to an integer within range $[1, 100]$), as well as conditional probability tables (CPTs) that depend on vertex keywords from its incoming edges, where m is randomly picked up within $[1, 5]$ by default. We tested two types of distributions for uncertain vertex keywords, Uniform and Skew ($Zipf$ distribution with skewness 0.8), and denote their corresponding data sets as $Uniform$ and $Skew$, respectively. For real data, YAGO data set [42] contains uncertainty, for example, in RDF data "directed_CheckedFactExtractor", facts (i.e., subjects/objects) are associated with confidences (i.e., probabilities). Thus, we can extract a probabilistic graph with N vertices from YAGO data, where each vertex is associated with a keyword variable that may take two possible keywords (one is the original label of the vertex and the other one is a dummy keyword that follows the distribution of keywords in the YAGO data). In order to test the case with multiple m (> 2) uncertain keywords per vertex v_i, we simulate the remaining $(m-1)$ keywords that follow the frequency distribution of keywords in YAGO, and obtain a probabilistic RDF graph. We also use the YAGO2 [17] data set which contains more complicated ontology than YAGO. The extraction of probabilistic RDF graphs for YAGO2 is similar to that for YAGO. To evaluate the pg-KWS query, we randomly pick up q query keywords from the keyword set of synthetic/real data.

TABLE 1
The experimental settings.

Parameters	Settings
α	$0.1, 0.2, \mathbf{0.5}, 0.8, 0.9$
q	$2, \mathbf{3}, 4, 5, 8, 10$
r	$2, \mathbf{3}, 4$
N	$10K, 20K, \mathbf{30K}, 40K, 50K, 100K, 1M$

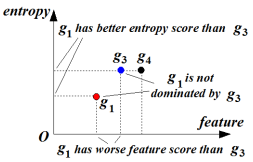


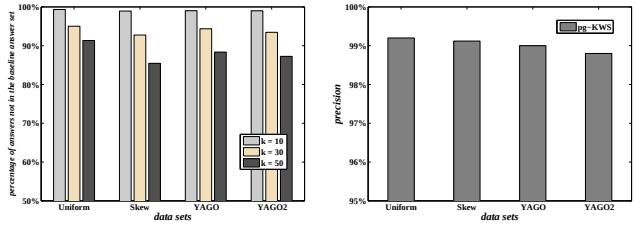
Fig. 7. The dominance in 2D score space.

To the best of our knowledge, there are no prior works that consider the keyword search over probabilistic RDF graphs. As mentioned in Section 2.2, our pg-KWS problem uses the ranking scores that consider both structural/IR features and entropies, which can reflect users' preference to keyword search answers over RDF graphs. In our experiments, we will compare the effectiveness of a *baseline method* (which solely considers the feature dimension) with that of our proposed measure.

Note that, previous techniques on uncertain query processing or the skyline computation cannot directly tackle our complex pg-KWS problem, which involves both keyword search queries (in a converted 2D score space) and probabilistic RDF graphs. Therefore, one straightforward method is illustrated as follows. For each probabilistic r-radius graph extracted from probabilistic RDF graph, we first calculate its 2D feature/entropy scores w.r.t. online query keywords. Then, we retrieve those probabilistic r-radius graphs that are not dominated by others in the 2D score space. Since query keywords are online specified, this method has to compute scores of probabilistic r-radius graphs for every query from scratch, and perform the dominance check for all graphs, which involves an exponential number of possible worlds and is very inefficient. In contrast, our pg-KWS approach (as mentioned in Fig. 6) applies effective pruning techniques to reduce the search space, and the number of subgraph candidates is much smaller than that of the straightforward method. Thus, our pg-KWS performance is by orders of magnitude better than that of the straightforward method. To clearly illustrate the trend of our pg-KWS approach, below, we will not plot curves for the straightforward method.

We evaluate the performance of our pg-KWS approach, in terms of the CPU time and I/O cost, where the CPU time is the time cost of accessing the index and applying pruning methods, and the I/O cost is the number of page accesses during the index traversal. Table 1 summarizes the experimental settings, where the numbers in bold font are default values. Below, for each set of experiments, we will vary one parameter at a time, while setting other parameters to their default values. All our experiments are conducted on a PC with Core(TM)2 Duo 3GHz CPU with 3G memory.

Effectiveness of pg-KWS: We test the effectiveness of our pg-KWS query by comparing with a baseline method [25] that obtains top-k subgraphs that have the highest feature



(a) (b)

Fig. 8. Effectiveness of pg-KWS. (a) The percentage of pg-KWS answers that are not in the answer set of the baseline method. (b) Precision of pg-KWS candidates after pruning vs. synthetic/real data sets.

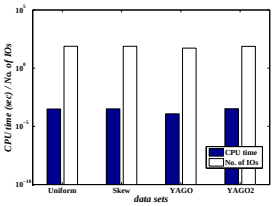


Fig. 9. Performance vs. real/synthetic data sets.

scores with nonzero probability.

We first manually inspect the case of certain graphs, g_1, g_3, and g_4 in Figs. 1(a), 3(a), and 3(b), respectively, to illustrate the effectiveness of our problem definition, where query keywords are "John", "law", "Edinburgh", and "assistant_prof". By computing and plotting the feature/entropy scores (defined in Definitions 2.4 and 2.5) of all the three graphs in the 2D score space, Fig. 7 shows that Graphs g_1 and g_4 are not dominated by other graphs. Thus, they are the answers to our keyword search query. Intuitively, Graph g_1 in Fig. 1(a) encodes the information of the novel, written by an author, which might be answers desired by users; moreover, Graph g_4 in Fig. 3(b) contains 4 people with different attributes (such as the name, job, living place, and major) which might be preferred by users who want to study their relationships (e.g., in the real application of social networks). From the semantics of our keyword-search-answer definition, Graph g_3 in Fig. 3(a) is dominated by g_4, and thus it is not the best answer that is preferred by users (due to the existence of g_4). Indeed, g_3 simply contains some common information in a university which is not quite useful to users (e.g., almost every university has a department of law and a faculty with the title of assistant professor). In contrast, if we apply the baseline method [25] that retrieves top-2 subgraphs (i.e., $k = 2$), then Graphs g_4 and g_3 with the highest feature scores will be returned. However, as we mentioned above, g_3 is not quite desirable by users; most importantly, the more meaningful Graph g_1 is not treated as the keyword search answer. Therefore, by our definition of scores in the 2D space, g_3 and g_4 can be successfully returned as answers, whereas the baseline method may miss some meaningful result.

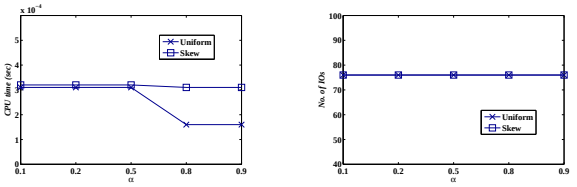
From the case study above, our pg-KWS approach can provide more meaningful answers that are not included by the baseline method. Next, we will evaluate the pg-KWS problem over probabilistic RDF graphs, and report the percentage of meaningful keyword-search-answers returned by our pg-KWS approach which do not appear in the top-k answer set of the baseline method [25] (note: here k is set to the number of pg-KWS answers). Fig. 8(a) shows such percentages on synthetic/real data sets, which indicates that about 85% $\sim$ 99% pg-KWS answers cannot be retrieved by the baseline method, where k varies from 10 to 30. This is because the baseline method only retrieves those subgraphs with high feature scores, while our pg-KWS definition may obtain additional subgraphs with lower feature scores but lower (better) entropy scores (which are also preferred by users). For larger k value such as 50, the baseline method returns more answers, which may include more pg-KWS answers. However, there are still 85% $\sim$ 91% pg-KWS answers that cannot be returned by the baseline. The experimental results confirm that our pg-KWS approach can return more useful answers, compared to the baseline.

Next, we evaluate the recall and precision of our pg-KWS pruning methods. We consider the returned pg-KWS query answers (i.e., probabilistic r-radius graphs) as the ground truth. The recall is defined as the number of pg-KWS query answers in the candidate set after pg-KWS pruning divided by the actual number of pg-KWS query answers. Since the pg-KWS approach applies pruning methods without introducing any false dismissals, the recall ratio of our pg-KWS approach is 100% (i.e., all pg-KWS answers are in the candidate set after pruning). Moreover, the precision is defined as the actual number of pg-KWS query answers divided by the number of pg-KWS candidates after applying the pruning. Fig. 8(b) shows the precisions of 4 real/synthetic data sets, which are high (i.e., all above 98%). Here, higher precisions indicate higher pruning power, and in turn the cost of refining those non-pg-KWS candidates (i.e., false positives) becomes lower. This confirms the effectiveness of our proposed pruning methods and the efficiency of the pg-KWS approach.

Performance vs. Real/Synthetic Data Sets: We also show the pg-KWS query efficiency on real/synthetic data in Fig. 9, with default settings. Our experiments show that our query answering approach can achieve low CPU time (i.e., about 3×10^{-4} second) and small numbers of I/Os (i.e., 70-80 I/Os), which indicates the effectiveness of our score bound and probabilistic pruning. Below, we will use synthetic data to test the robustness of our pg-KWS approach by varying different parameter values.

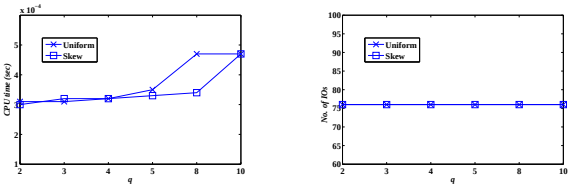
Performance vs. Probabilistic Threshold α: Next, we show the pg-KWS performance in Fig. 10 by varying α values, where other parameters have default values. In figures, when α increases from 0.1 to 0.9, the CPU time slightly decreases. This is mainly because of our probabilistic pruning (Section 3.2), which filters out false alarms g with probability $P(g) \leq \alpha$. Thus, larger α will give higher pruning power, and thus require lower CPU time (especially for large threshold α like 0.8 and 0.9). Furthermore, for different α, the I/O cost remains low (i.e., about 76), which shows the efficiency of our approach.

Performance vs. Number of Query Keywords q: Fig. 11 shows the effect of q on the pg-KWS performance, with default parameters. When we vary q from 2 to 10, the time cost increases. This is because higher costs are needed to check the keyword existence and online compute score



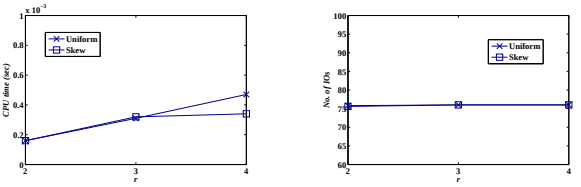
(a) CPU time (b) No. of I/Os

Fig. 10. The pg-KWS performance vs. α.



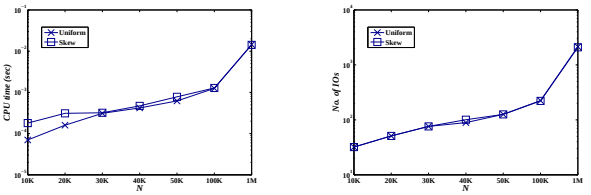
(a) CPU time (b) No. of I/Os

Fig. 11. The pg-KWS performance vs. q.



(a) CPU time (b) No. of I/Os

Fig. 12. The pg-KWS performance vs. r.



(a) CPU time (b) No. of I/Os

Fig. 13. The pg-KWS performance vs. N $(= |V(G)|)$.

bounds for the pruning. Thus, with larger q, the CPU time slightly increases. From the results, the CPU time remains low (i.e., below 5×10^{-4} second), and the I/O cost is small (i.e., < 80), which shows good scalability of our approach.

Performance vs. Radius of r-Radius Subgraph r: Fig. 12 presents the pg-KWS query performance, by varying parameter r from 2 to 4, where default values are used for other parameters. In figures, larger r would give higher time cost. This is because larger r-radius graphs would incur higher cost of checking keywords (pairs) for the pruning. Nonetheless, our pruning methods w.r.t. pre-computed data are still effective. Thus, in figures, the CPU time smoothly increases for larger r, and the I/O cost remains low.

Performance vs. RDF Graph Size N: We also report the pg-KWS query performance with different graph sizes N in Fig. 13, where the number of vertices N varies from $10K$ to $1M$, and other parameters are set to their default values. When graph data size N increases, both the CPU time and I/O cost also smoothly increase, that is, linearly w.r.t. N. Even for $N = 1M$, the CPU time still remains low (about 0.014 sec), and the number of I/Os is small. Such low costs are due to the effect of our proposed pruning methods.

For more experiments about the pruning power and index construction time/index size vs. N, please refer to R1 and R2, respectively, in supplementary materials.

6 Related Work

Certain RDF Data: RDF has been widely used in applications of the Semantic Web, and it has several data models, for example, triple store [47], [35], [4], C-store (or column store) [41], [2], [38], property tables [49], [48], and graphs [3], [44]. SPARQL query is a standard SQL-like query language for the RDF data. To issue a SPARQL query, users have to know the schema of RDF data, including the name or value of subject, predicate, or object. Previous works usually studied the problem of improving the efficiency of the SPARQL query answering, by using the indexing or join operations. Examples of RDF systems include C-Store [2], RDF-3X [34], MonetDB [38], and Hexastore [47].

The existing works [44], [14], [12] on keyword search over RDF graphs often assume that, labels of vertices in RDF graphs are certain. Specifically, Tran et al. [44] considered several structural/IR-based ranking score functions such as the path length, popularity score, and keyword matching score. Elbassuoni et al. [14], [12] used statistical language models and ranked subgraphs by estimating the query likelihood (or the probability to generate the query graph given data graphs' statistics such as TF/IDF). These scores are general ranking functions that are applicable to general certain graphs. In contrast, our pg-KWS problem considers not only the structural/IR score (i.e., feature score), but also the orthogonal entropy score, specific for RDF graphs. Moreover, while previous methods [44], [14], [12] are only proposed for certain RDF graphs, our pg-KWS problem is in a different context, probabilistic RDF graphs with correlations. Due to the data uncertainty, we have to handle an exponential number of possible worlds [9] over probabilistic RDF graphs. Thus, with distinct ranking scores and data model, our pg-KWS problem cannot directly borrow previous methods on certain graphs.

Probabilistic RDF Databases: There are some existing works on modeling probabilistic RDF data. For example, Fukushige [15] modeled probabilistic RDF data by the Bayesian network. However, this work only gave a probabilistic RDF data model by extending the vocabulary of RDF data. In this paper, we model our probabilistic RDF graph by Bayesian network, but focus on a different aspect (i.e., the efficiency of keyword search) from [15].

Udrea et al. [45] considered probabilistic RDF data that are represented by a graph with probabilistic edges (i.e., each edge is associated with an existence probability). Similarly, Huang and Liu [19] modeled probabilistic RDF data by the probabilistic database, which assumes that edges (tuples) in the graph (database) have existence probabilities. They differ from the data model in our pg-KWS problem, which contains probabilistic vertex keywords (rather than probabilistic edges). Additionally, the work in [19] considered SPARQL queries, instead of the keyword search. Lian and Chen [30] studied the subgraph matching problem in probabilistic RDF graphs, which obtains probabilistic subgraphs that match with a query graph (converted from SPARQL) with high probabilities, which thus also assumes that users have the knowledge of RDF schema. The proposed techniques explored pruning methods with graph structures and matching probabilities.

In contrast, our pg-KWS problem does not require users to know RDF schema, and considers a different query type (i.e., the keyword search). Thus, we need to design specific pruning techniques w.r.t. feature/entropy scores and dominance probabilities, and the approaches proposed in [30] cannot be directly applied to our pg-KWS problem.

Probabilistic/Uncertain Databases: In probabilistic/uncertain databases, the data uncertainty can be classified into two categories [39], *attribute uncertainty* and *tuple uncertainty*. Uncertain databases [8] consider the attribute uncertainty, in which uncertain objects are modeled by uncertainty regions. The tuple uncertainty is captured by probabilistic databases that contain relational tuples associated with existence probabilities.

Query answering on probabilistic/uncertain data considers the *possible worlds* semantics [9]. That is, a possible world is a materialized instance of the probabilistic database that may appear in the real world. A probabilistic query is equivalent to first answering such a query in all possible worlds individually, and then combining (aggregating) query answers returned from these possible worlds.

In the literature, various probabilistic queries over uncertain data have been proposed, including *probabilistic range query* (PRQ) [8], [43], [21], *nearest neighbor* (PNN) [8], [5], *reverse nearest neighbor* (PRNN) [29], *skyline* (PSQ) [37], *reverse skyline* (PRSQ) [28], and *similarity join* (PSJ) [24], [32]. All these works mainly focused on queries over probabilistic/uncertain spatial/relational data, rather than RDF graphs studied in this paper. Thus, we cannot borrow previous techniques, and have to design our own (pruning) methods. Further, top-k queries [40], [27] were studied in probabilistic databases to retrieve tuples with high ranks and probabilities. In contrast, our pg-KWS problem ranks subgraphs (instead of single tuples) containing keywords.

Probabilistic XML: Kimelfeld and Sagiv [23] considered the *twig query* over probabilistic XML trees, which is a different query from our pg-KWS query on probabilistic graphs with any correlations. Similarly, Li et al. [26] studied the keyword search over probabilistic XML trees, which ranks search results by the Smallest Lowest Common Ancestor (SLCA) semantics specific for XML trees (rather than probabilistic RDF graphs). Moreover, for the data model, prior works only considered 2 types of probabilistic nodes in probabilistic XML, with independent (IDD) or mutually exclusive (MUX) children. In contrast, our model can describe any correlations in probabilistic RDF graphs. Thus, we cannot borrow methods in probabilistic XML to solve our pg-KWS problem with a different ranking function, data structure, and uncertainty model.

Keyword Search: Keyword search over graphs has recently been very hot due to its ease of use by users in real applications [6], [22], [16], [25], [44]. Basically, these works return connected keywords in a graph with high ranks, and aim to improve the query efficiency.

Prior works have extensively studied how to return meaningful subgraphs that contain query keywords, and different semantics [20], [18], [31], [25] have been proposed, for example, using the total length of the shortest paths among nodes containing keywords, and/or IR scores related to TF/IDF ranking by keywords (i.e., the term frequency and inverse document frequency). One common metric [22], [16] considers the graph structure such as a tree with a root r and other leaf/non-leaf nodes containing query keywords. The ranking score is defined as the summation of path lengths from a root node to all nodes containing query keywords. Bicer et al. [6] adopted the relevance-based ranking for documents over structured data, and returns tree structures that are compact and contain all query keywords (in contrast, our pg-KWS problem returns subgraphs with semantics of features and keyword distributions). The IR-based score [18], [25] (i.e., $(TF \cdot IDF)/DL$) is also defined to rank subgraphs, where TF is the term frequency, IDF is the inverse document frequency, and DL is the average size of subgraphs. The IR scores of subgraphs can have different variants such as [18], [25].

For the keyword search over graphs, BANKS [20] presented a backward search method, which starts from nodes that contain query keywords, and traverses the graph through backward links until subgraphs with high scores are obtained. Later, BANKS-II [22] proposed the bidirectional search, which uses the pre-computed distance between keywords and nodes, and obtains the bounds of ranking scores to enable fast pruning and retrieval. BLINKS [16] designed a two-level indexing structure to speed up the bidirectional keyword search in graphs. EASE [25] proposed a keyword search approach which returns the so-called r-radius Steiner graphs with the highest scores w.r.t. structural/IR scores. These works assume the keyword search in certain graphs. In contrast, our work explores special properties of RDF graphs (with probabilistic RDF entities), and design user preferable scores and efficient query answering approaches.

7 Conclusions

In this paper, we formulate and tackle an important problem of keyword search over probabilistic RDF graphs, called pg-KWS queries. We propose effective pruning methods via offline pre-computed score bounds and probabilistic threshold to quickly filter out false alarms. Moreover, we construct an index for the pre-computed data for RDF, and present an efficient query answering approach. Extensive experiments have been conducted to verify the effectiveness and efficiency of our proposed approaches.

References

[1] W3C: Resource description framework (RDF). In *http://www.w3.org/RDF/*.

[2] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, 2007.

[3] R. Angles and C. Gutierrez. Querying RDF data from a graph database perspective. In *The Semantic Web: Research and Applications*, 2005.

[4] M. Atre, V. Chaoji, M. J. Zaki, and J. A. Hendler. Matrix "bit" loaded: a scalable lightweight join query processor for rdf data. In *WWW*, 2010.

[5] G. Beskales, M. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors inuncertain databases. In *Proc. 34th Int. Conf. on Very Large Data Bases*, pages 326–339, 2008.

[6] V. Bicer, T. Tran, and R. Nedkov. Ranking support for keyword search on structured data using relevance models. In *CIKM*, 2011.

[7] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[8] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 551–562, 2003.

[9] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4), 2007.

[10] X. L. Dong, L. Berti-Equille, and D. Srivastava. Integrating conflicting data: The role of source dependence. *PVLDB*, 2(1), 2009.

[11] X. L. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. *VLDBJ*, 18(2), 2009.

[12] S. Elbassuoni and R. Blanco. Keyword search over RDF graphs. In *CIKM*, 2011.

[13] S. Elbassuoni, M. Ramanath, R. Schenkel, M. Sydow, and G. Weikum. Language-model-based ranking for queries on RDF-graphs. In *CIKM*, 2009.

[14] S. Elbassuoni, M. Ramanath, R. Schenkel, and G. Weikum. Searching RDF graphs with SPARQL and keywords. *IEEE Data Eng. Bull.*, 33(1), 2010.

[15] Y. Fukushige. Representing probabilistic relations in RDF, 2005.

[16] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, 2007.

[17] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum. Yago2: exploring and querying world knowledge in time, space, context, and many languages. In *WWW (Companion Volume)*, pages 229–232, 2011.

[18] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, 2003.

[19] H. Huang and C. Liu. Query evaluation on probabilistic RDF databases. In *WISE*, 2009.

[20] A. Hulgeri and C. Nakhe. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.

[21] Y. Ishikawa, Y. Iijima, and J. X. Yu. Processing spatial range queries for objects with imprecise gaussian-based location information. In *ICDE*, 2009.

[22] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

[23] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic XML. In *VLDB*, 2007.

[24] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *Proc. 11th Int. Conf. on Database Systems for Advanced Applications*, pages 295–309, 2006.

[25] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, 2008.

[26] J. Li, C. Liu, R. Zhou, and W. Wang. Top-k keyword search over probabilistic XML data. In *ICDE*, 2011.

[27] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1), 2009.

[28] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 213–226, 2008.

[29] X. Lian and L. Chen. Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. *The VLDB Journal*, 18:787–808, 2009.

[30] X. Lian and L. Chen. Efficient query answering in probabilistic RDF graphs. In *SIGMOD*, 2011.

[31] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, 2006.

[32] V. Ljosa and A. K. Singh. Top-k spatial joins of probabilistic objects. In *Proc. 24th Int. Conf. on Data Engineering*, pages 566–575, 2008.

[33] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: Top-k keyword query in relational databases. In *SIGMOD*, pages 115–126, 2007.

[34] T. Neumann, M. Bender, S. Michel, R. Schenkel, P. Triantafillou, and G. Weikum. Distributed top-k aggregation queries at large. *Distributed and Parallel Databases*, 26(1), 2009.

[35] T. Neumann and G. Weikum. RDF-3X: a risc-style engine for RDF. *PVLDB*, 1(1), 2008.

[36] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 467–478, 2003.

[37] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, 2007.

[38] L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold. Column-store support for RDF data management: not all swans are white. *PVLDB*, 1(2), 2008.

[39] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. Hambrusch, J. Neville, and R. Cheng. Database support for probabilistic attributes and tuples. In *Proc. 24th Int. Conf. on Data Engineering*, pages 1053–1061, 2008.

[40] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.

[41] M. Stonebraker, D. J. Abadi, A. Batkin, X.Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: a column-oriented dbms. In *VLDB*, 2005.

[42] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A large ontology from wikipedia and wordnet. *Web Semant.*, 6(3), 2008.

[43] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. K., and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proc. 31st Int. Conf. on Very Large Data Bases*, pages 922–933, 2005.

[44] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *ICDE*, 2009.

[45] O. Udrea, V. S. Subrahmanian, and Z. Majkic. Probabilistic rdf. In *IRI*, 2006.

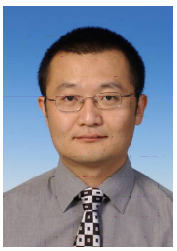
[46] D. Z. Wang, E. Michelakis, M. Garofalakis, and J. Hellerstein. Bayestore: Managing large, uncertain data repositories with probabilistic graphical models. In *VLDB*, 2008.

[47] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *PVLDB*, 1(1), 2008.

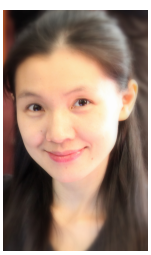
[48] K. Wilkinson. Jena property table implementation. In *SSWS*, 2006.

[49] K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds, and J. Database. Efficient RDF storage and retrieval in Jena2. In *ESWDB*, 2003.

Xiang Lian received the BS degree from the Department of Computer Science and Technology, Nanjing University, in 2003, and the PhD degree in computer science from the Hong Kong University of Science and Technology, Hong Kong. He is now an assistant professor in the Department of Computer Science at the University of Texas - Pan American. His research interests include probabilistic/uncertain data management, probabilistic RDF graphs, inconsistent probabilistic databases, and streaming time series.



Lei Chen received his BS degree in Computer Science and Engineering from Tianjin University, China, in 1994, the MA degree from Asian Institute of Technology, Thailand, in 1997, and the PhD degree in computer science from University of Waterloo, Canada, in 2005. He is now an associate professor in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. His research interests include crowdsourcing on social networks, uncertain and probabilistic databases, Web data management, multimedia and time series databases, and privacy. He is a member of the IEEE.



Zi Huang received the PhD degree in computer science from The University of Queensland, Australia, in 2007. Currently, she is a research fellow with the DKE group, the University of Queensland. Her research interests include multimedia information retrieval, multimedia database management, indexing and query processing, and bioinformatics. She is a member of the IEEE.

SUPPLEMENTARY MATERIALS (APPENDIX)

E1. Example of Feature Score

In the example of Figs. 1(a) and 3(a), assume that 4 query keywords, "John", "law", "Edinburgh", and "assistant_prof" are specified. Then, the feature score $feature(g_1)$ of Graph g_1 is:

$$\begin{aligned} & feature(g_1) \\ =& \frac{1}{2} \times \Big(\frac{1}{(2+1)^2} \times (score_{IR}(law) + score_{IR}(John)) \\ &+ \frac{1}{(5+1)^2} \times (score_{IR}(law) + score_{IR}(Edinburgh)) \\ &+ \frac{1}{(5+1)^2} \times (score_{IR}(law) + score_{IR}(assistant\_prof)) \\ &+ \frac{1}{(5+1)^2} \times (score_{IR}(John) + score_{IR}(Edinburgh)) \\ &+ \frac{1}{(5+1)^2} \times (score_{IR}(John) + score_{IR}(assistant\_prof)) \\ &+ \frac{1}{(4+1)^2} \times (score_{IR}(Edinburgh) + score_{IR}(assistant\_prof)) \Big) \end{aligned}$$

Similarly, the feature score $feature(g_3)$ of Graph g_3 is:

$$\begin{aligned} & feature(g_3) \\ =& \frac{1}{2} \times \Big(\frac{1}{(3+1)^2} \times (score_{IR}(law) + score_{IR}(John)) \\ &+ \frac{1}{(3+1)^2} \times (score_{IR}(law) + score_{IR}(Edinburgh)) \\ &+ \frac{1}{(3+1)^2} \times (score_{IR}(law) + score_{IR}(assistant\_prof)) \\ &+ \frac{1}{(4+1)^2} \times (score_{IR}(John) + score_{IR}(Edinburgh)) \\ &+ \frac{1}{(4+1)^2} \times (score_{IR}(John) + score_{IR}(assistant\_prof)) \\ &+ \frac{1}{(4+1)^2} \times (score_{IR}(Edinburgh) + score_{IR}(assistant\_prof)) \Big) \end{aligned}$$

E2. Example of Entropy Score

As in the example of Figs. 1(a) and 3(a), assume that IR scores of all the 4 keywords are all equal, that is, $score_{IR}(law) = score_{IR}(John) = score_{IR}(Edinburgh) = score_{IR}(assistant\_prof) = C$, where C is a constant. Then, we have $feature(g_1) = 0.262C < feature(g_3) = 0.3075C$, which indicates that g_3 is more preferable than g_1, in terms of the feature score.

Note, however, that the 4 query keywords in g_3 are actually loosely scattered in 4 different RDF entities, that is, ⟨university⟩, ⟨student⟩, ⟨city⟩, and ⟨faculty⟩, respectively, which do not have very strong relationships. For example, the facts that a university has a department of law and a student is called "John" have a weak relationship. Thus, the returned graph g_3 that contains these 4 keywords is not very meaningful, compared with Graph g_1. In contrast, keywords in Graph g_1 are closely related in 3 (fewer) entities (i.e., ⟨novel⟩, ⟨city⟩, and ⟨faculty⟩). Thus, Graph g_1 is actually more likely to be our desirable answer.

From the example above, we can see that solely considering the feature score to rank graphs might lead to keyword search answers (subgraphs) which are not very meaningful, but have high feature scores.

Therefore, in order to return our preferred pg-KWS result, we also introduce the second orthogonal dimension, the entropy score. Specifically, we have: $entropy(g_1) = -(1/2) \times log(1/4) - (1/4) \times log(1/4) - (1/4) \times log(1/4) = log\sqrt{8}$ and $entropy(g_3) = -log(1/4) = log(4)$. Thus, it holds that $entropy(g_1) < entropy(g_3)$, which implies that Graph g_1 has higher rank (i.e., more preferable).

E3. Example of Dominance in 2D Score Space

Based on the dominance definition of 2D score vector in Definition 2.6, we consider the previous example of Graphs g_1, g_3, and g_4 in Figs. 1(a), 3(a), and 3(b), respectively. Note that, Graph g_4 in Fig. 3(b) may indicate the relationship among 4 different persons in social networks with different attributes (e.g., major "law", name "John", etc.).

As shown in Fig. 7, it holds that $feature(g_1) < feature(g_3)$ and $entropy(g_1) < entropy(g_3)$. Since g_1 has lower (inferior) feature score than g_3, but smaller (better) entropy score than g_3, it indicates that g_1 is not dominated by g_3. Similarly, g_1 is not dominated by Graph g_4. Thus, g_1 is our keyword search answer (which is also what we want to obtain).

In contrast, in Fig. 7, Graph g_3 has worse feature score than g_4, and it has the same entropy score as g_4. Therefore, we say that g_4 dominates g_3. In other words, due to the existence of Graph g_4, Graph g_3 cannot be the keyword search answer. Since g_4 is not dominated by both graphs, g_1 and g_4, Graph g_4 is also our pg-KWS answer.

E4. Example of the pg-KWS Problem

Assume that we are given two r-radius graphs, $g_{1,2}$ in Fig. 2 and g_3 in Fig. 3(a), 4 keywords "John", "law", "Edinburgh", and "assistant_prof", and a threshold $\alpha = 0.2$. The possible world that $g_{1,2}$ contains keywords has the existence probability $0.7 \times 0.5 = 0.35$, whereas that of g_3 has probability 1. Then, from Eq. (5), the probability $P(g_{1,2})$ is given by $Pr\{pw(g_{1,2}) | \forall k_i k_i \in pw(g_{1,2})\} \cdot Pr\{g_3 \not\prec pw(g_{1,2})\} = 0.35 \times 1 = 0.35$. Thus, since $P(g_{1,2}) > 0.2$, subgraph $g_{1,2}$ is a pg-KWS answer.

Similarly, we can compute $P(g_3)$, given by $Pr\{g_3\} \cdot Pr\{g_{1,2} \not\prec g_3\} = 1 \cdot \sum_{\forall pw(g_{1,2})} Pr\{pw(g_{1,2}) \not\prec g_3\} = 1 \times 0.35 = 0.35 > 0.2$. Thus, subgraph g_3 is also one of the pg-KWS query answers.

E5. Example of Feature Score Bounds

Assume that $g_{1,2}$ in Fig. 2 has two possible worlds g_1 and g_2 (in Fig. 1). For keyword pair (John, law), as only g_1 contains keywords "John" and "law", we have: $lb\_score_{struc}(John, law, g_{1,2}) = min_{\forall pw(g_{1,2})}\{score_{struc}(John, law, pw(g_{1,2}))\} = \frac{1}{2} \cdot \frac{1}{(2+1)^2} = \frac{1}{18}$; similarly, we have $ub\_score_{struc}(John, law, g_{1,2}) = \frac{1}{18}$. Note that, since the two keywords "John" and "law" only exist in one possible world of subgraph $g_{1,2}$ (i.e., g_1), the two bounds of structural score $score_{struc}(John, law, g_{1,2})$ are thus the same. Structural score bounds of other keyword pairs can be defined analogously. Next, the IR score bounds are given by: $lb\_score_{IR}(k_i, g_{1,2}) = min\{score_{IR}(k_i, g_1), score_{IR}(k_i, g_2)\}$, and moreover $ub\_score_{IR}(k_i, g_{1,2}) = max\{score_{IR}(k_i, g_1), score_{IR}(k_i, g_2)\}$. The feature score bounds can be obtained by Eqs. (10) and (11) w.r.t. ad-hoc query keywords k_i or k_j.

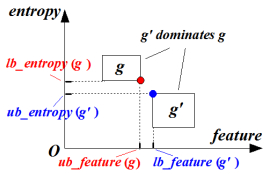


Fig. 14. Illustration of score bound pruning.

TABLE 2
Symbols and descriptions.

Symbol	Description
$\mathcal{G}$	a probabilistic RDF graph database
Σ	RDF data schema
$V(G)$ $(E(G))$	a set of vertices (edges) in probabilistic RDF graphs G
$S(G)$	a set of conditional probability tables (CPTs) in G
$pw(g)$	a possible world of probabilistic RDF graphs g
$Pr\{pw(G)\}$	the appearance probability of a possible world of G
$k_1, k_2, ..., k_q$	a set of q query keywords
g (or g_c)	probabilistic (or certain) graph
$pa(v_i)$	a set of parent vertices of vertex v_i
v_i	the name of a vertex
e_{ij}	the name of a directed edge $\overrightarrow{v_i v_j}$
$l(v_i)$ (or $l(e_{ij})$)	the keyword of a vertex v_i (or a directed edge e_{ij})
α	a probabilistic threshold
$feature(g)$	the feature score of a probabilistic subgraph g
$entropy(g)$	the entropy score of a probabilistic subgraph g
$PWG_i(g)$	the i-th possible world group of subgraph g

E6. Example of Entropy Score Bounds

In Fig. 2, assume that we have 4 query keywords "John", "law", "Edinburgh", and "assistant_prof", which are located in three RDF entities, ⟨novel⟩, ⟨city⟩, and ⟨faculty⟩, in Graph $g_{1,2}$. In 3 possible worlds of $g_{1,2}$, the number of keywords in entity ⟨novel⟩ is within [0, 2], whereas that in entity ⟨city⟩ or ⟨faculty⟩ is in [1, 1]. As a result, we can obtain $lb\_entropy(g_{1,2}) = -max\{log\frac{2}{2+1}, log\frac{1}{1+0}, log\frac{1}{1+0}\} = -log\frac{2}{3}$, and $ub\_entropy(g_{1,2}) = -min\{log\frac{1}{1+2}\} = -log\frac{1}{3}$ (here, we exclude $log\frac{0}{0+1}$, as it is invalid).

E7. Example of Probability Upper Bounds

In the example of Fig. 2, we can divide possible worlds of probabilistic graph, $g_{1,2}$, into two groups, $PWG_1(g_{1,2}) = \{g_1\}$ and $PWG_2(g_{1,2}) = \{g_2\}$, which have existence probabilities $PWG_1(g_{1,2}).p = Pr\{g_1\} = 0.35$ and $PWG_2(g_{1,2}).p = Pr\{g_2\} = 0.12$, respectively. On the other hand, graph g_3 in Fig. 3(a) has one PWG, $PWG_1(g_3)$, with probability 1. Thus, by Eq. (18), we can obtain $UB\_P(g_{1,2}) = PWG_1(g_{1,2}).p \cdot PWG_1(g_3).p \cdot \chi(PWG_1(g_3) \not\prec PWG_1(g_{1,2})) + PWG_2(g_{1,2}).p \cdot PWG_1(g_3).p \cdot \chi(PWG_1(g_3) \not\prec PWG_2(g_{1,2})) = 0.35 \times \chi(g_3 \not\prec g_1) + 0.12 \times \chi(g_3 \not\prec g_2)$.

P1. Proof of Lemma 3.1

Proof. We depict the lemma assumption (i.e., the relationship of score bounds between g and g') in the 2D feature-entropy space of Fig. 14. According to Definition 2.6, we can see that, for any possible worlds of g and g', it holds that $pw(g') \prec pw(g)$. As a result, in Inequality (5), we have: $Pr\{\bigwedge_{\forall g'} g' \not\prec pw(g)\} = 0$, and thus $P(g) = 0 \leq \alpha$. In other words, subgraph g does not satisfy the requirement of probability $P(g) > \alpha$ in Inequality (5), and thus g can be safely pruned, which completes the proof. □

P2. Proof of Lemma 3.2

Proof. We first prove the correctness of Inequality (12). Since it holds that $f_i \in [lb\_f_i, ub\_f_i]$, we have:

$$\begin{aligned} entropy(g) &= -\sum_{i=1}^{m}\left(\frac{f_i}{\sum_{j=1}^{m} f_j}\cdot log\left(1-\frac{\sum_{\forall j\neq i, 1\leq j\leq m} f_j}{\sum_{j=1}^{m} f_j}\right)\right) \\ &\geq -\sum_{i=1}^{m}\left(\frac{f_i}{\sum_{j=1}^{m} f_j}\cdot log\left(1-\frac{\sum_{\forall j\neq i, 1\leq j\leq m} f_j}{ub\_f_i + \sum_{\forall j\neq i, 1\leq j\leq m} f_j}\right)\right) \\ &\geq -\sum_{i=1}^{m}\left(\frac{f_i}{\sum_{j=1}^{m} f_j}\cdot log\frac{ub\_f_i}{ub\_f_i + \sum_{\forall j\neq i, 1\leq j\leq m} lb\_f_j}\right) \\ &= -\left(\sum_{i=1}^{m}\frac{f_i}{\sum_{j=1}^{m} f_j}\right)\cdot \max_i\left\{log\frac{ub\_f_i}{ub\_f_i + \sum_{\forall j\neq i, 1\leq j\leq m} lb\_f_j}\right\} \\ &= -\max_i\left\{log\frac{ub\_f_i}{ub\_f_i + \sum_{\forall j\neq i, 1\leq j\leq m} lb\_f_j}\right\}. \end{aligned}$$

Therefore, for any $i \in [1, m]$, we can obtain a lower bound as shown in the inequality above, which is exactly Eq. (12). The proof of Eq. (13) is similar and omitted. □

P3. Proof of Lemma 3.3

Proof. Since $P(g) \leq UB\_P(g)$ and $UB\_P(g) \leq \alpha$, by inequality transition, we have $P(g) \leq \alpha$. Thus, based on Definition 2.7, g cannot be the pg-KWS answers. □

P4. Proof of Lemma 3.4

Proof. By using the conjunction property in the probability formula, we have the following inequality: $Pr\{g' \not\prec pw(g)\} \geq Pr\{\bigwedge_{\forall g'} g' \not\prec pw(g)\}$. By substituting the inequality above into probability $P(g)$ in Eq. (5), we can derive its upper bound, $UB\_P(g)$, that is, $P(g) \leq \sum_{\forall pw(g)} Pr\{pw(g)\} \cdot Pr\{g' \not\prec pw(g)\}$, which is exactly the upper bound $UB\_P(g)$ in Eq. (14). □

P5. Proof of Lemma 3.5

Proof. Since $pw(g) \in \bigcup_i PWG_i(g)$ and $pw(g') \in \bigcup_j PWG_j(g')$ hold, from Eq. (14), we have:

$$\begin{aligned} P(g) &\leq \sum_{\forall pw(g)} Pr\{pw(g)\} \cdot \sum_{\forall pw(g'), pw(g') \not\prec pw(g)} Pr\{pw(g')\} \\ &\leq \sum_i PWG_i(g).p \cdot \sum_{\forall pw(g'), pw(g') \not\prec PWG_i(g)} Pr\{pw(g')\} \\ &\leq \sum_i PWG_i(g).p \cdot \sum_j \left(PWG_j(g').p \cdot \chi(PWG_j(g') \not\prec PWG_i(g))\right), \end{aligned}$$

which is exactly the upper bound $UB\_P(g)$ in Eq. (18). □

Q1. Query Procedure of Fig. 6

We maintain an initially empty *minimum heap* $\mathcal{H}$ (i.e., each time an entry with the minimum key is popped out from heap) whose entry is in the form (e, key) (line 1), where e can be either a node or a graph, and key of e is given by $key(e) = lb\_entropy(e) - ub\_feature(e)$ (note: the key definition here is similar to that in [36]). Intuitively, smaller key value of a node/graph has less chance to be dominated by others in the 2D score space. Thus, by using the min-heap $\mathcal{H}$, we always obtain an entry with the smallest key, and visit its corresponding node/graph e that is most likely to contain/be pg-KWS graph answers. In addition to heap $\mathcal{H}$, we also keep a candidate set, S_{cand}, to store graph candidates (line 2).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2365791, IEEE Transactions on Knowledge and Data Engineering

We traverse the tree index $\mathcal{I}$ by first inserting the root $root(\mathcal{I})$ into heap $\mathcal{H}$ (line 3). Then, each time we pop out the top entry (e, key) from heap $\mathcal{H}$ (lines 4-5). If entry e cannot be pruned by score bound or probabilistic pruning methods, w.r.t. candidates (and their PWGs) in S_{cand}, then we will check its children (or PWGs in a probabilistic r-radius graph) (lines 6-17).

When entry e is a probabilistic r-radius graph g, we apply the probabilistic pruning by considering PWGs of graph g. If g still cannot be ruled out, then g is a candidate graph and we add it to the candidate set S_{cand} (lines 7-9). When entry e is either a leaf or a non-leaf node, we check its children, graphs g or entries e_x, respectively (lines 10-13 and 14-17). For each g or e_x, we verify whether or not it can be safely pruned by score bound pruning (i.e., Lemma 3.1 for pruning graph and Lemma 4.1 for pruning node) or probabilistic pruning methods (i.e., Lemma 3.3). If a graph g cannot be pruned, we will insert entry (g, $key(g)$) into the heap $\mathcal{H}$ for further checking (line 13). Similarly, if an entry $e_x \in e$ cannot be pruned, we will insert entry (e_x, $key(e_x)$) into heap $\mathcal{H}$ (line 17). This process is repeated (lines 5-17) until heap $\mathcal{H}$ becomes empty (line 4). Finally, we refine graph candidates g in S_{cand} by calculating probabilities $P(g)$ as given in Definition 2.7, and report the actual pg-KWS query answers (line 18).

R1. Pruning Power vs. RDF Graph Size N

As reported in Fig. 15, the pruning power of our pg-KWS approach (i.e., the percentage of the subgraph candidates that are pruned by our pruning methods) are very high (i.e., about 97% $\sim$ 99.9%) for different graph sizes N from $10K$ to $1M$. These results show the good scalability of our pg-KWS approach w.r.t. RDF graph size N.

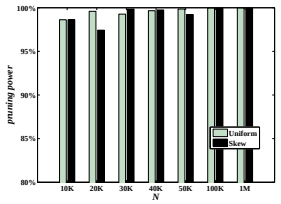
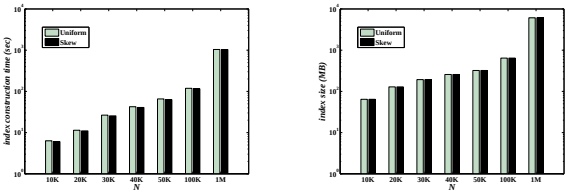


Fig. 15. Pruning power of pg-KWS vs. N.



(a) Index construction time (b) Index size

Fig. 16. Index Construction Time and Size vs. N.

R2. Index Construction Time and Index Size vs. RDF Graph Size N

We also present the total time cost of the index construction, including the time of offline pre-computation and building a tree index, and the index size in Fig. 16, for different data size N from $10K$ to $1M$, where other parameters are set to default values. From the figure, we can see that both the index construction time and index size increase when N becomes larger. When $N = 1M$, the time of building an index (of size about 6099 MB) over the pre-computed data is about 17.3 min, which indicates that the index construction time is scalable w.r.t. N.